

Philippe P. Pébay (Ed.)

Proceedings of the
**15th International
Meshing Roundtable**



 Springer

Proceedings of the 15th International Meshing Roundtable

Philippe P. Pébay (Ed.)

Proceedings of the 15th International Meshing Roundtable

 Springer

Philippe P. Pébay

Sandia National Laboratories
PO Box 969, MS 9051
Livermore, CA 94550 U.S.A.
E-mail: pppebay@ca.sandia.gov

Library of Congress Control Number: 2006930257

ISBN-10 3-540-34957-X Springer Berlin Heidelberg New York

ISBN-13 978-3-540-34957-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable for prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com

© Springer-Verlag Berlin Heidelberg 2006

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: by the author and techbooks using a Springer L^AT_EX macro package
Cover design: *design & production* GmbH, Heidelberg

Printed on acid-free paper SPIN: 11588672 89/techbooks 5 4 3 2 1 0

Preface

The papers in this volume were selected for presentation at the 15th International Meshing Roundtable, held September 17–20, 2006 in Birmingham, Alabama, U.S.A.. The conference 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 15 consecutive years, the International Meshing Roundtable has become recognized as an international focal point annually attended by researchers and developers from dozens of countries around the world.

The 15th International Meshing Roundtable consists of technical presentations from contributed papers, keynote and invited talks, short course presentations, and a poster session and competition. The Program Committee would like to express its appreciation to all who participate to make the IMR a successful and enriching experience.

The papers in these proceedings were selected from among 42 submissions by the Program Committee. Based on input from peer reviews, the committee selected these papers for their perceived quality, originality, and appropriateness to the theme of the International Meshing Roundtable. The Program Committee would like to thank all who submitted papers. We would also like to thank the colleagues who provided reviews of the submitted papers. The names of the reviewers are acknowledged in the following pages.

As Program Chair, I would like to extend special thanks to the Program Committee and to the Conference Coordinators for their time and effort to make the 15th IMR another outstanding conference.

Sandia National Laboratories
15th IMR Chair
July 2006

Philippe P. Pébay

Reviewers

<u>Name</u>	<u>Affiliation</u>
Aftosmis, Michael	NASA Ames, USA
Berkhahn, Volker	U. of Hanover, Germany
Blades, Eric	Mississippi State U., USA
Brewer, Michael	Sandia Nat. Labs, USA
Burgreen, Greg	Mississippi State U., USA
Carbonera, Carlos	Jostens, Inc., USA
Chawner, John	Point Wise, USA
Chernikov, Andrey	C. of William and Mary, Williamsburg, USA
Chrisochoides, Nikos	C. of William and Mary, Williamsburg, USA
Dillavou, Mark	U. of Alabama, Birmingham, USA
Hanks, Byron	Sandia Nat. Labs, USA
Ito, Yasushi	U. of Alabama, Birmingham, USA
Jiao, Xiangmin	U. of Illinois, U-C., USA
Knupp, Patrick	Sandia Nat. Labs, USA
Kraftcheck, Jason	U. of Wisconsin, USA
Labelle, Francois	U. of California, Berkley, USA
Lipnikov, Konstantin	Los Alamos Nat. Labs, USA
Marcum, David	Mississippi State U., USA
Michal, Todd	Boeing, USA
Ollivier-Gooch, Carl	UBC, Canada
Owen, Steve	Sandia Nat. Labs, USA
Pav, Steven	U. of California, San Diego, USA
Pebay, Philippe	Sandia Nat. Labs, USA
Rajagopalan, Krishnakumar	PDC, USA
Schneiders, Robert	MAGMA Giessereitechnologie GmbH, Germany
Shepherd, Jason	Sandia Nat. Labs, USA
Shewchuk, Jonathan	U. of California, Berkley, USA
Shih, Alan	U. of Alabama, Birmingham, USA
Si, Hang	WIAS, Germany
Simpson, Bruce	U. of Waterloo, Canada
Sjaardema, Gregory	Sandia Nat. Labs, USA
Steinbrenner, John	Point Wise, USA
Stephenson, Michael	Stephenson & A., USA
Taghavi, Reza	Simulations Works Inc., USA
Tautges, Tim	Sandia Nat. Labs, USA
Thompson, David	Mississippi State U., USA
Thompson, David	Sandia Nat. Labs, USA
Yamakawa, Soji	Carnegie Mellon U., USA

15th IMR Conference Organization

- Committee:**
- The late Timothy J. Baker**
Princeton University, Princeton, NJ
- Philippe Pébay (Chair)**
Sandia National Labs, Albuquerque, NM
pppebay@ca.sandia.gov
- Alan Shih (Papers Subcommittee Chair)**
University of Alabama at Birmingham, Birmingham, AL
ashih@uab.edu
- Michael Brewer**
Sandia National Labs, Albuquerque, NM
mbrewer@sandia.gov
- David Marcum**
Mississippi State Univeristy/SimCenter, Starkville, MS
Marcum@simcenter.msstate.edu
- Todd Michal**
The Boeing Company, St. Louis, MO
todd.r.michal@boeing.com
- Steven Pav**
Nellcor, Pleasanton, CA
Steven.Pav@tycohealthcare.com
- Krishnakumar Rajagopalan**
Program Development Company, White Plains, NY
Krishna@gridpro.com
- Coordinators:**
- Lynn Janik Washburn**
Sandia National Labs, Albuquerque, NM
lajanik@sandia.gov
- Jacqueline A. Finley**
Sandia National Labs, Albuquerque, NM
jafinle@sandia.gov
- Web Designer:**
- Bernadette Watts**
Sandia National Labs, Albuquerque, NM
bmwatts@sandia.gov
- Web Site:**
- <http://www.imr.sandia.gov>

Contents

Session 1A 2D Meshing

Non-Local Topological Clean-Up <i>Guy Bunin</i>	3
Quad-Dominant Mesh Adaptation Using Specialized Simplicial Optimization <i>Ko-Foa Tchou and Ricardo Camarero</i>	21
Mesh Modification Under Local Domain Changes <i>Narcís Coll, Marité Guerrieri and J. Antoni Sellarès</i>	39
The Cost of Compatible Refinement of Simplex Decomposition Trees <i>F. Betul Atalay and David M. Mount</i>	57

Session 1B Applications

Patient-Specific Vascular NURBS Modeling for Isogeometric Analysis of Blood Flow <i>Yongjie Zhang, Yuri Bazilevs, Samrat Goswami, Chandrajit L. Bajaj and Thomas J.R. Hughes</i>	73
Rapid Meshing of Turbomachinery Rows Using Semi-Unstructured Conformal Grids <i>Manuel A. Burgos, Roque Corral, Jaime Fernández-Castañeda and Carlos López</i>	93
Hybrid Mesh Generation for Viscous Flow Simulation <i>Yuanli Wang and Samuel Murgie</i>	109

**A Remeshing Procedure for Numerical Simulation
of Forming Processes in Three Dimensions**
L. Giraud-Moreau, H. Borouchaki and A. Cherouat 127

Session 2 Mesh Adaptation

**A Solution-Based Adaptive Redistribution Method
for Unstructured Meshes**
Yasushi Ito, Alan M. Shih, Roy P. Koomullil and Bharat K. Soni 147

**Analysis of Hessian Recovery Methods
for Generating Adaptive Meshes**
Konstantin Lipnikov and Yuri Vassilevski 163

**Anisotropic Mesh Adaptation for Evolving Triangulated
Surfaces**
Xiangmin Jiao, Andrew Colombi, Xinlai Ni and John C. Hart 173

**Multi-Dimensional Continuous Metric
for Mesh Adaptation**
Frédéric Alauzet, Adrien Loseille, Alain Dervieux, Pascal Frey 191

**How Efficient are Delaunay Refined Meshes?
An Empirical Study**
Bruce Simpson 215

Session 3A Mesh Optimization

**Small Polyhedron Reconnection: A New Way
to Eliminate Poorly-Shaped Tetrahedra**
Jianfei Liu and Shuli Sun 241

Optimal Mesh for P_1 Interpolation in H^1 Seminorm
Jean-François Lagüe and Frédéric Hecht 259

**Mesh Smoothing Based on Riemannian Metric
Non-Conformity Minimization**
*Yannick Sirois, Julien Dompierre, Marie-Gabrielle Vallet
and François Guibault* 271

**On Asymptotically Optimal Meshes by Coordinate
Transformation**
Guillermo D. Cañas and Steven J. Gortler 289

Session 3B Meshing Algorithms

- High Quality Bi-Linear Transfinite Meshing
with Interior Point Constraints**
Nilanjan Mukherjee 309
- Implementation in ALBERTA of an Automatic Tetrahedral
Mesh Generator**
*R. Montenegro, J.M. Cascón, J.M. Escobar, E. Rodríguez
and G. Montero* 325
- Sparse Voronoi Refinement**
Benoît Hudson, Gary Miller and Todd Phillips 339

Session 4 Geometry

- Volume and Feature Preservation
in Surface Mesh Optimization**
Xiangmin Jiao 359
- On The Use of Loop Subdivision Surfaces
for Surrogate Geometry**
Per-Olof Persson, Michael J. Aftosmis and Robert Haimes 375
- Surface Mesh Generation for Dirty Geometries
by Shrink Wrapping using Cartesian Grid Approach**
*Y.K. Lee, Chin K. Lim, Hamid Ghazialam, Harsh Vardhan and Erling
Eklund* 393
- A Hole-Filling Algorithm for Triangular Meshes
Using Local Radial Basis Function**
John Branch, Flavio Prieto and Pierre Boulanger 411

Session 5A Hexahedral Meshing

- A Constructive Approach
to Constrained Hexahedral Mesh Generation**
Carlos D. Carbonera and Jason F. Shepherd 435
- Automatic Hexahedral Mesh Generation
with Feature Line Extraction**
*Masayuki Hariya, Ichiro Nishigaki, Ichiro Kataoka
and Yoshimitsu Hiro* 453

Unconstrained Paving and Plastering: Progress Update

*Matthew L. Staten, Robert A. Kerr, Steven J. Owen
and Ted D. Blacker* 469

**An Automatic and General Least-Squares Projection
Procedure for Sweep Meshing**

Xevi Roca and Josep Sarrate 487

Session 5B Delaunay Meshing

On Refinement of Constrained Delaunay Tetrahedralizations

Hang Si 509

**Smooth Delaunay–Voronoi Dual Meshes
for Co-Volume Integration Schemes**

*Igor Sazonov, Oubay Hassan, Kenneth Morgan
and Nigel P. Weatherill* 529

A Study on Delaunay Terminal Edge Method

Maria-Cecilia Rivara 543

**Generalized Delaunay Mesh Refinement: From Scalar
to Parallel**

Andrey N. Chernikov and Nikos P. Chrisochoides 563

Index of Authors and Co-Authors 581

Index by Affiliation 583

Session 1A

2D Meshing

Non-Local Topological Clean-Up

Guy Bunin

Department of Physics, Technion, Haifa 32000, Israel
buning@tx.technion.ac.il

Abstract. A new approach to topological clean-up of 2D meshes is presented. Instead of searching for patterns in a mesh and replacing them as in other methods, the proposed method replaces a region in a mesh only according to the boundary of that region. This simplifies the classification of the different cases, and allows mesh modification over greater regions in the mesh. An algorithm for quadrilateral meshes utilizing this approach is presented in detail, and its effects on example problems are shown.

1. Introduction

Topological clean-up is the name given to methods aimed to improve the quality of a mesh, by changing its connectivity [1,2,3,4,6]. It is usually carried out after a mesh has been generated by some meshing algorithm. The goal of the clean-up is to decrease the number of nodes that are attached to too few or too many cells. In an all-quadrilateral mesh, if there are nodes attached to more or less than 4 cells, some cells will be forced to have inner angles different than 90 degrees. Decreasing the number of irregular nodes could therefore improve the quality of the mesh.

The methods presented in the literature on the subject are usually based on predefined "cases" or "patterns". Cases are specific configurations of the mesh connectivity that, once found in the mesh, are modified according to the case found. Because the number of possible configuration of a mesh increases dramatically with the configuration size, the cases are *local*, spanning a region of a few cells at most. By applying local clean-up operations successively on a mesh, many problematic configurations can be resolved. There are, however, possible mesh improvements that are not covered by these methods.

In this paper a new clean-up method is proposed. In this approach, the configurations are regions of the mesh classified according to the *boundary* of the region. The mesh connectivity is modified by replacing the mesh of the region with a different mesh connectivity, that was created by "topologically meshing" the interior of the boundary of that region. In other words, a new mesh connectivity conforming to the boundary of the replaced region is created, and if this new connectivity has a better topological quality, it can replace the existing mesh of the region. The method can be applied to improve the mesh structure even if irregular nodes are not close to each other, thus allowing the clean-up of cases not specified by previous methods.

The method presented, like other topological clean-up methods, does not take the vertex locations into account. Therefore, if applied without any restrictions, it might reduce the geometric quality of the mesh. This is especially true near mesh boundaries, since boundary vertices cannot be moved, and the amount of possible mesh smoothing is more limited. In such cases it may be better to use techniques that are more geometric in nature. For further discussion see section 2.5, where criteria for allowing clean-up operations are suggested.

The paper is organized as follows. In chapter 2 the algorithm is described. Chapter 3 presents examples of clean-up results. Chapter 4 presents conclusions and possible directions for future research.

2. Algorithm

2.1 Algorithm Overview

In the method suggested, every topological clean-up operation consists of 3 steps, see figure 1. First, a simply-connected part of the mesh is selected (Fig. 1, (1)). Then, the inside of the loop surrounding the chosen part of the mesh is "topologically-meshed", i.e. a new mesh-connectivity is created (Fig. 1, (2)). This "meshing" phase is based on the loop structure, which is determined by the first layer of cells outside the loop. Finally, if the new topological-mesh meets a number of conditions, the principal condition being the improvement of the topological structure, the existing part of the mesh is replaced by the new mesh (Fig. 1, (3)).

After a replacement takes place the new mesh nodes have no location. The geometric stage, of assigning coordinates to the vertices can be done as a fourth step after the replacement, or perhaps at once after all the topo-

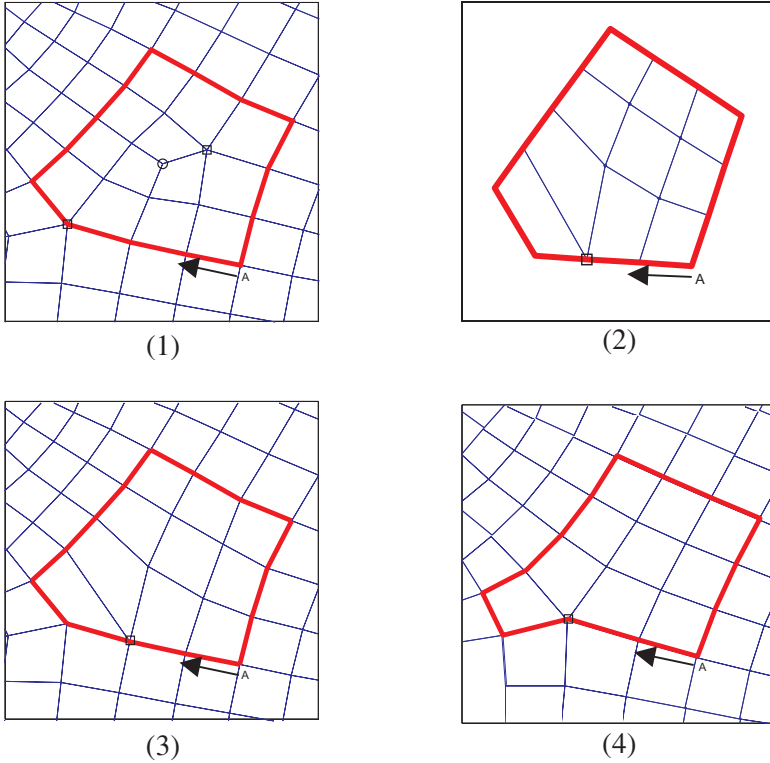


Fig. 1. The steps of a single clean-up operation. (1) A loop surrounding a part of the mesh is chosen. (2) The loop is given a different topological-mesh (connectivity only), based on the topological structure of the loop. (3) The mesh inside the loop is replaced by the new mesh created in step 2. (4) The new nodes are given coordinates, and the whole mesh is smoothed. (This part can also be done after all clean-up operations are over).

logical clean-up process has ended (Fig. 1, (4)). The problem of assigning locations to the new vertices will not be addressed here. For research on the subject see [7] and references therein.

In what follows only the case of quadrilateral meshes is addressed, though the basic principle may be extended to other cases, such as triangular and mixed meshes.

2.2 Definitions

- Node Valence – the number of cells (or edges) incident on the vertex.

- Clean Mesh – a mesh in which all nodes are of valence 4.
- Defect – a node of valence other than 4¹.
- Loop – a closed path traversing edges in a mesh.
- Topological Outer Angle – a closed loop in a mesh divides the mesh into two parts: the cells inside the loop, and the cells outside the loop. A vertex which is on the loop itself will therefore be incident upon cells inside and outside the loop. The outer angle of a vertex on a loop is calculated to be (see Figure 2):

$$\begin{aligned} \langle \text{Topological Outer Angle} \rangle = & \quad (1) \\ \langle \text{number of outside cells incident} \rangle - 2 \end{aligned}$$

If the vertex is on the mesh boundary, a factor of $\alpha/90^\circ$, α being the geometric outside angle of the boundary, is added to the right hand side of equation (1), to account for the lack of cells outside the mesh.

- Convex Loop – a convex loop is a loop whose nodes all have a non-negative topological outer angle.
- Corner of a Convex Loop – a corner of a convex loop is a vertex with a positive outer angle.
- Side of a Convex Loop – the edges of a loop between two given corners.

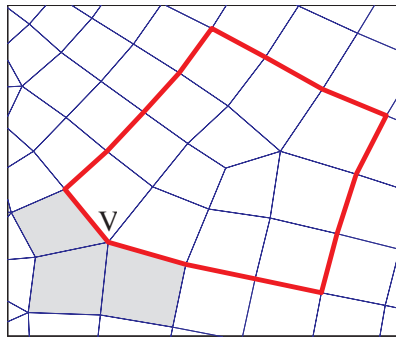


Fig. 2. The vertex V in the figure is incident upon 3 cells that are outside the loop. According to Eq. (1), its topological outer angle is $3 - 2 = +1$.

¹ Some authors refer to it as an irregular node.

2.3 Remeshing a Convex Loop

This section describes a way of topologically meshing a convex loop. That is, a new mesh connectivity inside a loop is created. This new connectivity is used in step 2 of the algorithm, after a loop has been chosen.

The input to the algorithm is the loop, and the topological outside angle (TOA) of each vertex on the loop. Since it is a convex loop (see definition above) the TOA is non-negative. The following algorithm has an even more restricted input domain: it only deals with loops that have TOA's of 0 or +1. This restriction actually limits the input to a topological analog of a polygon: there are N sides and N corners (see definition above), each corner has $\text{TOA} = +1$.

One Defect Solution

There is a class of convex loops that can be meshed with only one defect in the mesh. Figure 3 shows such a mesh, for a convex loop of 3 sides. As can be seen in the figure, certain constraints on the lengths of the sides should hold. We will discuss these relations now.

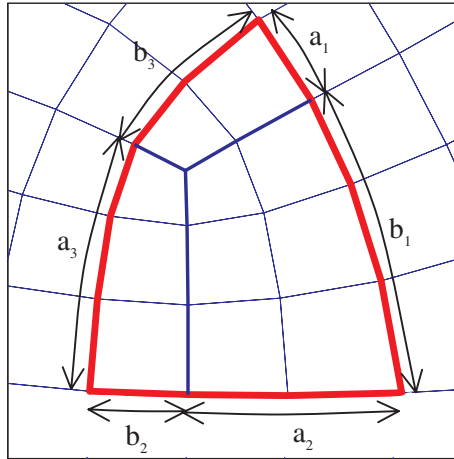


Fig. 3. A single-defect mesh inside a convex loop, for $N = 3$. The mesh can be viewed as composed of N structured meshes (clean meshed with 4 sides). The seaming of the loop constrains the side lengths. For $N=3$: $b_1 = a_3$; $b_2 = a_1$; $b_3 = a_2$.

A single defect mesh of N sides can be viewed as N structured meshes seamed together. Every side of the loop is composed of 2 sides of the 2 logical meshes, see figure 3. Let l_i , $I = 1 \dots N$ be the length of the loops sides, and a_i, b_i the lengths of the 2 parts of each l_i . Then $a_i + b_i = l_i$. Moreover, due to the seaming of the sides of the logical meshes, some parts must have equal lengths, for example $b_1 = a_3, b_2 = a_4$, etc., or in general: $b_i = a_{(i+1) \bmod N+1}$. For a loop with N sides, the constraints can be summarized in the following matrix equation:

$$M \begin{pmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_1 \\ \vdots \\ l_n \end{pmatrix} \quad (2)$$

Where $M = \begin{pmatrix} -D_2 & I \\ I & I \end{pmatrix}$ is a $2N \times 2N$ matrix, and D_2 is the $N \times N$ identity matrix, with columns $N-1, N$ brought to the beginning (becoming columns 1,2). E.g. for $N = 3$, D_2 is:

$$D_2 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad (3)$$

And for $N = 5$ D_2 is:

$$D_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad (4)$$

² A structured mesh is a clean mesh with 4 sides.

When N is not a multiple of 4, the matrix M can be inverted. Then the matrix equation (2) can be solved. If the a_i and b_i of the solution vector are all non-negative integers, this solution represents a single defect mesh. If not, the loop cannot be meshed with a single defect.

Many Defects

This section describes an algorithm for meshing loop that cannot be meshed with a single defect. We limit ourselves to loops having 3, 4 or 5 sides. We first define a "penalty factor" for loops. For loop with 3 or 5 sides:

$$P = \min(0, \text{abs}(\min_i(a_i, b_i))) \quad (5)$$

where a_i, b_i are the solutions to equation (2) for the loop. A loop with penalty 0 can be meshed with one defect. A loop with positive penalty can only be meshed with more defects³.

For 4-sided loops we define the penalty as follows. Let (a, b, c, d) be the side lengths of the loop. Then we define the penalty of the loop using the penalty defined for 3-sided loops.

$$P_4 = 1 + \max(P_3(a, c, \text{abs}(d - b)), P_3(b, d, \text{abs}(c - a))) \quad (6)$$

Here P_3, P_4 are the 3-sided loop and 4-sided loop penalties, respectively. The rationale behind this definition will be clear below, when we describe the meshing algorithm of 4-sided loops.

We now describe the meshing algorithm. The algorithm presented is recursive. For the case of a loop with 3 sides, the penalty of the loop is computed, Eq. (5). If $P=0$, then the loop can be meshed with one defect, completing the meshing of this loop. Otherwise, one of the sides is "broken" into two, and the resulting 4-sided loop is meshed recursively. This effectively puts a 3 defect into the mesh, see figure 4,A. The location of the "brake" is chosen according to the penalty of the 4-sided loop it would create.

For a 5-sided loop, if the penalty of the loop is 0, the loop is meshed with one defect. If not, the algorithm joins two adjacent sides of the loop

³ Actually, it can be proved that an odd number of defects with valence 3 or 5 is required.

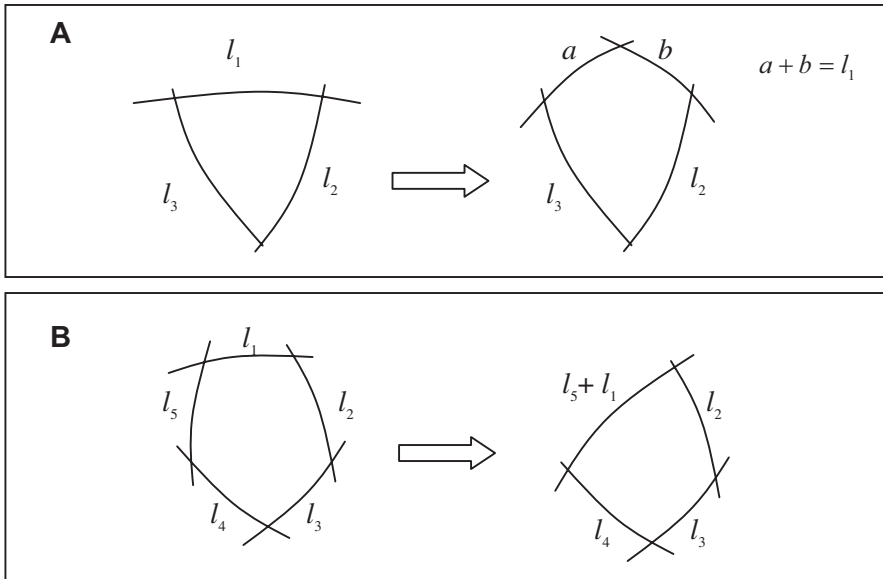


Fig. 4. Meshing of 3-sided (A) and 5-sided (B) loops that cannot be meshed with one defect. The loops are converted to 4-sided loops and meshed as such.

into one, and sends the result to be meshed as a 4-sided loop, see figure 4,B. The joining effectively creates a 5-defect. The sides to be joined are chosen according to the penalty of the 4-sided loop that would be created.

For meshing a 4-sided loop, let a, b, c, d be the side lengths of the loop. If $a = c$ and $b = d$, then the loop can be meshed with a clean mesh. Otherwise, the algorithm adds rows until a triangle is formed, see figure 5. Since the rows stretch between a pair of opposite sides, there are two possible directions for adding the rows, see figure 5A,B. The direction of adding the rows is chosen according to the penalty of the resulting triangle. Note that, in the side that was closed, a 5-defect is formed. This meshing method is the reason for the penalty definition for 4-sided loop: the penalty is just one plus the better of the penalties of the triangles that can be formed by adding rows. If the penalty of a 4-sided loop is 1, it can be meshed with 2 defects. If the penalty is higher, more defects are required.

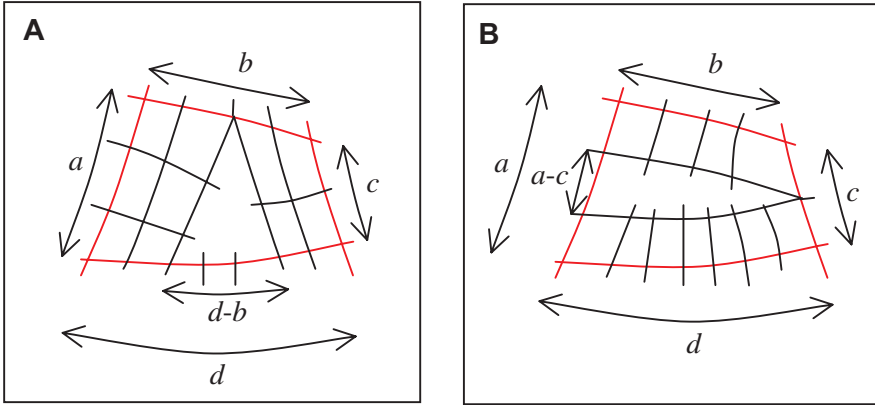


Fig. 5. Meshing of a 4-sided loop. In the case shown $a = 3; b = 4; c = 2; d = 7$. The 2 possible directions for adding rows are presented. The 3-sided loop left in case A takes a penalty of 0. The penalty for the loop in case B is 1.

2.4 Choosing the Loop to Remesh

The first step in a single clean-up operation is choosing a region to remesh. This region should preferably contain defects, so that a clean-up operation may be possible. There are various ways of finding loops around defects. Presented here is a very simple way; there are other, perhaps better ways.

The algorithm starts from a single defect in the mesh, called the "seed defect". A single layer of cells is added by adding all cells that are incident upon this defect. This group of cells will be called a "blob". If the loop surrounding this blob is not a convex loop (i.e. some outer angles are negative, see figure 6), then cells incident on the vertices with negative outer-angle are added to the blob. This process is repeated until the loop is convex. The loop is then sent to the topological meshing algorithm described in section 2.3. If a mesh replacement has been made, a new seed defect is found, and the process is repeated from the beginning. Otherwise, if no mesh replacement has been made, the loop is grown further by adding the cells incident on the blob, and making the loop convex again. This process is repeated until a replacement has been made, or it is certain that no replacements will be made by extending the blob further.

The seed defects are chosen by preferring defects that are furthest away from the boundary. If no replacement was made using the innermost defect as a seed defect, the process is repeated with the second innermost one, and so on. Once a mesh replacement made, the process starts again from the innermost defect.

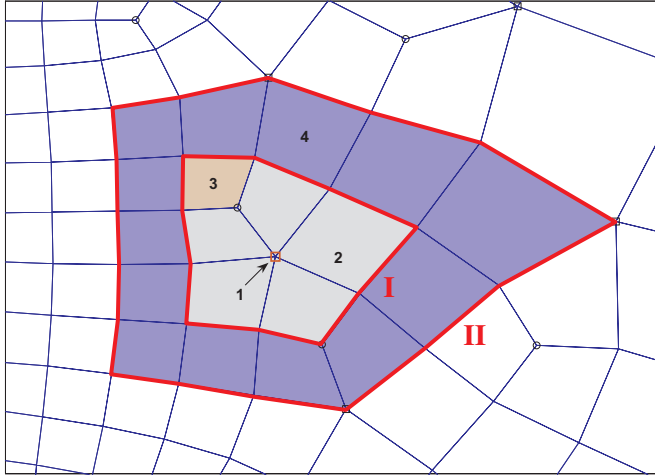


Fig. 6. Choosing a mesh region to be replaced. **1.** The seed defect. **2.** The cells incident on the seed defect. **3.** The cell added to the first layer (2) to make its loop convex. The first loop (designated as I) surrounds cells of (2)+(3). **4.** If no mesh replacement is made for loop I, the blob is extended. Loop II surrounds the larger blob.

2.5 Criteria for Allowing the Replacement

Certain criteria must be met for the mesh replacement to be held. The first, obvious demand is that the replacement will improve the topological quality of the mesh. The topological quality is defined as the sum of how much the valence of the nodes varies from 4. A mesh replacement will only be carried out if this sum is lower after the replacement.

Also, there are convex loops which the topological-meshing algorithm described above cannot mesh, such as a loop containing a node with topological outer angle greater than $+1$. Obviously, in such a case no replacement is made.

A topological clean-up process does not take into account the geometry of the mesh. This is less problematic when making mesh replacements far from the boundary of the mesh, where a smoothing process can move the mesh vertices to a great extent without creating low quality cells. Close to the mesh boundary, however, the mesh has less flexibility due to the fixed location of the boundary vertices, and the replacement is more risky. The following conditions help avoid creating poor cells.

A mesh replacement is allowed only if:

1. The blob radius (defined by the number of times it was grown, see section 2.4) is smaller than the distance of the closest defect in the blob to the boundary.

2. The new defects (if any) in the replaced mesh are further from the boundary than the defects in the existing blob.

The conditions above are of heuristic nature, but some intuitive reasoning can be given in their favor. The first condition stems from the idea that big modifications of the mesh (modifications over great distances) require more flexibility of the mesh that exists only far from the mesh boundary. The second condition reflects the importance of well aligned rows along the boundaries, as well as a reasoning similar to the one done for the first condition, to say that the flexibility of the mesh far from the boundary can help in reducing the effect of the defects on mesh quality.

3. Examples

In the examples below, we show the results of applying the clean-up algorithm on sample meshes. The input meshes used were created with the MSC/PATRAN software. To allow a cleaner comparison, all meshes were smoothed with Laplacian smoothing.

It is worth while noting that the mesh generator used *already contains* its own topological clean-up stage [2], so the comparison presented here shows mesh improvement on-top of what was achieved using the clean-up described in [2].

In the comparisons shown below 2 different quality measures are used. The shape measure β [4,5] is used to evaluate the quality of a single cell. As in [5], it is defined here as the minimum of the shape metrics of the 4 triangles that can be formed by the vertices of the quadrilateral. The purpose of the second measure is to evaluate cell size transitions. A gradual size transition is an important characteristic of a mesh. To quantify this property of a mesh we define a size transition metric as follows. For every pair of neighboring cells (cells that share an edge) the area of the larger cell is divided by the area of the smaller one. The size transition metric of the mesh is the average of this quotient over all pairs of neighboring cells. It is always larger or equal to 1. A gradual size transition will be characterized by a metric close to 1.

The first example shows a uniform size mesh. Figures 7-8 show the mesh before and after clean-up. Mesh defects are marked by squares or circles, according to node valence. Figure 9 compares the shape quality histograms. As can be seen, there is a great reduction in the number of cells with shape quality $\beta < 0.7$. This is directly connected to the reduction in the number of defects, as is shown in figure 10, in which the mesh

before the clean-up is drawn again, with cells having $\beta < 0.7$ filled in black. As can be seen, amongst the cells surrounding a node with valence 3 there is typically one with $\beta < 0.7$. Table 1 presents a comparison of mesh characteristics for this example.

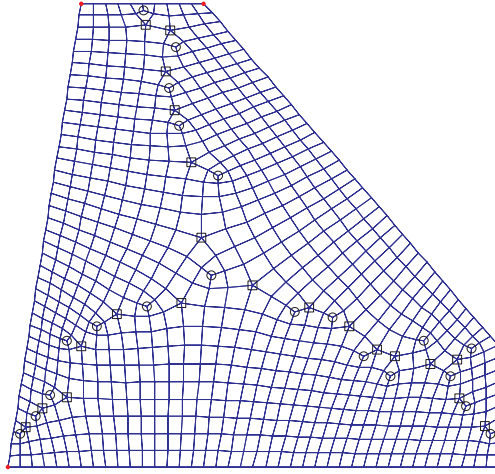


Fig. 7. Mesh of example 1 before clean-up.

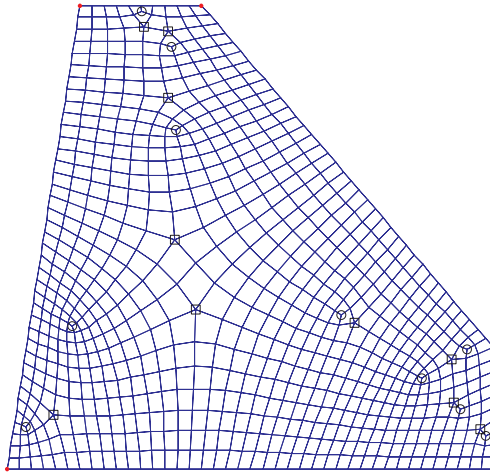


Fig. 8. Mesh of example 1 after clean-up.

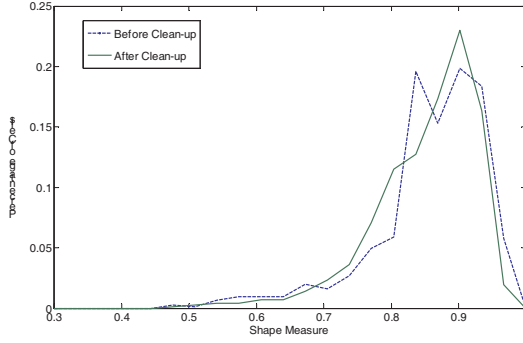


Fig. 9. Comparison of shape quality distributions in example 1 before and after clean-up.

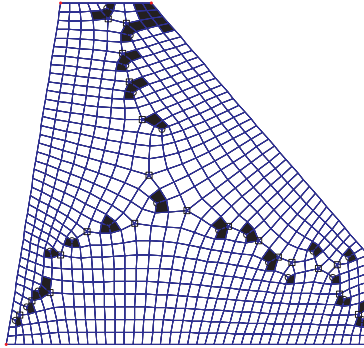


Fig. 10. Mesh of example 1 before clean-up; cells with $\beta < 0.7$ are filled.

Table 1. Mesh characteristics of example 1.

	Before Clean Up	After Clean Up
Number of Cells	745	721
Number of Defects	42	20
Average Metric β	0.857	0.852
Minimal Metric β	0.478	0.464
Number of Cells with $\beta < 0.7$	52	34
Size Transition Metric	1.115	1.119

The second example shows a mesh that contains a large change in mesh size, due to variation in boundary edge length, see Figs. 11,12. A new feature seen here is an improvement in the size transition. The size transition metric is reduced from 1.223 to 1.166.

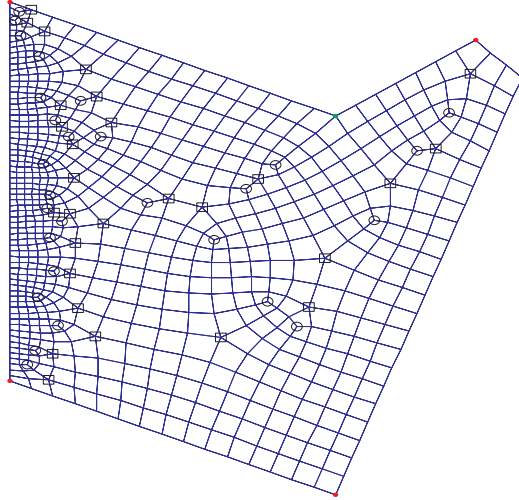


Fig. 11. Mesh of example 2 before clean-up.

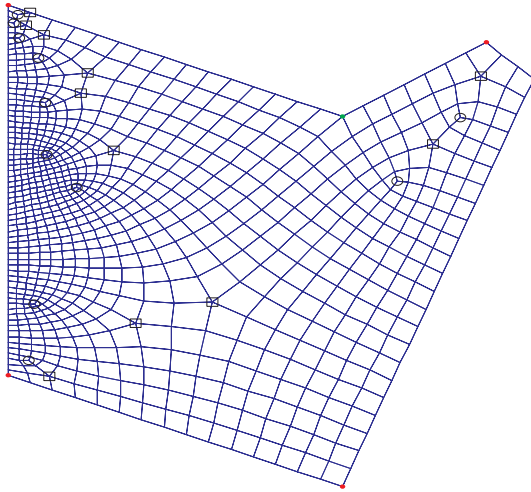


Fig. 12. Mesh of example 2 after clean-up.

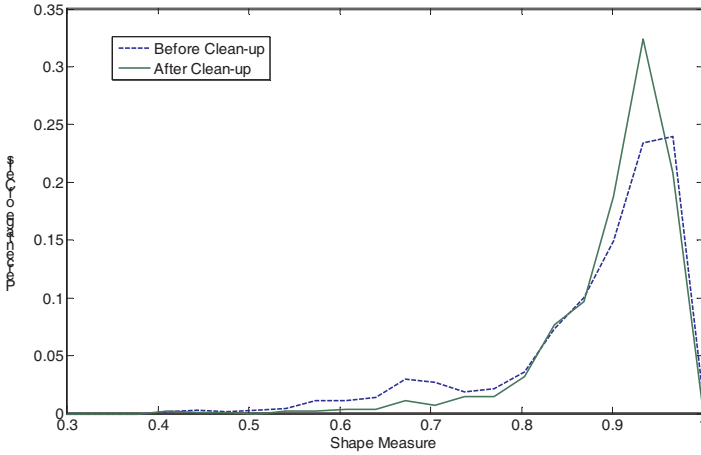


Fig. 13. Comparison of shape quality distributions in example 1 before and after clean-up.

Table 2. Mesh characteristics of example 2.

	Before Clean Up	After Clean Up
Number of Cells	696	802
Number of Defects	56	22
Average Metric β	0.881	0.900
Minimal Metric β	0.410	0.408
Number of Cells with $\beta < 0.7$	61	28
Size Transition Metric	1.223	1.166

The third example shows the action of the algorithm on a mesh with concave boundaries, and an interior boundary (hole).

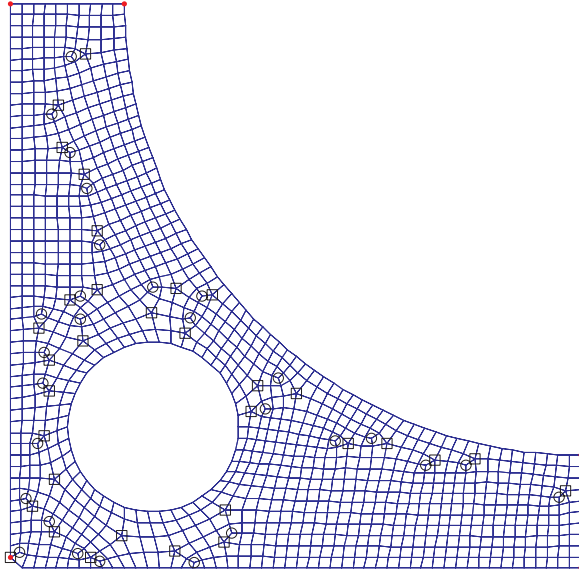


Fig. 14. Mesh of example 3 before clean-up.

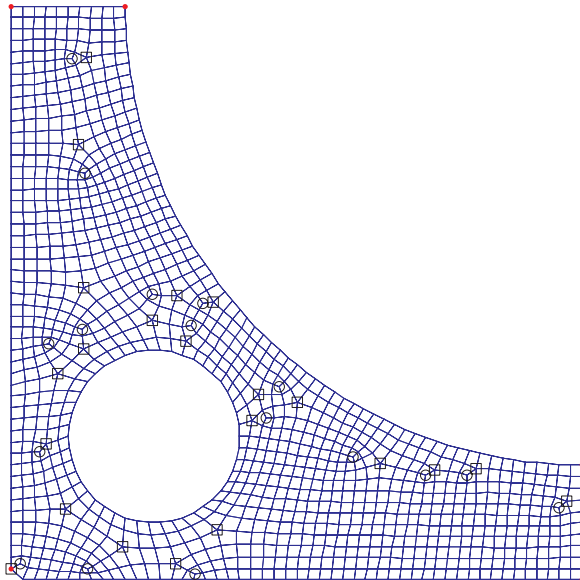


Fig. 15. Mesh of example 3 after clean-up.

Table 3: Mesh characteristics of example 3.

	Before Clean Up	After Clean Up
Number of Cells	1047	1041
Number of Defects	61	39
Average Metric β	0.900	0.906
Minimal Metric β	0.466	0.471
Number of Cells with $\beta < 0.7$	58	45
Size Transition Metric	1.107	1.108

4. Conclusions

A new approach to topological clean-up of 2D meshes was proposed. The basic clean-up operation in this method is creating a new connectivity for a region of the mesh, and replacing the existing mesh of the region. The new connectivity is created using information on the structure of the boundary of that region only. If this new connectivity is thought to improve the mesh quality, a replacement of this mesh region is made.

An algorithm utilizing this approach for quadrilateral meshes was presented in detail, and the results of applying it on example meshes were shown. These examples, as well as others, suggest that a large reduction in the number of cells with lower shape quality ($\beta < 0.7$) can be achieved. This fact is closely tied to the reduction in the number of defects (irregular nodes) in the mesh. The effect on the average shape quality is usually small, a few percent at most. Another observation is that the clean-up can help improve the cell size transitions in meshes where there are significant changes in cell size.

The proposed approach seems to be especially beneficial for clean-up far from the boundary. There, replacement operations over larger distances can be carried out, and the relative advantage over local clean-up techniques is more pronounced. This makes the technique a good complement to advancing front mesh generators, which tend to create defects inside the mesh, where the fronts collide in the meshing process.

Further research can include finding improved region selection algorithms, and perhaps improved topological meshing methods. Good algorithms for finding legal locations for the new nodes after replacement are important as well; the speed and robustness of the mesh replacement stage relies on them. The general approach can probably be applied to triangular and mixed quad-tri meshes as well.

Acknowledgements

The author wishes to thank Shlomi Hillel, Dov Levine, Zev Lovinger and Irad Yavne for many helpful discussions.

References

1. Blacker T.D., Stephenson M.B., *Paving: A New Approach to Automated Quadrilateral Mesh Generation*, Int. Journal for Numerical Methods in Eng., Vol. **32**, pp. 811–847, 1991.
2. Canann S.A., Muthukrishnan S. N., Phillips R. K., *Topological Improvement Procedures for Quadrilateral Finite Element Meshes*, Engineering with Computers, Vol. 14, no. 2, pp. 168–177, 1998.
3. Kinney P., *Clean Up: Improving Quadrilateral Finite Element Meshes*, Proc. of the 6th International Meshing Roundtable, pp. 449–461, 1997.
4. Lee C.K., Lo, S.H., *A new Scheme for the Generation of Graded Quadrilateral Meshes*, Computers and Structures, Vol. **52**, pp. 847-857, 1994.
5. Owen S.J., Staten M.L., Canann S.A., Saigal S., *Advancing Front Quadrilateral Meshing Using Triangle Transformations*, Proc. of the 7th International Meshing Roundtable, 1998.
6. Staten M.L., Canann S.A., *Post Refinement Element Shape Improvement for Quadrilateral Meshes*, **AMD-220**, Trends in Unstructured Mesh Generation, ASME, pp. 9–16, 1997.
7. Vachal P., Garimella R. V., Shashkov M. J., *Untangling of 2D meshes in ALE simulations*, J. of Computational Physics, vol. **196**, no. 2, pp. 627–644, 2004.

Quad-Dominant Mesh Adaptation Using Specialized Simplicial Optimization

Ko-Foa Tchou and Ricardo Camarero

Department of Mechanical Engineering, École Polytechnique de Montréal,
C.P. 6079, Succ. Centre-ville, Montreal (QC) H3C 3A7, Canada
[ko-foa.tchon|ricardo.camarero]@polymtl.ca

Summary. The proposed quad-dominant mesh adaptation algorithm is based on simplicial optimization. It is driven by an anisotropic Riemannian metric and uses specialized local operators formulated in terms of an L_∞ instead of the usual L_2 distance. Furthermore, the physically-based vertex relocation operator includes an alignment force to explicitly minimize the angular deviation of selected edges from the local eigenvectors of the target metric. Sets of contiguous edges can then be effectively interpreted as active tensor lines. Those lines are not only packed but also simultaneously networked together to form a layered rectangular simplicial mesh that requires little postprocessing to form a cubical-dominant one. Almost all-cubical meshes are possible if the target metric is compatible with such a decomposition and, although presently only two-dimensional tests were performed, a three-dimensional extension is feasible.

Key words: Quad-dominant, mesh adaptation, anisotropic Riemannian metric.

1 Introduction

Elementary modification operators are essential to optimize the computational meshes used by finite element and finite volume solvers. They have been successfully employed to automatically adapt simplicial meshes of triangles, in two dimensions, and tetrahedra, in three dimensions [1]. However, this level of automation has not yet been duplicated for cubical meshes of quadrilaterals, in two dimensions, and hexahedra, in three dimensions. Generation, let alone adaptation, of such meshes is still a challenge. There is, nevertheless, a strong demand for quality cubical meshes either due to the intrinsic properties of such elements or simply for compatibility with existing solvers.

Although fairly robust two-dimensional methods have been developed, current conformal all-hexahedral algorithms are usually limited in scope and cannot automatically process arbitrary shaped domains [2]. Acknowledging this difficulty, cubical-dominant algorithms allow a small percentage of non-cubical elements in order to achieve an increased level of automation. The present work proposes such an algorithm that combines cubical particle packing [3, 4]

with tensor line networking [5] and recasts the whole process as a specialized simplicial optimization. More precisely, physically-based attraction-repulsion forces are used to distribute the vertices of a simplicial mesh according to the local density prescribed by an anisotropic Riemannian control metric. Coupled with appropriate particle or vertex population control, this results in an approximate centroidal Voronoi-Delaunay triangulation. Furthermore, the proximity-based particle interaction force is modified to promote cubical Voronoi regions by using an L_∞ norm instead of the usual L_2 norm to compute metric distance. Particle population control is also reformulated in terms of simplicial refinement and coarsening operations using the same chessboard distance. These modifications provide, however, only local alignment and an additional constraint is needed to recover the globally layered structure of an ideal cubical mesh. An angle-based torsional spring-like force is used for this purpose and aligns selected mesh edges with the local metric eigenvectors. A physical interpretation of this optimization process can best be described as line networking. Set of contiguous mesh edges indeed effectively form tensor lines that are not only packed but also simultaneously interconnected together to form a layered rectangular simplicial mesh that requires little processing to form a cubical-dominant one. Almost all-cubical meshes are also possible if the target metric is compatible with such a decomposition. Finally, although presently only two-dimensional cases were considered, a three-dimensional extension is feasible and should be computationally competitive with classical simplicial optimization.

Following a brief summary of existing adaptation methods, the present paper describes the proposed specialized simplicial optimization algorithm and uses both academic and practical test cases to illustrate its capabilities.

2 Simplicial Versus Cubical Mesh Adaptation

2.1 Riemannian Metrics and Distance

Mesh adaptation algorithms are typically controlled by a target map specifying the desired element size according to its location in the domain. For anisotropic maps, size also varies according to element orientation. Metric-based algorithms cast such a target map as a tensor representing a deformation of space that modifies how the length, area and volume of mesh entities are measured [6, 7]. Such a Riemannian metric tensor is a symmetric positive definite matrix \mathbf{M} and can be factored as the product of a rotation matrix \mathbf{R} and a diagonal matrix $\mathbf{\Lambda}$ as in this two-dimensional formula

$$\mathbf{M} = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1} = (\mathbf{e}_1 \ \mathbf{e}_2) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} \mathbf{e}_1^t \\ \mathbf{e}_2^t \end{pmatrix}. \quad (1)$$

The columns of \mathbf{R} are the eigenvectors of \mathbf{M} and correspond to two prescribed directions \mathbf{e}_1 and \mathbf{e}_2 . The diagonal terms λ_1 and λ_2 are the strictly positive eigenvalues of \mathbf{M} . The target sizes h_1 and h_2 along \mathbf{e}_1 and \mathbf{e}_2 are given by the inverse square root of those eigenvalues, i.e., $h_i = 1/\sqrt{\lambda_i}$.

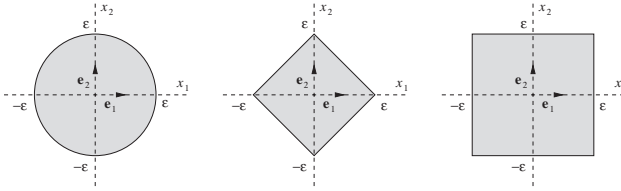


Fig. 1. Two-dimensional ϵ -balls for a metric distance based on an L_2 (left), an L_1 (middle) and an L_∞ norm (right).

Using such a target map, adapting a simplicial mesh is equivalent to requiring that all its edges have a unit metric length, as explained in Sect. 2.2. The notion of metric distance between two vertices is thus essential. For a locally constant metric \mathbf{M} , it is defined as follows

$$l_{ab} = \sqrt{(\mathbf{p}_b - \mathbf{p}_a)^t \mathbf{M} (\mathbf{p}_b - \mathbf{p}_a)} \quad (2)$$

where a and b are any two vertices and \mathbf{p} is a position vector. For non-constant metrics, l_{ab} can be approximated by integrating this formula along segment ab or by using an averaged metric. Following Minkowski's formula in Euclidean space, a generalized distance can also be defined in metric space as an L_p norm

$$l_{ab} = \left(\sum_{i=1}^d |x_i|^p \right)^{1/p} \quad (3)$$

where $x_i = \sqrt{\lambda_i} \mathbf{e}_i \cdot (\mathbf{p}_b - \mathbf{p}_a)$ and d is the considered dimension. If $p = 2$ then distance is measured using the classical L_2 norm given in Eq. 2. On the other hand, $p = 1$ corresponds to an L_1 norm also called taxicab or Manhattan distance while $p = \infty$ corresponds to the infinity norm also called Chebyshev or chessboard distance

$$l_{ab} = \max_{1 \leq i \leq d} |x_i|. \quad (4)$$

The differences induced by those norms in the partition of space are illustrated in Fig. 1. The chessboard distance, i.e., the L_∞ norm, is of particular interest for cubical adaptation as explained in Sect. 3.1.

2.2 Simplicial Adaptation

The ideal simplex is considered to be a *regular* one, i.e., an equilateral triangle in two dimensions (Fig. 2) or an equilateral tetrahedron in three dimensions. A quality regular simplicial mesh should be composed of such elements. However, equilaterality implies constant or almost-constant element sizes. To facilitate the generation of variable density simplicial meshes, a Riemannian metric tensor can be introduced. A quality mesh should then be composed of regular simplices in metric space, i.e., the edges of its elements should all have the same metric length or, more precisely, a unit metric length. A perfectly adapted

mesh is therefore called a unit mesh. This compact and elegant framework to measure shape quality and size conformity has been extensively used to generate adapted simplicial meshes. See for example [1] and the references cited therein. Furthermore, adapting a mesh to a solution is an iterative process that can be done by global mesh regeneration or local mesh modifications. In an iterative adaptation context where modifications to a previous mesh are expected to be minimal, the latter approach may be advantageous. The elementary simplicial mesh modification primitives or operators optimize the vertex density and element shape quality to get as close as possible to a unit mesh in metric space. They can be classified as follows:

- refinement and coarsening improve local vertex density by inserting and deleting vertices;
- reconnection improves element shape by flipping faces and edges;
- relocation improves both element shape and local vertex density by repositioning individual vertices at optimal locations.

Those primitives can also be used after global mesh generation as postprocessing operations. Global remeshing can then be viewed as a way to produce good initial meshes for a local optimization process.

2.3 Cubical Adaptation

As for simplices, the ideal cubical element is a *regular* one, i.e., a square in two dimensions (Fig. 2) or a cube in three dimensions. This, however, implies edges not only equal in length but also joined at right angles. In practice, this orthogonality is very important and, in an anisotropic metric-based adaptation framework, can be preserved in physical space only if the edges

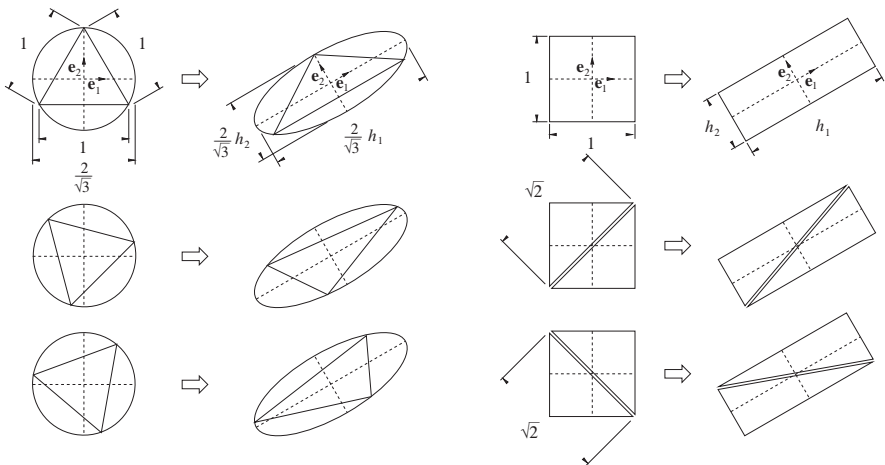


Fig. 2. Ideal triangle (left) versus ideal quadrilateral and its triangular decompositions (right) in metric space and in physical space. The metric tensor is considered locally constant.

of the cubical element are aligned with the local eigenvectors. This explicit directional constraint contrasts with the weak and indirect alignment of an adapted simplex. A perfect simplicial element can indeed have any orientation as long as its edges have a unit metric length and its loose alignment with the minor eigenvector of the metric, corresponding to the smallest eigenvalue and thus the biggest target size, is only due to the stretching introduced by the spacial transformation (Fig. 2). A perfectly adapted cubical mesh is thus understandably more difficult to obtain than a simplicial one. In addition to this directional constraint, cubical connectivity is also more complex to manipulate. It indeed has a layered structure that practically precludes strictly local modifications. Vertex relocation or smoothing is the most widely used method to adapt cubical meshes, particularly structured ones [8]. However, although it can improve both vertex density and element shape, smoothing is limited by a fixed mesh topology. Mesh refinement and coarsening, on the other hand, involve connectivity modifications [9, 10, 11, 12]. The major issue with conformal refinement-coarsening is, however, that element shape quality cannot be maintained without a powerful local reconnection method and the coveted cubical flip operator is still only theoretical in three dimensions [13].

Simplicial-to-cubical conversion methods avoid this problem by optimizing simplicial meshes and generating cubical elements only in a postprocessing step. Local mesh modifications thus only involve simplex manipulation and are greatly facilitated. Once the target vertex density is achieved, adjacent simplicial elements are merged to form cubical ones. Different merging strategies are possible including advancing fronts [14, 15, 16], graph theory [17] and quality constraints [18, 3, 4]. Although, some non-cubical elements may remain, particularly in three dimensions, such cubical-dominant meshes are acceptable in many applications. Furthermore, while Riemannian metrics have been used with direct cubical adaptation, the lack of a proper local reconnection operator limits their appeal. This indirect approach, on the other hand, can use the full complement of local optimization operators available for simplicial meshes. However, the ideal vertex distribution for a unit simplicial mesh is very different from the one required by perfectly adapted cubical elements. Not all edges should indeed have a metric length of one. For example, in two dimensions, edges that correspond to diagonals in the quadrilateral mesh should have a metric length of $\sqrt{2}$. Additionally, right triangles with sides aligned with the local metric eigenvectors are better suited than equilateral ones for simplicial-to-cubical conversion (Fig. 2).

Better vertex distributions can be obtained using physically-based packing algorithms that approximate centroidal Voronoi-Delaunay triangulations. An appropriate modification of the proximity-based force used to distribute the mesh vertices indeed enables the partition of the domain into cubical Voronoi regions [3, 4]. This improves the local alignment with the metric eigenvectors of the resulting simplices. The merged cubical elements thus have better shape and require much less postprocessing. The globally layered structure of the ideal cubical mesh is, however, difficult to recover using a strictly local method. Networks of tensor lines everywhere tangent to the local metric eigenvectors can be used for this purpose [5]. Due to the orthogonal nature

of this network, the resulting quadrilateral elements are very well shaped and triangular elements are only inserted when the tensor lines are fragmented to ensure the conformity of the final mesh. The practical implementation of such a method requires, however, an assortment of techniques that are not as computationally efficient as simplicial adaptation and may be difficult to extend to three dimensions.

The next section proposes an alternative approach that combines cubical particle packing with tensor line networking and recasts the whole process as a specialized simplicial optimization.

3 Specialized Simplicial Optimization

3.1 Specialized Simplicial Operators

For high quality simplicial-to-cubical conversion, edges corresponding to diagonals should have an L_2 metric length of $\sqrt{2}$, in two dimensions, or $\sqrt{3}$, in three dimensions, and not 1. The unit meshes generated by classical simplicial operators based on such an L_2 distance are thus not best suited. This discrepancy can be avoided by using an L_∞ norm to compute metric distance because all edges should then have a unit length including diagonals. This chessboard distance has thus been used to modify the specialized two-dimensional operators presented here.

Vertex relocation – A physically-based approach was chosen in the present work [19, 20, 3, 4]. In this paradigm, mesh vertices are particles and the following potential is used to derive the interaction forces between those particles

$$\phi(x) = \frac{1}{4} e^{-x^4} - \frac{3}{16} \Gamma\left(\frac{1}{4}, x^4\right) + C \quad (5)$$

where $\Gamma(a, z) = \int_z^\infty t^{a-1} e^{-t} dt$ is the incomplete gamma function and C is an arbitrary constant. The first derivative of this potential corresponds to the function given by Bossen and Heckbert [20]

$$\phi'(x) = \frac{d\phi}{dx} = (1 - x^4) e^{-x^4}. \quad (6)$$

If an L_2 norm is used to measure distance then the following attraction-repulsion force is exerted on vertex or particle i by particle j

$$\mathbf{F}_{ij} = -\frac{\phi'(l_{ij})}{l_{ij}} (\mathbf{p}_j - \mathbf{p}_i) = -\phi'(l_{ij}) h_{ij} \mathbf{u}_{ij} \quad (7)$$

where l_{ij} is the metric distance between i and j as defined in Eq. 2, h_{ij} is the target size along edge ij and \mathbf{u}_{ij} is the unit vector $(\mathbf{p}_j - \mathbf{p}_i)/\|\mathbf{p}_j - \mathbf{p}_i\|$. When coupled with appropriate population control, such a force will distribute the particles according to the density prescribed by the target metric. At equilibrium, the empty region maintained by this force around each particle

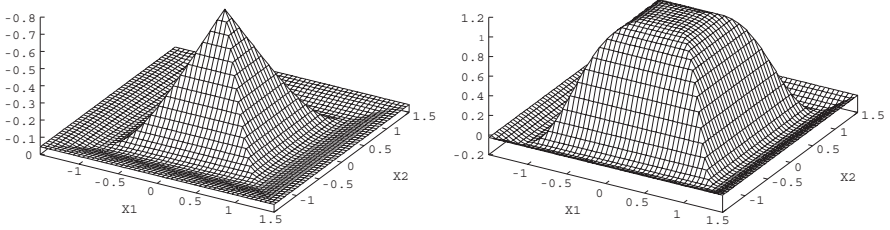


Fig. 3. Two-dimensional attraction-repulsion potential field (left) and the modulus of the corresponding force (right) for an L_∞ distance and an Euclidean metric, i.e., the identity matrix. The constant C in Eq. 5 is chosen so that the potential is equal to zero at a unit distance.

is analogous to an approximate Voronoi cell. Furthermore, since the particle is at the center of this cell, the resulting partition of space is an approximate centroidal Voronoi tessellation. The Voronoi cells have the shape of the ϵ -ball associated with the type of distance used to define the attraction-repulsion potential (Fig. 1). When an L_2 distance is used, those Voronoi cells have an elliptic or ellipsoidal shape. However, if an L_∞ or chessboard distance is used instead then those cells will have the desired cubical shape. Let m be the index of the eigenvector for which $|\sqrt{\lambda_m} \mathbf{e}_m \cdot (\mathbf{p}_j - \mathbf{p}_i)|$ is maximum. The chessboard attraction-repulsion force is then given by

$$\mathbf{F}_{ij} = -\alpha \phi'(l_{ij}) h_m \mathbf{e}_m \quad (8)$$

where l_{ij} is computed with Eq. 4, α is the sign of $\mathbf{e}_m \cdot (\mathbf{p}_j - \mathbf{p}_i)$ and $h_m = 1/\sqrt{\lambda_m}$ is the target size associated with the local metric eigenvector \mathbf{e}_m (Fig. 3). Using a first order equation of motion [20], the position of a particle or vertex i is updated at each iteration n as follows

$$\mathbf{p}_i^{n+1} = \mathbf{p}_i^n + \omega \sum_{j \in \mathcal{N}_i^v} \mathbf{F}_{ij}^n \quad (9)$$

where ω is a constant set to 0.2 and \mathcal{N}_i^v is the set of vertices sharing an edge with i . Of course boundary vertices have to be reprojected after each update.

Refinement and coarsening – Mesh refinement and coarsening correspond to particle population control in the physical paradigm. When the local density is too low, a particle is created and, when it is too high, a particle is destroyed. The normalized density around a particle i can be estimated by the inverse of the averaged metric area of its neighbors

$$\rho_i = \frac{\beta |\mathcal{N}_i^e|}{\sum_{j \in \mathcal{N}_i^e} \sqrt{\det(\mathbf{M}_j)} A(T_j)} \quad (10)$$

where $A(T_j)$ is area of triangle T_j , \mathbf{M}_j is the averaged metric within T_j , \mathcal{N}_i^e is the set of triangles sharing i and $|\mathcal{N}_i^e|$ is the number of triangles in this set.

The constant β is set to $\sqrt{3}/4$ for ellipse packing and 0.5 for square packing. The associated error is computed as follows

$$\delta(\rho) = \begin{cases} \rho - 1 & \text{if } \rho \geq 1, \\ \frac{1}{\rho} - 1 & \text{otherwise.} \end{cases} \quad (11)$$

The edge splitting and collapsing primitives from simplicial optimization can be used to insert and delete vertices and control this density. More precisely, if an edge has a metric length greater than a given threshold l_{\max} then it is split in two by introducing a new vertex in the middle. Similarly, if an edge has a metric length lower than a given threshold l_{\min} then it is collapsed by merging its two end vertices. Depending on the optimization strategy used, those operators may, however, tend to oscillate for rapidly varying metrics: they delete vertices that they just inserted. To stabilize the process, an estimation of the normalized density after each operation can be used as a safeguard [20]. Let ρ_i be the local density around a vertex i as defined in Eq. 10. When one of the interior edges connected to i is split then the local density becomes

$$\rho_i^+ = \rho_i (|\mathcal{N}_i^e| + 2) / |\mathcal{N}_i^e| \quad (12)$$

because two neighboring triangles must also be split in two. Similarly, when one interior edge connected to i is collapsed then the local density becomes

$$\rho_i^- = \rho_i (|\mathcal{N}_i^e| - 2) / |\mathcal{N}_i^e| \quad (13)$$

because two neighboring triangles must also be collapsed. Only the length based criterion is used for boundary edges but interior edges are split or collapsed only if $\delta(\rho_i^+) \leq \delta(\rho_i)$ or $\delta(\rho_i^-) \leq \delta(\rho_i)$ respectively. Again, to make the packing cubical, an L_∞ norm is used to compute the metric length of the edges. Furthermore, care must be taken when dealing with edges on curved boundaries to avoid degenerate configurations. See for example [21] for more details on how it can be done.

Local reconnection – In simplicial-to-cubical conversion methods, what is important is the rectangular distribution of the vertices more than their connectivity, as long as it is reasonable, i.e., coherent with the local target mesh density. This simplicial connectivity is essentially used for fast neighbor searching during relocation and population control operations. Once the vertex distribution is rectangular their simplicial connections will naturally be made of right angle simplices. That is why the local reconnection used in the present work is the classical operator from simplicial optimization. Furthermore, as only two-dimensional cases have been considered for now, only an edge swap or flip has been implemented. This operator replaces the edge shared by two triangles with a new edge linking their opposite vertices. This flip is only performed if the worst shape of the resulting triangles is better than the shape of the initial ones. The following measure of shape quality was used

$$\mathcal{Q}(T) = 4\sqrt{3} A(T) \min_{1 \leq i \leq 3} \frac{\sqrt{\det(\mathbf{M}_i)}}{\sum_{1 \leq j < k \leq 3} (\mathbf{p}_k - \mathbf{p}_j)^T \mathbf{M}_i (\mathbf{p}_k - \mathbf{p}_j)} \quad (14)$$

where i , j and k refer to the vertices of triangle T .

3.2 Tensor Line Alignment

Although an improvement over classical simplicial operators, the modifications presented in the previous section provide only local alignment with the eigenvectors of the target metric. The resulting Voronoi regions will be cubical but there is no guaranty that those regions will be aligned to form continuous layers that are so important to maximize the proportion of cubical elements in the final mesh. The staggered configuration in Fig. 4 is as valid as the aligned one. Only adequate boundary conditions can favor the latter one. A stronger and more explicit global alignment force is needed. The approach proposed here is reminiscent of the method introduced in [5]. The difference is that here the global metric topology embodied by the tensor line network is no longer traced beforehand but is recovered through essentially local modifications of a simplicial mesh.

For this purpose a torsion spring-like force explicitly minimizes the angular deviation of selected mesh edges with the local metric eigenvectors. Associated with lineal springs, such angle-based forces are already used in mesh smoothing [22]. Their use here is, however, more analogous to active polylines in computer graphics. More precisely, those active polylines are tensor lines formed by links that correspond to selected mesh edges. These edges will become the sides of the final cubical elements. They are identified by computing their angular deviation from the local eigendirections. Taking into account the deformation introduced by a metric \mathbf{M} , this angular deviation for an edge ij is given by

$$\theta_{ij} = \arccos \frac{\sqrt{\lambda_m} \mathbf{e}_m \cdot (\mathbf{p}_j - \mathbf{p}_i)}{\sqrt{(\mathbf{p}_j - \mathbf{p}_i)^t \mathbf{M} (\mathbf{p}_j - \mathbf{p}_i)}} \quad (15)$$

where m has the same definition as in Eq. 8. If $|\theta_{ij}| \leq \theta_{\max}/2$ then ij is a candidate tensor line link for the eigenvector field m (Fig. 5). If more than one edge is admissible for a given combination of eigenvector and orientation along this eigenvector then the one with the smallest deviation is chosen. The corresponding alignment force acting on vertex i is computed as follows

$$\mathbf{G}_{ij^*} = C_\theta (\mathbf{p}_j - \mathbf{p}_{j^*}) \quad (16)$$

where C_θ is a constant and $\mathbf{p}_{j^*} = \mathbf{p}_i + [(\mathbf{p}_j - \mathbf{p}_i) \cdot \mathbf{e}_m] \mathbf{e}_m$ is the projection of \mathbf{p}_j along \mathbf{e}_m (Fig. 5). This alignment constraint can be introduced as a modification to the relocation force and the vertex position update is then written as follows

$$\mathbf{p}_i^{n+1} = \mathbf{p}_i^n + \omega \left(\sum_{j \in \mathcal{N}_i^v} \mathbf{F}_{ij}^n + \sum_{j^* \in \mathcal{N}_i^{v^*}} \mathbf{G}_{ij^*}^n \right) \quad (17)$$

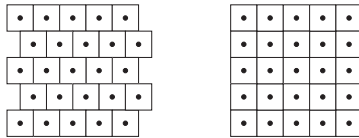


Fig. 4. Staggered (left) and aligned (right) square packing.

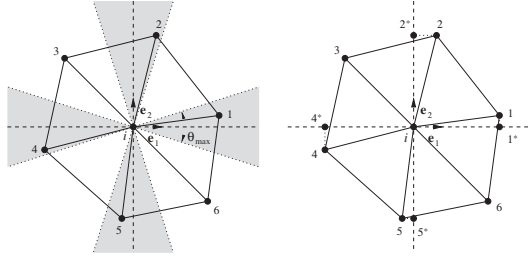


Fig. 5. Edge alignment.

where $\mathcal{N}_i^{v^*}$ is the set of links associated with i . Note that the constant C_θ must be chosen carefully. The greater C_θ , the better the alignment but also the stiffer the optimization problem will be. After some numerical experiments, C_θ was set to 2 and θ_{\max} to $\pi/4$.

Reevaluated each time a vertex position is updated, those links play the role of a specialized reconnection operator. More than simplicial connectivity, those links indeed reflect the topology of the final cubical mesh. Because of the continuity of the metric, the end-to-end collection of links also forms tensor lines. Those lines are continuous as long as the metric is continuous and the prescribed element size is reasonable compared to the metric variation or tensor line curvature. Those active lines can also be considered as directional agglomerations of particles and their thickness is equal to the local target size prescribed by the metric. When links are set or broken then those lines are effectively fused or split. When particles are created, moved or destroyed so are those lines. The packing of particles also means that lines are packed and form layers upon layers. Furthermore, since particles are linked along both eigenvectors, those lines are also interconnected. The process can thus be interpreted as physically-based line networking or weaving.

3.3 Optimization Algorithm

Algorithm 1 sums up the main steps performed during the optimization process. A coarse initial triangulation is required as an input. The boundary of the domain must be represented by this triangulation and, when refining boundary edges, those constraints must be preserved. An empty constrained Delaunay triangulation is a good example of initial mesh. The algorithm is vertex-wise. It loops through all the vertices, moves them and enforces population control until convergence or a given number of iterations is completed. The threshold l_{\min} is set to 0.75 while l_{\max} is set to 1.33. The movement of a vertex due the attraction-repulsion as well as alignment forces is limited to the visibility zone of its immediate neighbor vertices, i.e., no inverted triangle should be created by the computed displacement. Any displacement that does not meet this requirement is discarded. Furthermore, after each successful relocation or refinement-coarsening operation, the local reconnection operator is called recursively to maintain the quality of the neighboring simplices. Note

Algorithm 1 Specialized simplicial optimization

Require: constrained empty triangulation

```

repeat
  for all vertex  $i$  do
    update position
    if moved then
      reconnect neighborhood
    end if
    find  $is$  and  $il$ , the shortest and longest edges connected to  $i$ 
    if  $l_{is} < l_{\min}$  and ( $is$  on boundary or  $\delta(\rho_i^-) \leq \delta(\rho_i)$ ) then
      collapse  $is$ 
    else if  $l_{il} > l_{\max}$  and ( $il$  on boundary or  $\delta(\rho_i^+) \leq \delta(\rho_i)$ ) then
      split  $il$ 
    end if
    if an edge has been collapsed or split then
      reconnect neighborhood
    end if
  end for
until convergence

```

also that the metric is stored at each vertex of the triangulation. For new vertices or when vertices are moved, the metric is computed using an analytical function or by interpolation in a background mesh. In all edge-based computations, the metric is considered constant and the average of the values stored at the two end vertices is used.

Although similar in purpose, the present cubical packing force field was established by a very different reasoning and does not have the same shape than the one proposed in [3, 4]. An alignment force is also used to further promote the formation of continuous layers and ultimately increase the percentage of cubical elements in the final mesh. Additionally, like the pliant method proposed by Bossen and Heckbert [20], this process is recast as a simplicial mesh optimization and should eventually have the same computational efficiency. The problem has, however, more constraints and should thus take more time to converge. One potential difficulty is, as with other optimization methods, to get stuck in a local minimum. A possible approach to avoid such local minima is to start with a wider permissible edge length range and to slowly tighten it as the mesh is adapted. This results in a gradual and globally consistent evolution of the mesh that is more likely to reach a global minimum. An additional strategy is to first adapt the mesh using only attraction-repulsion forces combined with refinement-coarsening operations. The result is then used as an initial mesh for the stiffer problem with alignment forces. Both of these strategies are used for the test examples presented in the next section.

Once optimized, very little additional processing is needed to produce the final high quality quad-dominant mesh. Available edges for simplicial-to-cubical conversion are first identified. Forbidden edges are tensor line links and constrained boundary edges. For each of the available edges, the quality of the candidate quadrilateral element Q formed by the adjacent triangles is

computed as follows

$$\mathcal{Q}(Q) = \frac{2}{\sqrt{3}} \min_{1 \leq i \leq 4} \mathcal{Q}(T_i) \quad (18)$$

where T_i is the corner triangle associated with vertex i of Q . Edges with $\mathcal{Q} < 0.3$ are discarded. The remaining edges are listed in decreasing order according to this quality, the corresponding quadrilaterals are constructed and their edges removed from the list until it is empty.

4 Numerical Results

4.1 Academic Test Examples

The first example is defined on a $[0, 7] \times [0, 9]$ rectangular domain. The prescribed target sizes for this example are

$$h_1 = \begin{cases} 1 - \frac{19}{40}y & \text{if } y \in [0, 2], \\ 20^{(2y-9)/5} & \text{if } y \in]2, \frac{9}{2}], \\ 5^{(9-2y)/5} & \text{if } y \in]\frac{9}{2}, 7], \\ \frac{1}{5} + \frac{1}{20}(y-7)^4 & \text{if } y \in]7, 9] \end{cases} \quad (19)$$

and $h_2 = 1.01 h_1$ with \mathbf{e}_1 and \mathbf{e}_2 corresponding to the Cartesian axes. This is a quasi-isotropic version of the analytic metric from [1]. Purely isotropic metrics are indeed considered degenerate for tensorline-based cubical meshing because \mathbf{e}_1 and \mathbf{e}_2 would then be indefinite [5]. The resulting adapted triangular and quad-dominant meshes are presented in Fig. 6. The size variation prescribed by this metric does not allow an all-quadrilateral mesh but the algorithm is still able to directionally partition the domain and maximize the number of such elements, i.e., there is 84.8 percent of quadrilaterals. Table 1 gives some

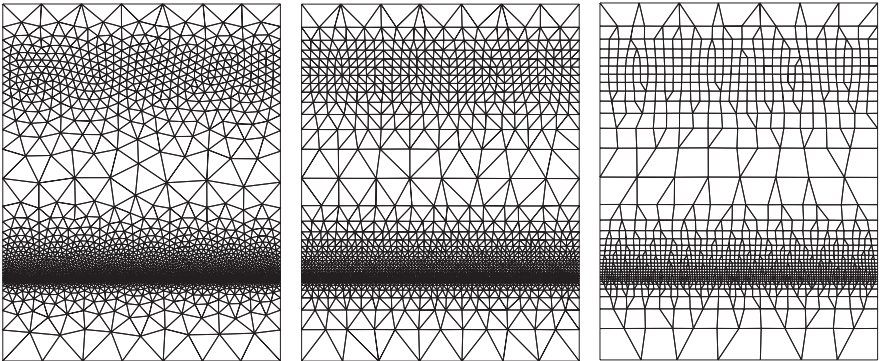


Fig. 6. Quasi-isotropic metric (Eq. 19) – Unit simplicial mesh obtained using classical L_2 -based (left) and specialized L_∞ -based operators (middle) as well as final quad-dominant mesh (right).

Table 1. Some statistics for the meshes presented in Figs. 6 to 10.

	Fig. 6	Fig. 7	Fig. 8	Fig. 9	Fig. 10
Nb edges	5268	2853	8895	11815	39917
Min. length	0.652	0.983	0.487	0.485	0.499
Ave. length	1.006	0.999	0.986	0.997	0.987
Max. length	1.364	1.026	1.890	1.686	1.778
Std dev.	0.071	0.005	0.096	0.070	0.094
Nb triangles	292	0	606	446	2100
Min. quality	0.703	-	0.501	0.507	0.454
Ave. quality	0.854	-	0.885	0.845	0.864
Max. quality	0.982	-	1.000	0.989	1.000
Std dev.	0.044	-	0.065	0.084	0.070
Nb quads.	1624	972	2671	3874	12307
Min. quality	0.530	0.667	0.310	0.353	0.304
Ave. quality	0.894	0.943	0.873	0.876	0.855
Max. quality	0.996	1.000	0.996	0.990	0.995
Std dev.	0.119	0.057	0.122	0.068	0.101
Quad-dom.	84.8%	100%	81.5%	89.7%	85.4%

statistics for this mesh as well as the other meshes presented in this section. These statistics include the number of edges and their metric length as well as the number of triangular and quadrilateral elements and their shape quality as computed with Eqs. 14 and 18. Note that, for comparison, a mesh obtained with classical operators is also presented in Fig. 6.

The next example is anisotropic and is defined on the same rectangular domain. Again the prescribed directions correspond to the Cartesian axes while the target sizes are computed as follows

$$h_1 = \begin{cases} 1 - \frac{19}{40}x & \text{if } x \in [0, 2], \\ 20^{(2x-7)/3} & \text{if } x \in]2, \frac{7}{2}], \\ 5^{(7-2x)/3} & \text{if } x \in]\frac{7}{2}, 5], \\ \frac{1}{5} + \frac{1}{20}(x-5)^4 & \text{if } x \in]5, 7], \end{cases} \quad h_2 = \begin{cases} 1 - \frac{19}{40}y & \text{if } y \in [0, 2], \\ 20^{(2y-9)/5} & \text{if } y \in]2, \frac{9}{2}], \\ 5^{(9-2y)/5} & \text{if } y \in]\frac{9}{2}, 7], \\ \frac{1}{5} + \frac{1}{20}(y-7)^4 & \text{if } y \in]7, 9]. \end{cases} \quad (20)$$

The resulting mesh is presented in Fig. 7. Contrary to the first test case, a completely quadrilateral mesh is possible and the present method is able to generate it.

The final analytic test case is another quasi-isotropic metric, dubbed *banana*, that uses a size distribution taken from Lewis et al. [23] and is defined as follows

$$h_1 = \frac{1}{100} [1 + 30(y - x^2)^2 + (1 - x)^2] \quad (21)$$

and $h_2 = 1.01 h_1$ with $(x, y) \in [-1.25; 1.25] \times [-0.5; 1.25]$. The normalized gradient of the size distribution, i.e., $\nabla h_1 / \|\nabla h_1\|$, is taken as \mathbf{e}_1 while \mathbf{e}_2 is simply equal to \mathbf{e}_1 rotated by an angle of 90 degrees. The tensor line network

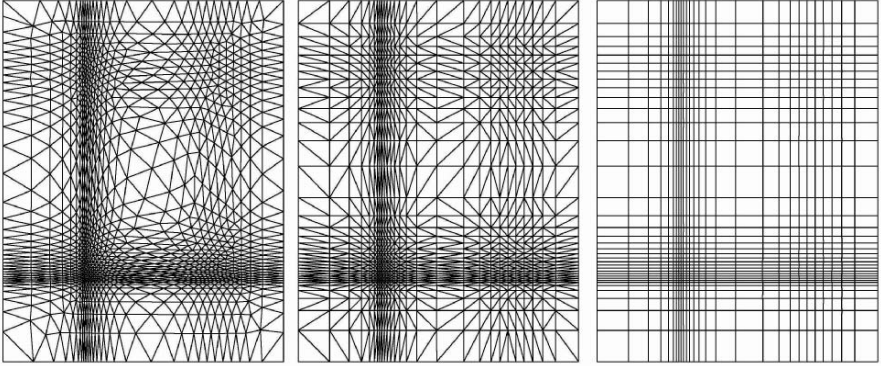


Fig. 7. Anisotropic metric (Eq. 20) – Unit simplicial mesh obtained using classical L_2 -based (left) and specialized L_∞ -based operators (middle) as well as final quadrilateral mesh (right).

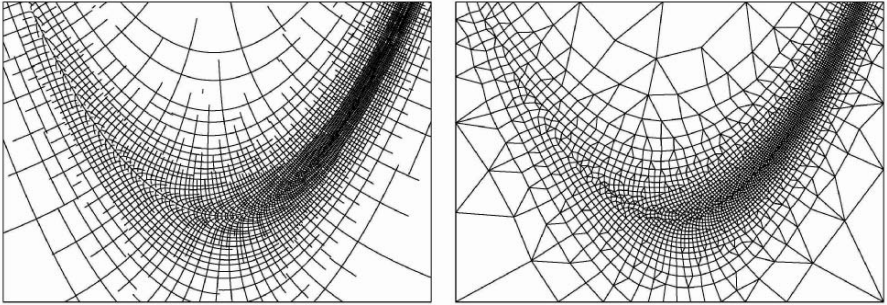


Fig. 8. Banana metric (Eq. 21) – Tensor line network (top) and corresponding adapted quad-dominant mesh (bottom).

for this metric is not aligned with the Cartesian axes as in the previous examples and is presented in Fig. 8 along with the final adapted quad-dominant mesh. The quality of the quadrilaterals in this mesh is substantially superior to the quality of the merged elements obtained by Borouchaki and Frey [18], i.e., an average of 0.87 versus 0.63, although the percentage of triangular elements is higher, i.e., 18.5 versus 3 percent (Table 1). Element shape quality is intentionally favored by the present method.

4.2 Practical Application Examples

The first practical application example is a mesh generated for a geometry-based metric. It shows how to use level set information encapsulated in a metric to generate high quality meshes. The considered domain is a heat exchanger and the metric is constructed using a variation of the ideas proposed in [24]. The vector \mathbf{e}_1 is set to the normalized gradient of ϕ , the distance-to-the-closest-boundary function, while \mathbf{e}_2 is again equal to \mathbf{e}_1 rotated by an angle of

90 degrees. The target size h_1 along \mathbf{e}_1 is limited by the local thickness of the domain. The function ϕ can be used to compute the medial axis of the domain and deduce this local thickness [24]. Furthermore, a user defined clustering has been added to the strictly geometric information extracted from ϕ . More precisely, the target size h_1 , which is normal to the boundaries of the domain, is computed as follows

$$h_1 = \min(h_1^w + (\gamma - 1)\phi, h_\tau) \quad (22)$$

where h_1^w is the user prescribed size of the elements at the closest boundary, γ is the associated geometric growth ratio and h_τ is a limiting size computed from the local thickness τ of the domain. For the considered example, h_τ was set to 0.25τ . Along \mathbf{e}_2 , the target size h_2 is computed as follows

$$h_2 = h_2^w(1 + |\kappa|\phi) \quad (23)$$

where κ is the curvature of the closest boundary and $h_2^w = 2\pi/N|\kappa|$. The user prescribed constant N corresponds to the number of elements needed to discretize a perfectly circular boundary. Furthermore, using Eq. 23 with adjacent circular and completely flat boundaries introduces very difficult size transitions and explicit gradation has to be used to ensure high quality meshes [25]. The maximum target size growth between adjacent elements, γ_0 , was set to 1.2. Figure 9 shows the tensor line network for $N = 32$, $\gamma = 1.2$ and $h_1^w = 0.1r$ with r being the radius of the internal circular boundaries. The resulting adapted mesh is also presented in Fig. 9 and counts almost 90 percent of quadrilaterals with an average quality of 0.876 (Table 1).

The final example uses a solution-based metric extracted from the computed flow around a NACA 0012 airfoil using an Hessian-based error estimator

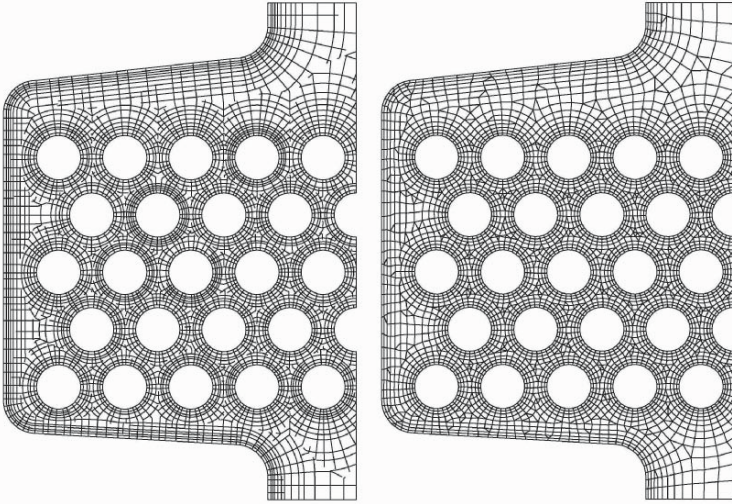


Fig. 9. Heat exchanger – Tensor line network (left) and corresponding quad-dominant meshes (right).

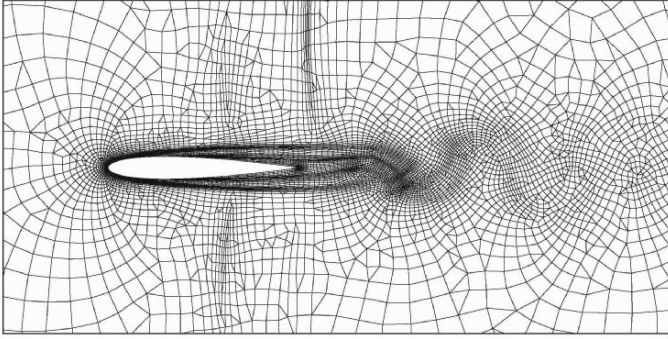


Fig. 10. Naca 0012 airfoil – Quad-dominant mesh adapted to an unsteady flow at a Mach number of 0.85 and a Reynolds number of 5000.

[26]. An unsteady laminar transonic flow with a Reynolds number of 5000 and a Mach number of 0.85 was considered. Again, gradation with $\gamma_0 = 1.2$ was used. The resulting mesh is presented in Fig. 10 and counts about 85 percent of quadrilaterals with an average quality of 0.855 (Table 1).

4.3 Future Developments

The results presented above show that the proposed algorithm is able to generate high quality quad-dominant meshes. However, to obtain high percentages of quadrilateral elements, the metric must prescribe smoothly varying target sizes. Just as for triangles, there is a physical limit on how fast target size can vary to get well shaped quadrilaterals. A specialized gradation algorithm must be developed as already suggested in [5]. All-cubical meshes are also possible if this gradation algorithm can provide compatible metrics. The problem of obtaining such meshes is thus displaced from the actual generation task to the metric construction task. This is hopefully a simpler problem.

The more compatible the metric, the easier it is to obtain a global minimum. However, even for a completely compatible metric such as the one defined by Eq. 20, local minima are possible and this implies the presence of unnecessary triangles in the final mesh. To avoid such a local minimum, the following two-step strategy was used. The mesh is first adapted without alignment forces. Three passes are used: 200 iterations with $l_{\min} = 0.75$ and $l_{\max} = 2.66$ then 200 iterations with l_{\max} reduced to 2.00 and finally 200 iterations with $l_{\max} = 1.33$. The second step uses the result as an initial mesh and performs an additional 200 iterations with alignment forces, $l_{\min} = 0.75$ and $l_{\max} = 1.33$. This fixed number of iterations is probably overkill but the goal of the present work is to prove the soundness of the proposed specialized operators and not yet their efficiency. Given those considerations, CPU timings are not very meaningful but, to give an order of magnitude, the quadrilateral mesh presented in Fig. 7 took 101.62 seconds on an Intel Pentium M running at 1.3GHz. There is, furthermore, little overhead per iteration per vertex compared to classical operators. The real increase in computer time is due to

the number of iterations needed to converge to a more difficult problem. An optimized iterative process will also be the goal of future developments.

Finally a three-dimensional extension is very possible and more practical than tensor surface networks. Simplicial-to-cubical conversion templates indeed already exist [4, 17] and classical simplicial reconnection operators can be used in a future three-dimensional optimizer. The generalization of all the other operators presented in Sect. 3 is mostly trivial.

5 Conclusion

The proposed specialized simplicial optimization is capable of automatically generating high quality quad-dominant meshes with a layered structure aligned along the local eigenvectors of an anisotropic Riemannian control metric. This confirms the soundness of the L_∞ -based simplicial operators and the angle-based alignment force. The computational efficiency is not yet established but is expected to be similar to classical simplicial optimization. Only two-dimensional cases were considered in the present paper but a three-dimensional extension is feasible. Besides cubical meshes, the present method can also provide simplicial mesh postprocessing when right angle simplices are required. All-cubical meshes are also possible if the metric is compatible. Automatic cubical mesh generation can thus be achieved using a two-prong approach: the construction of a compatible metric and the generation of the corresponding mesh. The present paper deals with the second part. Future developments should take care of the first part.

References

1. George, P.-L. and Borouchaki, H. (1998) Delaunay Triangulation and Meshing. Applications to Finite Elements. Hermès, Paris.
2. Blacker, T. (2001) Automated conformal hexahedral meshing constraints, challenges and opportunities. *Engineering with Computers*, 17:201–210.
3. Shimada, K., Liao, J.-H., and Itoh, T. (1998) Quadrilateral meshing with directionality control through the packing of square cells. Seventh International Meshing Roundtable, Dearborn, MI, Oct., pp. 61–76, Sandia National Laboratories.
4. Yamakawa, S. and Shimada, K. (2003) Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *Int. J. Numer. Meth. Engng*, 57:2099–2129.
5. Tchou, K.-F., Guibault, F., Dompierre, J., and Camarero, R. (2005) Adaptive hybrid meshing using metric tensor line networks. 17th AIAA Computational Fluid Dynamics Conference, Toronto, ON, Canada, Jun., no. AIAA-2005-5332.
6. Vallet, M.-G. (1992) Génération de maillages éléments finis anisotropes et adaptatifs. Ph.D. thesis, Université Pierre et Marie Curie, Paris VI, France.
7. Simpson, R. B. (1994) Anisotropic mesh transformations and optimal error control. *Applied Numerical Mathematics*, 14:183–198.
8. Thompson, J. F., Warsi, Z. U. A., and Mastin, C. W. (1985) Numerical grid generation: Foundations and applications. North-Holland.

9. Schneiders, R. (2000) Octree-based hexahedral mesh generation. *Int. J. of Comp. Geom. & Applications*, 10:383–398.
10. Maréchal, L. (2001) A new approach to octree-based hexahedral meshing. Tenth International Meshing Roundtable, Newport Beach, CA, Oct., pp. 209–221, Sandia National Laboratories.
11. Tchon, K.-F., Dompierre, J., and Camarero, R. (2004) Automated refinement of conformal quadrilateral and hexahedral meshes. *Int. J. Numer. Meth. Engng*, 59:1539–1562.
12. Benzley, S. E., Harris, N. J., Scott, M., Borden, M., and Owen, S. J. (2005) Conformal refinement and coarsening of unstructured hexahedral meshes. *Journal of Computing and Information Science in Engineering*, 5:330–337.
13. Bern, M., Eppstein, D., and Erickson, J. (2002) Flipping cubical meshes. *Engineering with Computers*, 18:173–187.
14. Owen, S. J., Staten, M. L., Canann, S. A., and Saigal, S. (1999) Q-morph: An indirect approach to advancing front quad meshing. *Int. J. Numer. Meth. Engng*, 44:1317–1340.
15. Owen, S. J. and Saigal, S. (2000) H-morph: An indirect approach to advancing front hex meshing. *Int. J. Numer. Meth. Engng*, 49:289–312.
16. Lee, Y. K. and Lee, C. K. (2003) A new indirect anisotropic quadrilateral mesh generation scheme with enhanced local mesh smoothing procedures. *Int. J. Numer. Meth. Engng*, 58:277–300.
17. Meshkat, S. and Talmor, D. (2000) Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *Int. J. Numer. Meth. Engng*, 49:17–30.
18. Borouchaki, H. and Frey, P. J. (1998) Adaptive triangular-quadrilateral mesh generation. *Int. J. Numer. Meth. Engng*, 41:915–934.
19. Shimada, K. and Gossard, D. (1995) Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. *ACM Third Symposium on Solid Modeling and Applications*, Salt Lake City, UT, May, pp. 409–419.
20. Bossen, F. J. and Heckbert, P. S. (1996) A pliant method for anisotropic mesh generation. Fifth International Meshing Roundtable, Pittsburgh, PA, Oct., pp. 63–76, Sandia National Laboratories.
21. Li, X., Shephard, M. S., and Beall, M. W. (2003) Accounting for curved domains in mesh adaptation. *Int. J. Numer. Meth. Engng*, 58:247–276.
22. Farhat, C., Degand, C., Koobus, B., and Lesoinne, M. (1998) Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163:231–245.
23. Lewis, R. W., Zheng, Y., and Usmani, A. S. (1995) Aspects of adaptive mesh generation based on domain decomposition and Delaunay triangulation. *Finite Elements Anal. Des.*, 20:47–70.
24. Tchon, K.-F., Khachan, M., Guibault, F., and Camarero, R. (2005) Three-dimensional anisotropic geometric metrics based on local domain curvature and thickness. *Comp.-Aided Design*, 37:173–187.
25. Li, X., Remacle, J.-F., Chevaugeon, N., and Shephard, M. S. (2004) Anisotropic mesh gradation control. Thirteenth International Meshing Roundtable, Williamsburg, VA, Sep., pp. 401–412, Sandia National Laboratories.
26. Habashi, W. G., Dompierre, J., Bourgault, Y., Ait-Ali-Yahia, D., Fortin, M., and Vallet, M.-G. (2000) Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. Part I: General principles. *Int. J. Numer. Meth. Fluids*, 32:725–744.

Mesh Modification Under Local Domain Changes*

Narcís Coll, Marité Guerrieri and J. Antoni Sellarès

Departament d'Informàtica i Matemàtica Aplicada, Universitat de Girona,
{coll,mariteg,sellares}@ima.udg.es

Summary. We propose algorithms to incrementally modify a mesh of a planar domain by interactively inserting and removing elements (points, segments, polygonal lines, etc.) into or from the planar domain, keeping the quality of the mesh during the process. Our algorithms, that combine mesh improvement techniques, achieve quality by deleting, moving or inserting Steiner points from or into the mesh. The changes applied to the mesh are local and the number of Steiner points added during the process remains low. Moreover, our approach can also be applied to the directly generation of refined Delaunay quality meshes.

1 Introduction

In many two-dimensional geometric modelling problems it is desirable to obtain a triangular mesh that respects the domain of interest ensuring that the triangles of the triangulation satisfy some quality requirements. There exist many works on the generation of a quality mesh for a Planar Straight Line Graph (PSLG) domain. Delaunay refinement mesh generation algorithms have taken place in this frame of investigations [12, 13, 11, 7]. In keeping quality of a mesh two objectives are pursued. First, get skinny triangles, triangles without the required quality, out of the mesh. Second, force segments of the PSLG into the mesh. Both goals are achieved by the addition of Steiner points, points that do not belong to the original mesh. In current Delaunay refinement algorithms, two kinds of Steiner points deal with the former goal, namely, circumcenters and off-centers. The later objective is carried out by the addition of midpoints on constrained segments to insert. Local optimization techniques, like refinement, derefinement, topological changes and mesh smoothing, are employed usually as a postprocess to improve mesh quality [1, 5].

In this work we address the problem of adjusting a mesh to local changes of its domain. The initial motivation for this problem came from our interest

*Partially supported by the Spanish Ministerio de Educación y Ciencia under grant TIN2004-08065-C02-02.

on the simulation of cuts in triangulated objects. We modify a quality mesh of a PSLG under the insertion/removal of elements (points, segments, polygonal lines, etc.) to/from the PSLG, while keeping the quality of the mesh along the process. There exist some work related to the dynamic insertion and deletion of points and segments on Delaunay triangulations [6, 8], although they do not fit into the schema of Delaunay refinement algorithms.

Obviously, when a PSLG is changed, we can use a Delaunay refinement algorithm to modify the underlying mesh: the updated PSLG can be considered as input PSLG and a new mesh can be generated from scratch. However, when a PSLG is dynamically modified, apart of the quality requirement of its underlying mesh, we expect that additional features are going to be provided, namely:

Incrementality: The mesh of the updated PSLG is obtained without the regeneration of the whole mesh.

Locality: The changes applied to the mesh do not imply a propagation of these modifications to the whole mesh.

Optimality: The Steiner points added as a result of the modification of the mesh should be as few as possible.

A possible incremental solution to the insertion problem, alternative to generate a new mesh from scratch, is to apply a Delaunay refinement algorithm to the PSLG obtained joining: the current PSLG, the Steiner points of the current mesh and the elements to be inserted into the PSLG. In this way we can take advantage of the work done during the generation of the current mesh. However, considering Steiner points of the current mesh as part of the new PSLG contributes to the degradation in the distribution of Steiner points in successive updates of the mesh, generating a high number of small triangles, as we will show with some examples below. Suppose we have an initial PSLG composed of a square boundary and two points a and b (Figure 1(a)). After applying Ruppert's Delaunay refinement algorithm we obtain the mesh shown in Figure 1(b). The insertion of a new point, c , onto the PSLG close to an existing Steiner vertex, s , creates two new skinny triangles that have to be refined (Figure 1(c)). Then, an incremental Delaunay refinement algorithm generates the mesh in Figure 1(d).

The solution we propose to avoid the excessive insertion of vertices and the generation of small triangles produced by a degradation in the distribution of points in successive updates of the mesh is the deletion or the movement of Steiner vertices, followed if necessary by the addition of Steiner vertices according to Ruppert's algorithm. In Figure 1 we can compare the previous result of applying a Delaunay incremental refinement algorithm, Figure 1(d), with the movement of the Steiner vertex s to a suitable zone, in Figure 1(e), and with the deletion of the Steiner vertex, shown in Figure 1(f). From these examples one can observe that moving the vertex results in a much better mesh than in the case of using the incremental algorithm, but the optimal solution is obtained by the deletion process. This suggests that removal of

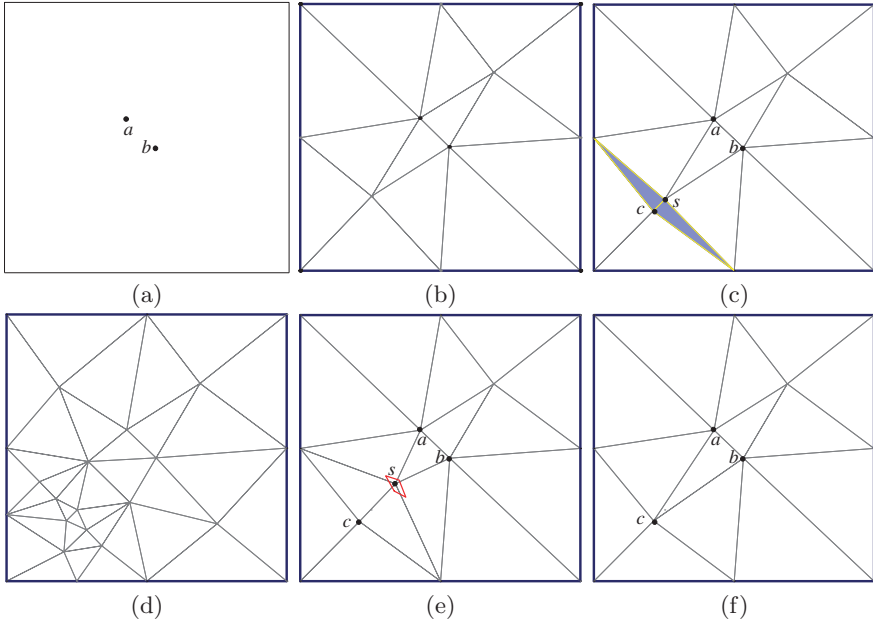


Fig. 1. (a) An initial PSLG composed of a square boundary and two points. (b) A Delaunay refined triangulation of the PSLG. (c) A new point to be added to the mesh with the skinny triangles associated to its insertion in grey. (d) The resulting mesh after applying an incremental Delaunay refinement algorithms to the mesh in Figure 1(c). (e) Result obtained if the Steiner vertex is moved to a suitable region. (f) Mesh achieved if the Steiner vertex is deleted.

Steiner vertices will therefore have priority in front of movement of Steiner vertices.

In this paper we introduce a framework, that combines mesh improvement techniques, for modifying incrementally a mesh under local changes of its PSLG domain. Starting from a quality mesh of the initial PSLG, we insert/remove elements to/from the PSLG in such a way that as the PSLG changes the underlying mesh is modified keeping its quality during the process. Our algorithms make only local modifications: deletion, movement or insertion of Steiner points from/into the mesh. Moreover, the number of Steiner points added during the process remains low.

One of the main ideas behind our proposed algorithms is that the bad vertex of some skinny triangles can be moved to a *quality zone* ensuring that a prefixed mesh quality is obtained. We give a numerical method for finding the optimal placement where the vertex has to be moved.

Our framework can also be applied to directly generate refined Delaunay quality meshes. We give initial experimental results showing that the number of Steiner points obtained with our approach is smaller compared to the number obtained when traditional circumcenter refinement methods are used.

2 Preliminaries

A *Planar Straight Line Graph* (PSLG) is a set of points and segments satisfying two constraints: all endpoint segments are points in the PSLG, segments may intersect each other only at their endpoints.

A triangulation \mathcal{T} is a *conforming triangulation* of a PSLG, Ω , if each point in Ω corresponds to a *vertex* in \mathcal{T} , and a *segment* of Ω is represented by a linear contiguous sequence of *edges* of \mathcal{T} . New Steiner vertices, not points of Ω , may appear, and each segment of Ω may have been subdivided into shorter edges by these additional vertices. Flipping these edges is forbidden, then they are marked as *locked*. In a *conforming Delaunay triangulation* of a PSLG, the Steiner vertices are added so that the Delaunay property is maintained.

The *star* of a vertex q , \mathcal{S}_q , of a triangulation \mathcal{T} consists of all the triangles of \mathcal{T} that contain q . The *link* of q , \mathcal{L}_q , is the polygon determined by the set of edges of the triangles in \mathcal{S}_q that are not incident to q . Since the average degree of a node in a planar graph is less than six [3], the average number of triangles of \mathcal{S}_q or the average number of edges of \mathcal{L}_q , is at most six. The *kernel* of \mathcal{L}_q , denoted by $\ker(\mathcal{L}_q)$, is the set of all points $p \in \mathcal{L}_q$, such that for every vertex v of \mathcal{L}_q , the segment vp is within \mathcal{L}_q .

Given an edge $e \in \mathcal{L}_q$, being e_i and e_f its endpoints, we take the following notation (see Figure 2):

- $H_{q,e}$ is the open half-plane determined by e and containing the vertex q .
- $t_{q,e}$ is the triangle with vertices e_i , e_f and q .
- $t'_{q,e}$ the adjacent triangle to $t_{q,e}$ by e .
- $c_{q,e}$ the circumcircle of $t'_{q,e}$.
- $a_{q,e}$ the arc $c_{q,e} \cap H_{q,e}$. We will say that $c_{q,e}$ is the supporting circle of $a_{q,e}$.

A triangle having an angle $\beta < \alpha$, for certain fixed α , is called *skinny*.

The *diametral circle* of a subsegment (portion of a segment from a PSLG) is the (unique) smallest circle that contains the subsegment.

A subsegment is said to be *encroached* if a vertex lies strictly inside its diametral circle, or if the subsegment does not appear in the triangulation.

2.1 Incremental Delaunay Algorithm

There exists three types of algorithms for constructing Delaunay triangulations, namely, divide-and-conquer, swepline and incremental. Because of our goals we concentrate our attention in the latter ones.

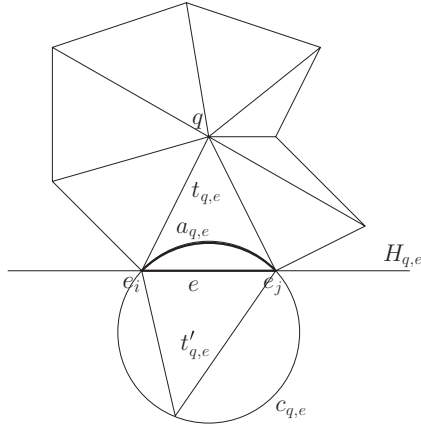


Fig. 2. Notation used in the definitions.

Incremental algorithms add vertices one by one and update the triangulation after each vertex is added maintaining the Delaunay property. The original algorithm, developed by Lawson [9], is based upon edge flips. There are incremental algorithms due to Bowyer [2] and Watson [14] that do not use edge flips. In Lawson’s algorithm, when a vertex is inserted, the triangle that contains it is found, and three new edges are inserted to attach the new vertex to the vertices of the containing triangle. If the new vertex falls upon an edge of the triangulation, that edge is deleted, and four new edges are inserted to attach the new vertex to the vertices of the containing quadrilateral. Next, a recursive procedure tests whether the new vertex lies within the circumcircles of any neighboring triangles, each affirmative test triggering an edge flip that removes a locally non-Delaunay edge. Each edge flip reveals two additional edges that must be tested.

2.2 Ruppert’s Algorithm

The Delaunay refinement algorithm, first described by Ruppert [12], refines the mesh by inserting additional *Steiner* vertices (using Lawson’s algorithm to maintain the Delaunay property) until all triangles satisfy the quality constraint. Vertex insertion follows two rules:

- Any *encroached* subsegment is bisected by inserting a vertex at its midpoint.
- Each *skinny triangle* is normally split by inserting a vertex at its circumcenter. The Delaunay property guarantees that the triangle is eliminated. However, if the new vertex would encroach upon any subsegment, then it is not inserted; instead, all the subsegments it would encroach upon are split.

Encroached subsegments are given priority over skinny triangles.

Ruppert’s algorithm guarantees the following properties:

- *Edges of the mesh well-graded.* New edges generated are greater than the smallest distance between two non-incident features of the input PLSG.
- *Optimal size of the mesh.* The number of Steiner points added is within a constant factor of the minimum number of points added by *any* meshing of the same input.
- *Termination.* In order to ensure Ruppert’s algorithm termination the upper bound for α is $\arcsin \frac{1}{2\sqrt{2}} \approx 20.7^\circ$, angles between incident segments in the input PSLG greater or equal than 90° are required, and co-circular points are not allowed. Shewchuk [13] relaxes this minimum angle requirement to 60° . Pav, in [11], showed that the algorithm terminates for a wider class of input than previously suspected, establishing this bound in 45° . In fact, other considerations have to be taken into account with respect to the input, but they rarely appear in practice.

3 Quality Zones

One of the main ideas behind our proposed algorithms is that some skinny triangles can be eliminated by moving the Steiner points corresponding to their bad vertex. The solution relies on the fact that, for a given Steiner vertex, it is possible to define a zone where it can be moved ensuring that a prefixed mesh quality can be obtained. To make sure that the result is a valid triangulation, a Steiner vertex q has to be moved to a point in $\ker(\mathcal{L}_q)$ and then the Delaunay property among the new triangles has to be maintained by flipping locally non-Delaunay edges.

3.1 Definitions

Let \mathcal{T} be a Delaunay refined triangulation of a PSLG and α be a given quality angle.

The *feasible zone of an edge* $e \in \mathcal{L}_q$ for an angle β is the set $\mathcal{F}_{e,\beta} = \{p \in H_{q,e} | \widehat{e_i p e_f} \geq \beta, \widehat{p e_f e_i} \geq \beta, \widehat{e_f e_i p} \geq \beta\}$ (see Figure 3(a)).

As can be observed in Figure 3(a), the feasible zone is a convex region obtained as intersection of two half-planes and a circle. Let d be the point in $H_{q,e}$ such that $e_i e_f d$ is an isosceles triangle with $\widehat{d e_i e_f} = \widehat{e_i e_f d} = \beta$. One half-plane is defined by the line through e_i and d that does not contain e_f , and the other half-plane is defined by the line through e_f and d' that does not contain e_i . The center f of the circle is the point of $H_{q,e}$ located on the bisector of $e_i e_f$, such that $\widehat{e_i f e_f} = 2\beta$. Then, from a well known geometric property, at any point p on the circle boundary the following expression is satisfied: $\widehat{e_i p e_j} = \beta$.

The *feasible zone* of a vertex q for an angle β is the set $\mathcal{F}_{q,\beta} = \bigcap_{e \in \mathcal{L}_q} \mathcal{F}_{e,\beta}$.
 The *non-skinny zone* of a vertex q is the set $\mathcal{F}_{q,\alpha}$.

The *Delaunay zone* of an edge $e \in \mathcal{L}_q$, denoted $\mathcal{D}_{q,e}$, is the set of points of $H_{q,e}$ external to $c_{q,e}$ (see Figure 3(b)).

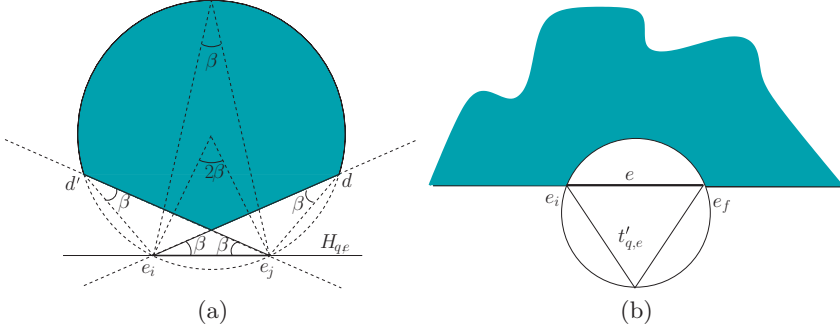


Fig. 3. (a) A feasible zone $\mathcal{F}_{q,e,\beta}$. (b) Delaunay zone $\mathcal{D}_{q,e}$.

The *Delaunay zone* of a vertex q is the set $\mathcal{D}_q = \bigcap_{e \in \mathcal{L}_q} \mathcal{D}_{q,e}$.

The Delaunay zone of a vertex is an open non-convex set and, as exhibited in Figure 4, may be constituted by several non-connected components. The boundary of \mathcal{D}_q will be denoted by $\overline{\mathcal{D}_q}$.

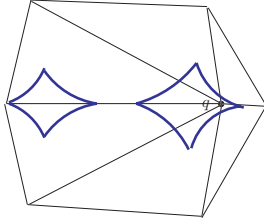


Fig. 4. Delaunay zone with two non-connected components.

The *quality zone* of a vertex q for the angle α is the set $\mathcal{Q}_{q,\alpha} = \mathcal{F}_{q,\alpha} \cap \mathcal{D}_q$ (see Figure 5).

From the last definition it is clear that if $p \in \mathcal{Q}_{q,\alpha}$ then the triangles of \mathcal{S}_p are non-skinny and the exterior adjacent triangles to \mathcal{L}_p edges are Delaunay.

It is not difficult to prove the following:

Lemma 1. *Let q be a vertex of a triangulation \mathcal{T} . Then, we have:*

- If $\mathcal{F}_{q,\beta} \neq \emptyset$, $\mathcal{F}_{q,\beta}$ is a convex set included in $\ker(\mathcal{L}_q)$.
- If $\mathcal{F}_{q,\beta} \neq \emptyset$ and $\beta' > \beta$, $\mathcal{F}_{q,e,\beta'} \subsetneq \mathcal{F}_{q,e,\beta}$ and $\mathcal{F}_{q,\beta'} \subsetneq \mathcal{F}_{q,\beta}$.

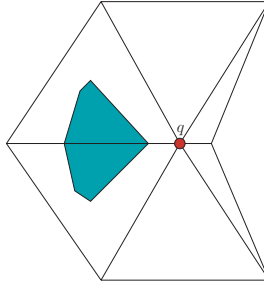


Fig. 5. Quality zone $\mathcal{Q}_{q,\alpha}$

3.2 Finding a Point That Maximizes the Minimum Angle

Let p be a point in $\ker(\mathcal{L}_q)$. We denote by $T_q(p)$ the set of triangles determined by p and the edges of \mathcal{L}_q . If \mathcal{L}_p is formed by k edges, we have a collection of $3k$ angular functions $\phi_j(p)$, $j = 1, \dots, 3k$, each one representing an angle of a triangle of $T_q(p)$.

We are interested in finding a point $\tilde{p} \in \ker(\mathcal{L}_q) \cap \mathcal{D}_q$ maximizing the function $\Phi(p) = \min_j \phi_j(p)$. When instead of searching for the point $\tilde{p} \in \mathcal{D}_q$ we seek a point $p^* \in \ker(\mathcal{L}_q)$, we have a collection of quasiconvex functions and the problem can be solved in $O(k)$ time using Generalized Linear Programming [1].

Then, we first find the point p^* . If $p^* \in \mathcal{D}_q$ then p^* is the optimal solution \tilde{p} . Otherwise we find the point \bar{p} that maximizes the function $\Phi(p)$ restricted to $\overline{\mathcal{D}_q}$ and we take a point inside \mathcal{D}_q close to \bar{p} as the target point \tilde{p} .

Instead of implementing the Generalized Linear Programming approach for finding p^* , we describe an alternative numerical technique for solving the problem directly that also allow us to find \bar{p} if it is necessary. The technique is based on the following lemmas:

Lemma 2. *Let l be a line such that $l \cap \mathcal{F}_{q,\beta} \neq \emptyset$. Then, we have:*

1. $l \cap \mathcal{F}_{q,\beta}$ is a point or a segment.
2. If $l \cap \mathcal{F}_{q,\beta}$ is a segment, the midpoint m of $l \cap \mathcal{F}_{q,\beta}$ satisfies $\Phi(m) \geq \beta$.

Proof. Let $s = l \cap \mathcal{F}_{q,\beta}$. Since $l \cap \mathcal{F}_{q,\beta}$ is a convex set, s is a point or a segment. Let m be the midpoint of s . Suppose that $\Phi(m) < \beta$. By Lemma 1 we know that $\mathcal{F}_{q,\beta} \subset \mathcal{F}_{q,\Phi(m)}$ and consequently m is an interior point of $\mathcal{F}_{q,\Phi(m)}$, which is contradictory with the fact that m is on the boundary of $\mathcal{F}_{q,\Phi(m)}$.

Lemma 3. *Let $a_{q,e}$ be a Delaunay arc whose supporting circle $c_{q,e}$ contains p^* . Let p' be the point that maximizes $\phi(p)$ restricted to $a_{q,e}$, and let \bar{p} be the point that maximizes $\phi(p)$ restricted to $\overline{\mathcal{D}_q}$. If $p' \in \overline{\mathcal{D}_q}$ then $\phi(p') = \phi(\bar{p})$.*

Proof. Since $p' \in \overline{\mathcal{D}_q}$, clearly $\phi(p') \leq \phi(\bar{p})$. Suppose $\phi(p') < \phi(\bar{p})$. Since p' maximizes $\phi(p)$ restricted to $a_{q,e}$, $\mathcal{F}_{q,\phi(p')}$ is tangent to $c_{q,e}$ at p' . Moreover, due to that $\mathcal{F}_{q,\phi(p')}$ is convex, $c_{q,e}$, $p^* \in \mathcal{F}_{q,\phi(p')}$ and $p^* \in c_{q,e}$, we have $\mathcal{F}_{q,\phi(p')} \subset c_{q,e}$. By Lemma 1 we have $\mathcal{F}_{q,\phi(\bar{p})} \subsetneq \mathcal{F}_{q,\phi(p')}$, consequently $\mathcal{F}_{q,\phi(\bar{p})} \subsetneq c_{q,e}$, which is contradictory with the fact that $\bar{p} \in \mathcal{D}_q$.

The maximizing algorithm consists of two main steps, first we find the point p^* that maximizes Φ . If p^* happens to lie inside the Delaunay Zone, \mathcal{D}_q , then it is the target point \bar{p} ; on the contrary we have to find \bar{p} on \mathcal{D}_q . Both steps are based on the optimal gradient method that progressively approaches the point maximizing a function. This method needs the direction $v(p)$ in which the function increases more quickly. In our case we approximate $v(p)$ as follows (see Figure 6):

- 1 Determine the minimum angle β of $T_q(p)$.
- 2 If β is adjacent to p , let $v(p)$ be the direction of its angle bisector.
- 3 If β is not adjacent to p , let r be the vertex of β and let $v(p)$ be the perpendicular vector to rp satisfying that the half-plane determined by p and $v(p)$ does not contain the other vertex of the triangle.

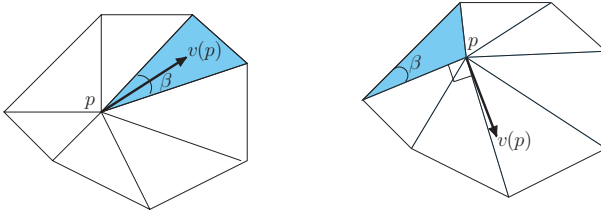


Fig. 6. On the left, β is adjacent to p and increases in the direction of its angle bisector. On the right, β is adjacent to an edge of \mathcal{L}_q and increases in the direction perpendicular to the edge adjacent to p and β .

Figure 7 illustrates the first step of the maximizing algorithm that finds p^* based on Lemma 2. Let p_k be the point obtained after the k -th iteration of the algorithm. The point q is used to compute the initial iteration p_0 . The point p_{k+1} is computed from p_k by applying the following steps:

- 1 Compute the minimum angle β_k of $T_q(p_k)$ and the vector $v(p_k)$.
- 2 Compute the feasible zone \mathcal{F}_{q,β_k} .
- 3 Compute the segment s as the intersection between \mathcal{F}_{q,β_k} and the half-line determined by p_k and $v(p_k)$.
- 4 Take p_{k+1} as the midpoint of s .

The algorithm finishes when the distance between p_k and p_{k+1} is lesser than ε and the difference between β_k and β_{k+1} is lesser than δ , where ε and δ are parameters that permit controlling the accuracy of the solution.

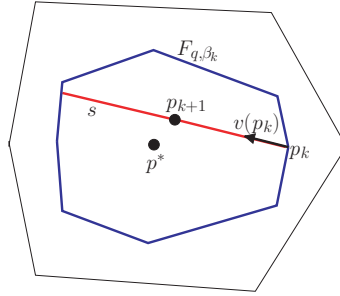
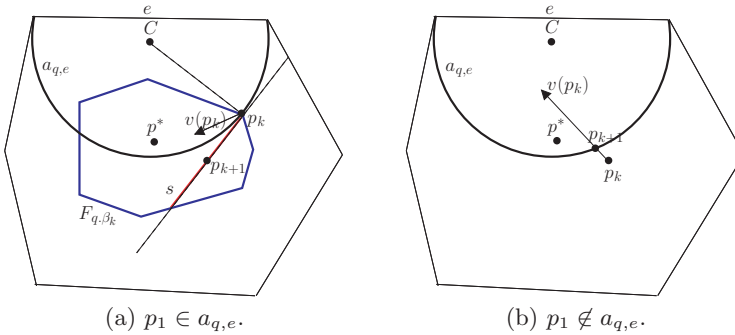


Fig. 7. First step of the maximizing algorithm.

Figure 8 illustrates the second step of the maximizing algorithm, based on Lemma 3, that finds a point \bar{p} on \overline{D}_q maximizing $\Phi(p)$. Let $a_{q,e}$ be a Delaunay arc whose supporting circle $c_{q,e}$ contains p^* . Any point of $a_{q,e}$ is qualified to compute the initial iteration p_0 .



(a) $p_1 \in a_{q,e}$. (b) $p_1 \notin a_{q,e}$.
Fig. 8. Second step of the maximizing algorithm.

There are two cases for computing p_{k+1} from p_k . The first one is triggered when p_k lies on $a_{q,e}$ (see Figure 8(a)). In this case we apply the following steps:

- 1 Compute the minimum angle β_k of $T_q(p_k)$ and the vector $v(p_k)$.
- 2 Compute the orthogonal projection v of $v(p_k)$ onto the tangent line to $c_{q,e}$ at p_k .
- 3 Compute the feasible zone \mathcal{F}_{q,β_k} .
- 4 Compute the segment s intersection between \mathcal{F}_{q,β_k} and the half-line determined by p_k and v .
- 5 Take p_{k+1} as the midpoint of s .

The second case occurs when p_k does not lie on $a_{q,e}$ (see Figure 8(b)). In this case the point p_{k+1} is taken as the intersection point between $a_{q,e}$ and the half-line determined by p_k and $v(p_k)$.

Only when the point \bar{p} lies on $\overline{\mathcal{D}_q}$, the point \tilde{p} is taken as the last iteration point not lying on $a_{q,e}$. Then, the described process must be applied to Delaunay arcs whose supporting circle contains p^* until the point \tilde{p} is found. If the process fails for all these arcs, we compute the set P of the intersection points between the arcs, and we take \tilde{p} as an interior point of \mathcal{D}_q very close to the point of P maximizing $\Phi(p)$.

4 Basic Operations

Movement and deletion of Steiner points are the *basic operations* used by our improvement process. Steiner points to be treated by the process can belong to two main groups. The first group is formed by the Steiner points located on any segment of the PSLG, and the second group is formed by the remaining Steiner points. We have named *restricted* vertices the points of the first group, since their movement will be restricted to the corresponding segment, and *free* vertices the points of the second group.

4.1 Moving Free Vertices

The key concept regarding the movement of a free vertex q is to substitute this vertex by the best point in the quality zone $\mathcal{Q}_{q,\alpha}$, being α the quality of the mesh. Then, we have to find a point $\tilde{p} \in \mathcal{D}_q$ maximizing the function $\Phi(p)$ and satisfying $\Phi(\tilde{p}) \geq \alpha$. In order to do that, we apply the maximizing algorithm explained in the previous section. Observe that the simple insertion of \tilde{p} into the triangulation guarantees that exterior triangles to \mathcal{L}_q are Delaunay, but does not guarantee the Delaunay property among interior triangles. For this reason, the vertex q has to be deleted and the vertex \tilde{p} has to be inserted using an Incremental Delaunay algorithm. Then, it is possible that $\mathcal{L}_{\tilde{p}} \neq \mathcal{L}_q$ and, consequently, $\Phi'(\tilde{p}) \geq \Phi(\tilde{p})$, where $\Phi'(p)$ is the function to be maximized in the interior of $\mathcal{L}_{\tilde{p}}$. In this case, we initiate an iterative process that in each step updates q with the vertex \tilde{p} obtained in the previous step, applies the optimizing algorithm to q and inserts the new \tilde{p} using an Incremental Delaunay algorithm. The process finishes when $\mathcal{L}_{\tilde{p}} = \mathcal{L}_q$. Only when the final \tilde{p} satisfies $\Phi(\tilde{p}) \geq \alpha$, the point \tilde{p} is inserted into the mesh as a Steiner vertex.

4.2 Deleting Free Vertices

Devillers in [4] proposed an algorithm to delete a point from a Delaunay triangulation. Basically, his algorithm retriangulates \mathcal{L}_q by determining *Delaunay*

ears using the concept of *power* of q . In our case we need to obtain not just a Delaunay triangulation, but a Delaunay refined one. Consequently we verify whether the Delaunay ear is skinny or not, stopping the process when a skinny one is found.

4.3 Moving Restricted Vertices

The movement of restricted vertices is constrained over their correspondent subsegments. This kind of vertices can be present on a boundary subsegment or on a non-boundary subsegment of a PSLG. Since the optimizing algorithm can easily be adapted in order to guarantee that the vertex \tilde{p} lies on the subsegment, in both cases we apply the iterative process explained in Section 4.1.

4.4 Deleting Restricted Vertices

Deletion of a restricted vertex depends on whether it belongs or not to a boundary subsegment of the PSLG. The presence of a restricted vertex on a non-boundary subsegment implies the application of the deletion algorithm explained in Section 4.2 to the two sides of the subsegment independently. In this way the point is deleted only if each side independently fulfills the point deletion verification, and then the region in each side is retriangulated individually. The same steps from the algorithm of Section 4.2 can be carried out without changes, with the only extra consideration that a restricted vertex on a subsegment can not be deleted if a vertex of its link is encroached by the subsegment. In case of a boundary subsegment the process detailed above is applied only to the interior side.

4.5 Expanding Deletion of Vertices

Once a vertex has been deleted from the mesh other vertices are susceptible of deletion. Each time a vertex is deleted all its adjacent Steiner vertices are added to a queue to be deleted, causing in this way several iterations. The iteration process ends when none of the vertices in the queue can be deleted.

5 Modifying a Delaunay Refined Mesh

Modification of a Delaunay refined mesh means to insert new PSLG elements into the mesh or delete PSLG elements from the mesh. The elements can be points, segments, polygonal lines and polygonal holes.

An element is inserted in the mesh by inserting its points and then checking if each segment of the element is a sequence of edges of the mesh. If the check fails, the segment is inserted by a recursive process that adds its midpoint and

checks if the two subsegments are edges of the mesh. The edges corresponding to segments are marked as locked. When the element is a polygonal hole, the triangles located in the interior of the hole are deleted.

An element is deleted by marking its edges as unlocked, considering its points Steiner vertices and trying to delete them by using the Devillers algorithm. When the element is a polygonal hole, we first triangulate its interior.

After the insertion or deletion of an element, a process of improvement is called that expands the quality through the mesh. We could follow two approaches: to apply our improvement process each time a part of the element is inserted or deleted, or apply the process right after the whole element is inserted or deleted. This situation has been illustrated with the insertion of a segment in Figure 9 (first approach) and in Figure 10 (second approach). As can be observed in the figures, the use of the improvement process after each point insertion increases the number of Steiner points. Consequently, in our algorithms we will follow the second approach.

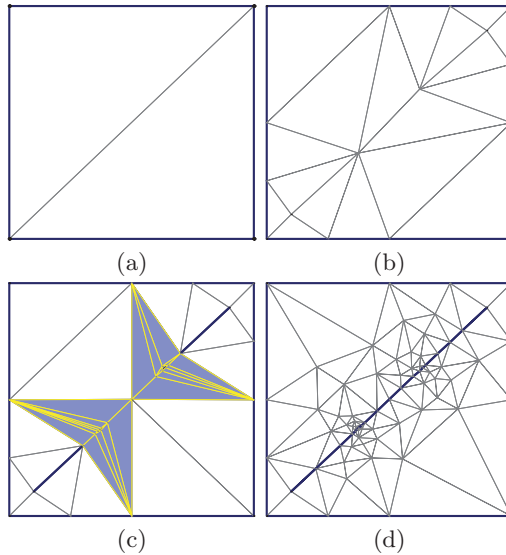


Fig. 9. (a) Initial mesh of a PSLG formed by a square boundary. (b) The mesh after the insertion of two points. (c) The resulting mesh with skinny triangles generated by the insertion of a segment whose endpoints are the points previously added. (d) Resulting mesh obtained applying an improvement process after each partial element addition.

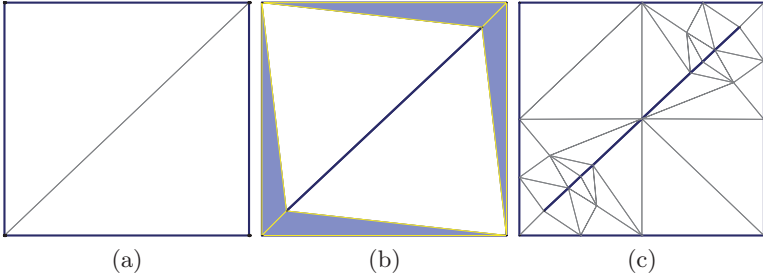


Fig. 10. (a) Initial mesh of a PSLG formed by a square boundary. (b) The resulting mesh with skinny triangles generated by the insertion of a segment. (c) Resulting mesh delaying improvement process at the end.

5.1 Improvement Process

The improvement process receives as input two lists: the list of points to be inserted, initially containing only midpoints of encroached subsegments, and the list of skinny triangles to be removed, originated by the modification of an element. The output of the process is a mesh with the desired quality. The process maintains the two lists and finishes when both are empty. Priority is established on midpoints. To remove a skinny triangle, we first check its Steiner vertices for deletion, then we check its Steiner vertices for movement, and finally we add circumcenters to the list of points. This order of treatment of skinny triangles is important to obtain a reduction in the number of vertices. Following the rule from Ruppert's algorithm encroached circumcenters are not inserted and the midpoint of the encroached subsegments are added to the list of points to be inserted.

6 Generating a Delaunay Refined Mesh

As stated in the introduction, our improvement process can also be applied to generate a refined Delaunay quality mesh of a PSLG. The complete process consists of the following steps. First, a conforming Delaunay triangulation of the PSLG is generated, then the list of skinny triangles to be removed is obtained, and finally our improvement method is applied to eliminate those skinny triangles. Observe that initially the list of points to be inserted is empty.

7 Experimental Results

We have implemented our algorithms in C++ language and using OpenGL libraries to build an interactive interface. Our application takes a triangulated

PSLG as input. This initial mesh is refined until the desired quality is achieved. Also, the mesh can be modified by adding or deleting elements while keeping the quality established.

We have run several simulations in order to test our implementation and we have compared these simulations with meshes generated using *TRIANGLE*, a freely available software produced by Jonathan R. Shewchuck [13] (<http://www.cs.cmu.edu/~quake/triangle.html>), and an incremental Ruppert algorithm based on the work of Miller et al. [10]. In table 1 we present the statistics and the reference to the correspondent set of outputs. The input PSLG is composed of a square boundary and a polygonal hole described by 20 points. The final PSLG consists of the initial PSLG plus four polygonal holes each one of them composed of 10 points. In incremental Ruppert and in our approach these last four holes are added to the mesh one at a time. Tests have been carried out varying the quality requirement. The first column of table 1 shows the quality measures considered. The following columns show the number of triangles of the initial mesh and those generated by *TRIANGLE*, the incremental Ruppert algorithm, finishing with our algorithm. It can be observed in the results obtained that the number of triangles generated by the incremental Ruppert, or from the scratch are higher than applying our algorithm. Also notice that the percentage of the increment increases as the quality angle gets higher.

Table 1. Number of triangles obtained after the insertion of four holes.

α	Initial mesh	From scratch	Incremental Ruppert	Our approach	Figure
20°	124	439	421	314	11
25°	170	514	547	413	12
30°	257	717	1275	565	13
32°	350	1745	2771	674	14

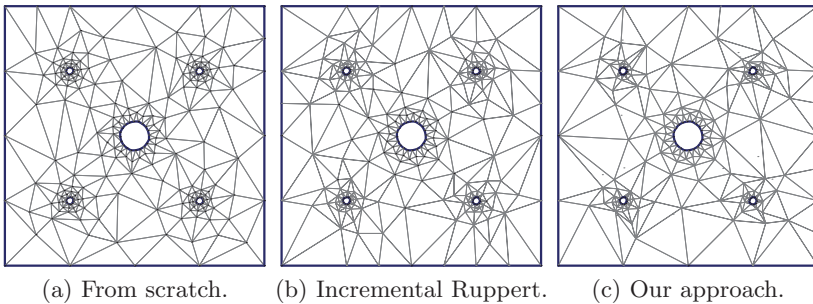


Fig. 11. Results obtained for an angle $\alpha = 20^\circ$.

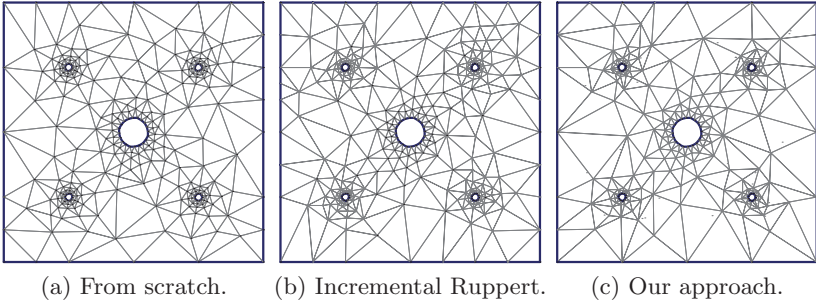


Fig. 12. Results obtained for an angle $\alpha = 25^\circ$.

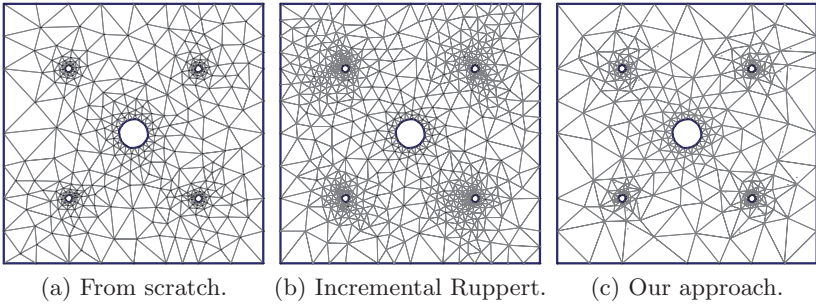


Fig. 13. Results obtained for an angle $\alpha = 30^\circ$.

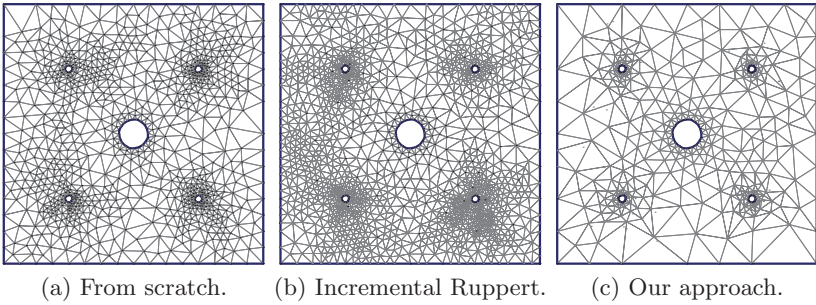


Fig. 14. Results obtained for an angle $\alpha = 32^\circ$.

All previous examples deal with the addition of elements to the initial PSLG, but our algorithm allows us to delete elements from the PSLG. Figure 15 shows an example of segment deletion. Notice that the mesh obtained after the segment deletion is quite similar to the initial mesh.

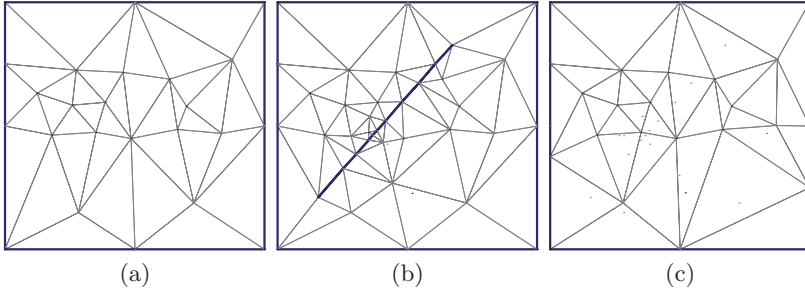


Fig. 15. (a) Initial mesh. (b) A mesh with a segment to be deleted. (c) Resulting mesh after the segment deletion.

Finally, we present some initial results obtained when we use our improvement method for Delaunay refined mesh generation. The PSLG used models the Lake Superior. Figure 16(a) shows the mesh generated taking 34° as quality angle by *TRIANGLE*, and Figure 16(b) the result after applying our method.

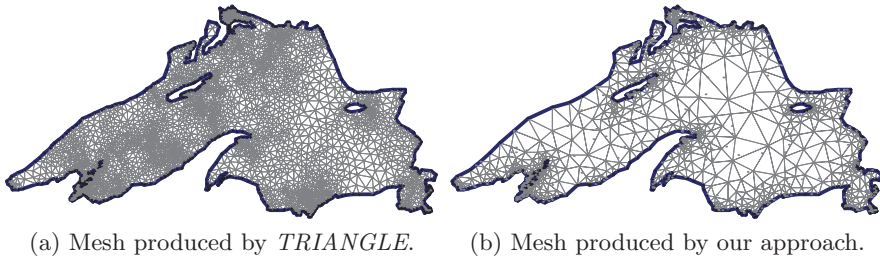


Fig. 16. Results obtained for an angle $\alpha = 34^\circ$.

8 Future Work

Future work includes an exhaustive analysis of the conditions in which the algorithm terminates.

A large experimentation with other input PSLGs is necessary to confirm our initial results in Delaunay refined mesh generation.

Our ultimate goal is to extend our framework towards the modification and generation of three dimensional meshes.

Acknowledgments

The authors wish to thank Frederic Pérez for his help in preparing this paper, and to the reviewers for their comments and suggestions.

References

1. N. Amenta, M. Bern and D. Epstein. Optimal point placement for mesh smoothing. *In SODA:ACM-SIAM Symposium on Discrete Algorithms*, 1997.
2. A. Bowyer. Computing Dirichlet Tessellations. *Computer Journal*, 24:2:162–166, 1981.
3. K. Clarkson and K. Mehlhorn and R. Seidel. Four results on randomized incremental constructions. *Computational Geometry: Theory and Applications*, 3:185–212, 1993.
4. O. Devillers. On deletion in Delaunay triangulation. *15th Annual ACM Symposium on Computational Geometry*, 181–188, 1999.
5. L. Freitag and M. Jones and P. Plassmann. An efficient parallel algorithm for mesh smoothing. *In Proceedings of the Fourth International Meshing Roundtable*, 47–58, 1995.
6. N. Han-Wen and A.-F. van der Stappen. A Delaunay approach to interactive cutting in triangulated surfaces. *In J.D. Boissonnat, J. Burdick, K. Goldberg & S. Hutchinson (Eds.), Algorithmic Foundations of Robotics V, Springer-Verlag*, 113-129 , 2004.
7. S. Har-Peled and A. Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. *21st Annual ACM Symposium on Computational Geometry (SoCG)*, 228–229, 2005.
8. M. Kallmann and H. Bieri and D. Thalmann. Fully Dynamic Constrained Delaunay Triangulation. *Geometric Modelling For Scientific Visualization - Springer-Verlag*, 2003.
9. C.-L. Lawson. Software for C^1 Surface Interpolation. *Mathematical Software III(John R.Rice, editor)*, 161–194, 1977.
10. G.-L. Miller and S.-E. Pav and N.-J. Walkington. Fully incremental 3d Delaunay mesh generation. *In Proceedings 11th International Meshing Roundtable*, 75–86, 2002.
11. S.-E. Pav. Delaunay Refinement Algorithms. *Department of Mathematical Sciences - Carnegie Mellon University - PhD thesis*, 2003.
12. J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18:3:548–585, 1995.
13. J.-R. Shewchuk. Delaunay Refinement Mesh Generation. *School of Computer Science - Carnegie Mellon University - PhD thesis*, 1997.
14. D.-F. Watson. Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *Computer Journal*, 24:2:167–172, 1981.

The Cost of Compatible Refinement of Simplex Decomposition Trees

F. Betul Atalay¹ and David M. Mount²

¹ Mathematics and Computer Science Department, Saint Joseph's University,
Philadelphia, PA. fatalay@sju.edu

² Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD. mount@cs.umd.edu

Summary. A hierarchical simplicial mesh is a recursive decomposition of space into cells that are simplices. Such a mesh is *compatible* if pairs of neighboring cells meet along a single common face. Compatibility condition is important in many applications where the mesh serves as a discretization of a function. Enforcing compatibility involves refining the simplices of the mesh further, thus generates a larger mesh. We show that the size of a simplicial mesh grows by no more than a constant factor when compatibly refined. We prove a tight upper bound on the expansion factor for 2-dimensional meshes, and we sketch upper bounds for d -dimensional meshes.

1 Introduction

Hierarchical data structures based on repeated subdivision of space have been widely used in application areas such as finite element analysis, computer graphics, scientific visualization, geometric modeling, image processing and geographic information systems. In many such applications, the spatial decomposition serves as a discretization of the domain of a scalar or vector field, which associates each point of real d -dimensional space with a scalar or vector value, respectively. The field values are sampled at the vertices of the subdivision, and for any other query point the field value could be computed by an appropriate (often linear) interpolation of the field values at the vertices of the cell that contains it. The subdivision is adaptively refined to improve the approximation of the field at regions of high variation.

Our interest in this paper is on simplicial decompositions, particularly on *regular hierarchical simplicial meshes* [10, 2]. This is a generalization of the concept of hierarchical regular triangulation in the plane. Each element of such a mesh is a d -simplex, that is, the convex hull of $d + 1$ affinely independent points [4]. A simplicial mesh is said to be *regular* if the vertices of the mesh are regularly distributed and the process by which a cell is subdivided is identical for all cells. The regular simplicial mesh that we consider is generated by a

process of repeated bisection applied to a hypercube that has been initially subdivided into $d!$ congruent simplices. The subdivision pattern repeats itself on a smaller scale at every d levels.

A simplicial mesh is called *compatible* if pairs of neighboring cells meet along a single common face. A compatible simplicial mesh is also referred to as a simplicial complex. (See Fig. 1 for a 2-dimensional example.) The compatibility condition is important since otherwise cracks may occur along the faces of the subdivision, which in turn causes discontinuities in the function and presents problems when using the mesh for interpolation. A compatible mesh ensures at least C^0 continuity and is desirable for many applications. 2-dimensional simplicial meshes have been used for multi-resolution terrain modeling and rendering [5, 9, 3, 12, 6]; 3-dimensional meshes for volume rendering of 3-dimensional scalar fields (such as medical datasets) [7, 14], and 4-dimensional meshes for visualization of time-varying flow fields.

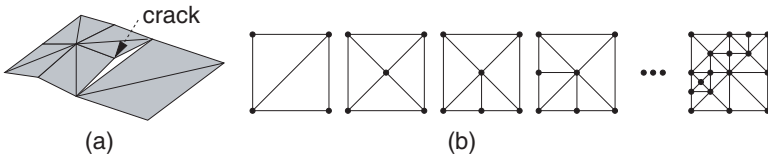


Fig. 1. (a) A crack (b) Compatible simplicial mesh in the plane

Refining a simplicial mesh to enforce compatibility requires refining additional simplices if they share split faces with their neighbors. The cost of compatible refinement is that a larger mesh will be generated. Our goal in this paper is to show that when a simplicial mesh is refined to enforce compatibility, its size will grow by no more than a constant factor. We prove a tight upper bound on the expansion factor for 2-dimensional meshes, and upper bounds for d -dimensional meshes.

Previously, Weiser [13] and Moore [11] proved results on the cost of *restricting* quadtrees. A *restricted quadtree* is a quadtree in which two neighboring leaf cells in the quadtree may differ at most by one level [8]. Moore calls a restricted quadtree as a *1-balanced quadtree*. Weiser showed that a square quadtree grows no more than nine times bigger and a triangular quadtree grows no more than thirteen times bigger when refined to provide 1-balance [13]. Moore later showed Weiser's bounds could be reduced by showing that a square quadtree grows at most eight times larger, and triangular quadtrees grow ten times larger when refined for 1-balance [11]. Moore also showed that his bounds are tight.

We follow Moore's methodology to show similar results for the family of bisection-based regular hierarchical simplicial meshes. Note that, Moore's triangular quadtrees are constructed by repeatedly subdividing a triangle into four smaller triangles which are similar to the original triangle. Moore also

analyzes degree-three triangular “quadtrees” where each triangle is subdivided into nine smaller similar triangles. In any case, all the triangles in the entire triangular quadtree are similar triangles. The simplicial meshes that we consider in this paper arises from a different family of meshes, which are constructed by repeatedly subdividing a simplex into two child simplices which are congruent to each other but not similar to their parent. Thus, the subdivision rule is different from a triangular quadtree and there is more than one similarity class of simplices in the mesh. This bisection-based subdivision rule can be applied in any dimension d . In addition, Moore’s analysis is based on 1-balancing, whereas we are interested in compatibility refinement which imposes a tighter requirement.

The remainder of the paper is organized as follows. In Section 2 we present basic definitions and describe regular hierarchical simplicial meshes. In Section 3 we prove an upper bound on the size of a 2-dimensional compatibly refined simplicial mesh. In Section 4, we prove that this upper bound is asymptotically tight. In Section 5, we sketch an upper bound for d -dimensional meshes.

2 Preliminaries

The *simplex decomposition tree* (*sd-tree* for short) is a collection of binary trees representing a regular hierarchical simplicial mesh in real d -dimensional space. Assume that the domain of interest has been scaled to lie within a unit *reference hypercube*. The reference hypercube is initially subdivided into $d!$ congruent simplices that share the major diagonal. It is well known that the collection of these simplices fully subdivide the hypercube, and further that this subdivision is compatible [1]. These $d!$ simplices form the starting point of our simplicial decomposition. Simplices are then refined by a process of repeated subdivision, called *bisection*, in which a simplex is bisected by splitting one of its edges at its midpoint. The edge to be bisected is determined by a specific vertex ordering [10, 2]. Intuitively, this bisection scheme alternates bisecting the major diagonal of the hypercube first, then the diagonals of the $d-1$ faces, then the diagonals of the $d-2$ faces, and so on, finally bisecting the edges (1-faces) of the hypercube. (In the 2-dimensional and the 3-dimensional case, this bisection scheme is equivalent to bisecting the longest edge of the simplex.) Hence, each of the $d!$ coarse simplices at the highest level is the root of a separate binary tree, which are conceptually joined under a common super-root corresponding to the hypercube. See Fig. 2 for a 2-dimensional subdivision and the corresponding *sd-tree*.

Define the *level*, ℓ , of a simplex to be the *depth* modulo the dimension, that is, $\ell = (p \bmod d)$, where p denotes the depth of a simplex in the tree. The depth of a root simplex is zero, and of any other simplex is one more than the depth of its parent.

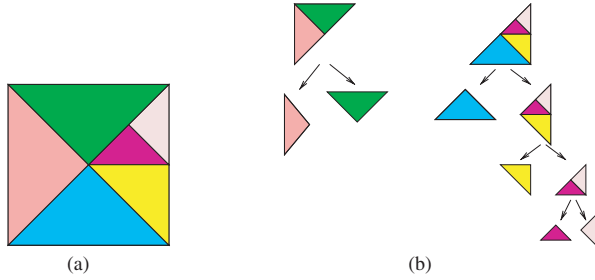


Fig. 2. (a) 2-dimensional simplicial subdivision (b) The corresponding sd-tree

Maubach [10] showed that with every d consecutive bisections, the resulting simplices are similar copies of their d -fold grandparent, subject to a uniform scaling by $1/2$. Thus, the pattern of decomposition repeats every d levels in the decomposition. Since the two children of any simplex are also *congruent*, it follows that all the simplices at any given level of the decomposition tree are congruent to each other. Thus, all the similarity classes can be represented by d canonical simplices, one per level. In Fig. 3(a) and (b) the shaded simplices denote the two canonical simplices for a 2-dimensional subdivision. Notice, for example, that any simplex in the subdivision shown in Fig. 1 is congruent to either one of the two canonical simplices. Consequently, it suffices to consider only the canonical simplices when analyzing the structure of the tree.

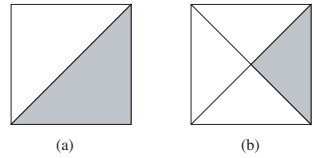


Fig. 3. (a) *level-0* simplex (b) *level-1* simplex

A d -dimensional simplex decomposition tree is said to be *compatible*, if each simplex in the subdivision shares a $(d - 1)$ -face with exactly one neighbor simplex. If the subdivision is not compatible, we can further refine simplices to provide compatibility. Consider the non-compatible subdivision in Fig. 4(a). The simplices that share a bisected edge with already bisected simplices need to be bisected as well. However, note that new bisections will possibly trigger more bisections at the higher levels of the tree. In Fig. 4(b) the dashed edges illustrate the bisections triggered due to compatibility refinement of the subdivision shown in (a).

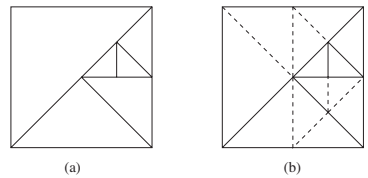


Fig. 4. (a) Before and (b) after compatible refinement

3 Upper Bound for 2-Dimensional Decompositions

In this section, we prove that the size of a 2-dimensional regular hierarchical simplicial mesh grows only by a constant factor when compatibly refined.

Theorem 1. *A non-compatible simplex decomposition tree in 2-dimensional space with n nodes can be compatibly refined to form a simplex decomposition tree with no more than $14n$ nodes.*

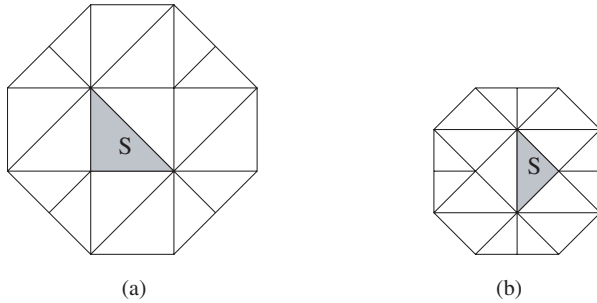


Fig. 5. (a) Barrier of a *level-0* simplex (b) Barrier of a *level-1* simplex

We follow a similar method as Moore [11] to prove this theorem. We start by finding a *barrier*, that is, a configuration of simplices around a particular simplex S . Then, we show that if such a barrier is produced after a series of splits (possibly none), then simplex S will never split during compatibility refinement. Recall that, in the 2-dimensional case, we have two canonical simplices which are shown in Fig. 3(a) and (b). We call them *level-0* simplices and *level-1* simplices, respectively. The barriers for each class of simplices are illustrated in Fig. 5(a) and (b). Before proving the above theorem, we first introduce the notion of a safe simplex and prove a lemma that shows that such simplices cannot be split.

Definition 1. (Safe Simplex) *A simplex S is safe if none of the barrier elements are split initially or such a barrier comes into existence after any series of splits.*

Lemma 1. *A safe simplex S will never split during compatibility refinement.*

Proof. (of Lemma 1)

We will prove the lemma by induction on the depth of the simplex in the simplex decomposition tree. Let p denote the depth of the simplex S .

Basis

If S is the deepest leaf, S will not split since a simplex will only split if it has a split neighbor.

Induction Step

Assume that our inductive hypothesis holds for simplices at depth $p + 1$. We will show that the inductive hypothesis holds for a simplex S at depth p . We need to consider the two canonical simplices separately.

Let us first consider the case where S is a *level-0* simplex, that is, $p \bmod 2 = 0$. Note that for S to split, either one of the three neighboring simplices shown in Fig. 6(a) should split. If none of the three neighbor simplices labeled 0, 1 and 2 split, S will never split. Suppose that S is initially surrounded by the barrier shown in Fig. 6(b), such that no barrier element is split. Even if all the boundary barrier elements that are at depth p split due to compatibility refinement, we will have the structure depicted in Fig. 6(c). Notice that the neighbors labeled 0 and 2 are at depth $p + 1$, and they have their own barriers. Thus, these neighbors are safe. The barrier for neighbor 0 is depicted with thick lines in Fig. 6(d), and the barrier for neighbor 2 is symmetric to that. Therefore, by the inductive hypothesis that was assumed to hold for simplices at depth $p + 1$, neighbors 0 and 2 need not split.

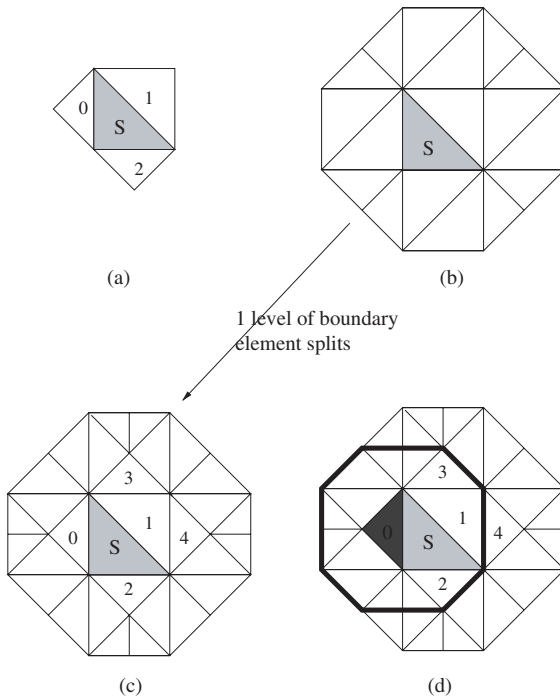


Fig. 6. Induction step for a *level-0* simplex

Similarly simplices labeled 3 and 4 at depth $p + 1$ are surrounded by a barrier and they will not split. If 3 and 4 do not split, neighbor simplex 1 will

not split. Thus, none of the neighbor simplices of S will split ensuring that S will not split. This concludes the case of a *level-0* simplex.

The case of S being a *level-1* simplex can be proved similarly. Suppose that S is initially surrounded by the barrier shown in Fig. 7(a), such that no barrier element is split. Even if all the boundary barrier elements that are at depth p split due to compatibility refinement, we will have the structure depicted in Fig. 7(b). Notice that if none of the three neighbor simplices of S which are labeled 0, 1 and 2 split, S will never split. Neighbors 1 and 2 which are at depth $p+1$ have their own barriers, and so, they are safe. The barrier for neighbor 1 is depicted with thick lines in Fig. 7(c). The barrier for neighbor 2 is symmetric. Therefore, neighbors 1 and 2 need not split. Similarly, simplices labeled 3 and 4 are at depth $p + 1$ and have their own barriers, preventing neighbor 0 to split. Since none of the neighbor simplices of S split, S will not split. \square

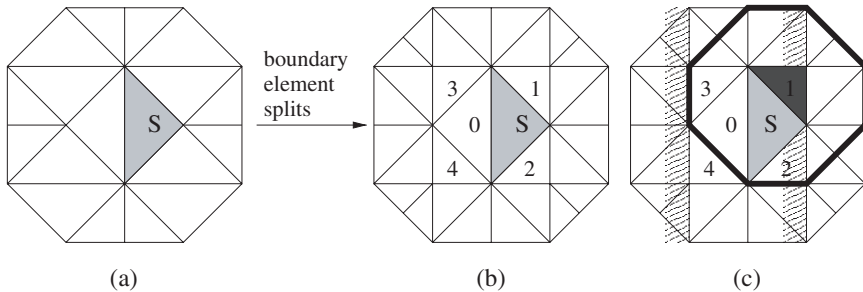


Fig. 7. Induction step for a *level-1* simplex

Proof. (of *Theorem 1*)

Using Lemma 1, if a simplex S splits, it must not be safe, that is, one or more of its barrier elements is initially split. We will hold one of its barrier elements responsible for splitting of S . A split element R may be responsible for splitting 13 other split elements, since it may be in the barrier of 13 possible elements as depicted in Fig. 8 when R is a *level-0* simplex, and in Fig. 9 when R is a *level-1* simplex. In Fig. 8, part (a) shows the nine possible *level-0* simplices whose barriers contains R , and part (b) shows the four possible *level-1* simplices whose barriers may contain R . Thus, compatibly refining a simplex decomposition tree could increase the number of nodes by at most a factor of 14. \square

4 Tightness of the Upper Bound

In this section, we demonstrate that the upper bound of *Theorem 1* is asymptotically tight by constructing an infinite family of simplex decomposition trees, each of which grow fourteen times larger less an additive constant, when

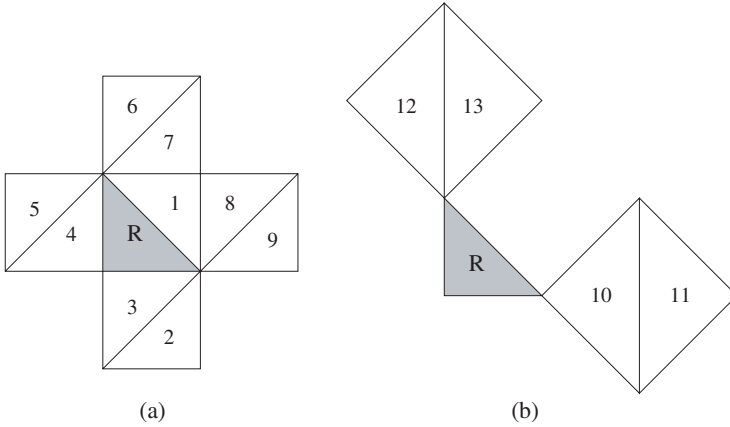


Fig. 8. 13 simplices whose barriers contain *level-0* simplex R

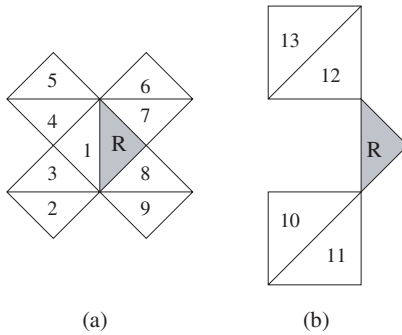


Fig. 9. 13 simplices whose barriers contain *level-1* simplex R

compatibly refined. Let T_i denote the i -th tree in this family of trees. T_i is constructed as follows. We start with the tree shown in Fig. 10(a). Initially, we designate the simplex with thick borders as the *next* simplex to split. Each split generates two new child simplices. After the split we update *next* to be the child simplex which has the central vertex of the subdivision as one of its vertices. To construct T_i , we split the *next* simplex $2i$ times. Figure 10(b) shows such a subdivision after six splits ($i = 3$). To complete the construction of T_i , we replace the *next* simplex with the subdivision shown in Fig. 10(c).

Figure 11 and Fig. 12 show the first three trees of this family. Figure 11(a) and (b) show T_1 before and after compatible refinement, respectively. Splits performed during compatible refinement are depicted with dashed lines. T_1 has 19 internal nodes before refinement, and 97 internal nodes after refinement. Fig. 11(c) shows T_1 and T_2 such that T_1 is within the gray inner square with thick borders. T_2 is defined within the outer square. The thicker dashed lines correspond to the additional splits (i.e., internal nodes generated) due to compatible refinement of T_2 .

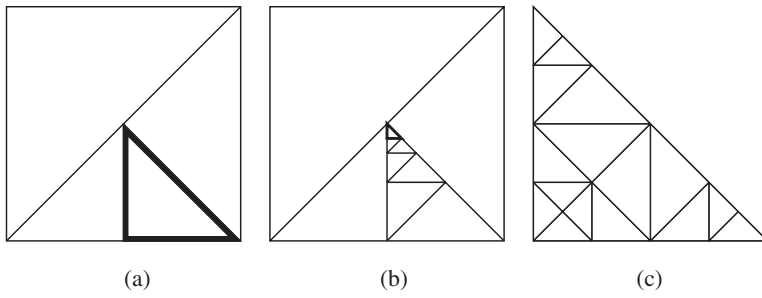


Fig. 10. Construction of T_i

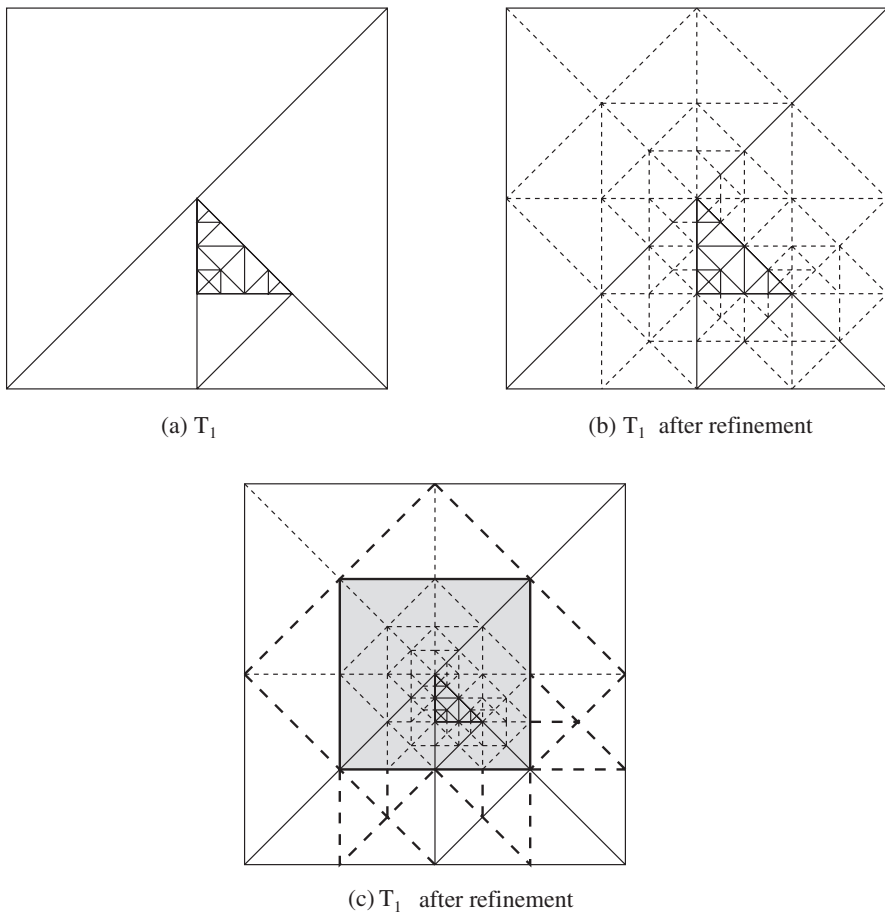


Fig. 11. T_1 within the inner gray square and T_2 within the outer square

Figure 12 shows T_1 , T_2 and T_3 such that T_1 is within the innermost light gray square, T_2 is within the dark gray next outer square and T_3 is defined within the outermost square.

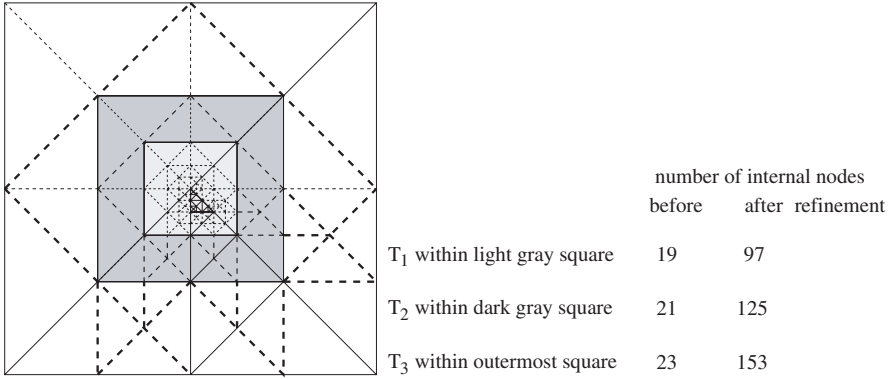


Fig. 12. T_3

From these first three trees of the sequence we observe a pattern that, T_{i+1} contains two more internal nodes than T_i , and the compatible refinement of T_{i+1} produces twenty six more internal nodes than the compatible refinement of T_i . See Fig. 13 for a depiction of how T_{i+1} is related to T_i . In this figure, the thick solid lines represent the two additional splits in T_{i+1} compared to T_i , and the thick dashed lines (seven of them are on the border of T_i) constitute the twenty six additional splits needed for the compatible refinement of T_{i+1} than were necessary for the compatible refinement of T_i . (The thin solid lines were already accounted for in previous trees of the sequence.) The number of internal nodes before and after the compatible refinement for the first three trees of the family is also given in Fig. 12.

Based on above observations, any tree of this family with n nodes generates a tree with $14n - 169$ nodes after compatible refinement. As n increases, the expansion factor approaches the upper bound of 14.

5 The Expansion Factor in Higher Dimensions

Unlike the 2-dimensional case, we do not have tight bounds on the expansion factor for dimensions 3 and higher. However, we will sketch an upper bound on their size after compatible refinement. In our results for 2-dimensional trees described in previous sections, we have chosen the minimum barrier for a simplex in order to prove a tight upper bound. Our approach will be to generate a larger barrier, but one that is easier to analyze. Such a barrier for a *level-0* simplex in the 2-dimensional case is shown in Fig. 14. This barrier contains 18 simplices all of the same depth. (All of them are *level-0* simplices.)

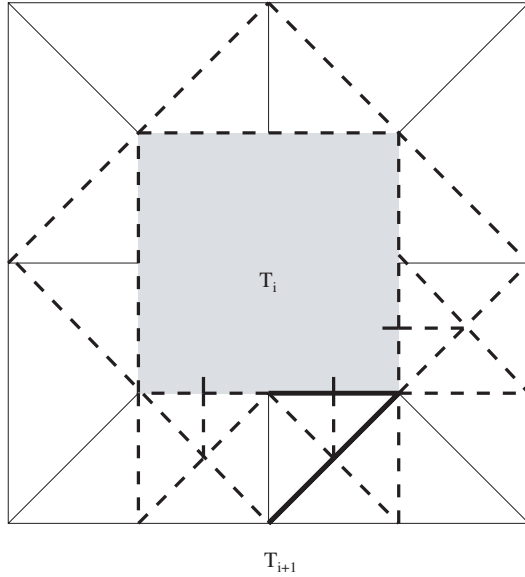


Fig. 13. T_i and T_{i+1}

Consequently, a *level-0* simplex could be in the barriers of 17 other simplices, meaning that if it were split, it could be responsible for 17 other element splits. Thus, compatibly refining a 2-dimensional simplex decomposition tree could increase the number of nodes by at most a factor of 18.

If we analyze the construction of the barrier, we note that the square containing S in Fig. 14 is surrounded by 8 squares and each square contains 2 *level-0* simplices. We can generalize such a barrier to d -dimensional case as follows. Consider a *level-0* simplex S within a d -dimensional hypercube H . Surround H by $3^d - 1$ hypercubes such that each face of H is shared by a neighbor hypercube. Each of these neighbor hypercubes contains $d!$ simplices. This results in a barrier containing $3^d d!$ *level-0* simplices including S . Consequently, a *level-0* simplex could be in the barriers of $(3^d d!) - 1$ simplices, therefore, if it is split, it could be responsible for $(3^d d!) - 1$ other element splits. Thus, a d -dimensional simplex decomposition tree could grow by at most a factor of $3^d d!$ when compatibly refined.

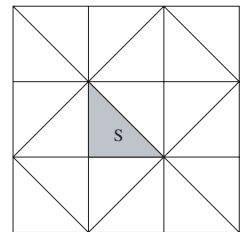


Fig. 14. A naive barrier for a *level-0* simplex in 2-dimensions

However, above analysis is only for *level-0* simplices. In the d -dimensional case, we have d canonical simplices to be considered. We can construct the barrier for a *level- k* simplex S as follows. S is contained in a d -dimensional hypercube H . Surround H by $3^d - 1$ hypercubes as before, but consider that

each hypercube is subdivided into $d! \cdot 2^k$ level- k simplices instead. This results in a barrier containing $3^d d! \cdot 2^k$ level- k simplices including S . Consequently, a level- k simplex could be in the barriers of $3^d d! \cdot 2^k - 1$ simplices. Since k could be $d - 1$ at most, an d -dimensional simplex decomposition tree could grow by at most a factor of $3^d d! \cdot 2^{d-1}$ when compatibly refined.

6 Conclusion

We have shown that when compatibly refined the size of a 2-dimensional simplex decomposition tree grows at most by a factor of 14 and this is tight. This is a worst-case bound, however, and our preliminary experiments on randomly generated sd-trees suggest that, in practice the expansion factor is much smaller. For example, over a 100 randomly generated 2-dimensional sd-trees of maximum height 32, the average expansion factor was found to be only 4.7, and the maximum expansion factor was found to be 5.9. For 3-dimensional sd-trees of maximum height 32, the average was 31.2 and the maximum was 36.1. For 4-dimensional sd-trees of maximum height 32, the average was 227.6 and the maximum was 244.6.

For dimensions higher than 2, we have sketched an upper bound, but a more complete analysis would be needed to prove tight bounds. Since a d -dimensional sd-tree contains simplices from d different similarity classes, and for a complete analysis each canonical simplex has to be considered separately, it would therefore be much more challenging to prove tight upper bounds for general dimensions.

References

1. E. Allgower and K. Georg. Generation of triangulations by reflection. *Utilitas Mathematica*, 16:123–129, 1979.
2. F. B. Atalay and D. M. Mount. Pointerless implementation of hierarchical simplicial meshes and efficient neighbor finding in arbitrary dimensions. In *Proc. International Meshing Roundtable (IMR 2004)*, pages 15–26, 2004.
3. M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proc. IEEE Visualization'97*, pages 81–88, 1997.
4. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.
5. W. Evans, D. Kirkpatrick, and G. Townsend. Right-triangulated irregular networks. *Algorithmica.*, 30(2):264–286, 2001.
6. T. Gerstner. Multiresolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.
7. T. Gerstner and M. Rumpf. Multiresolutional parallel isosurface extraction based on tetrahedral bisection. In *Proc. Symp. Volume Visualization*, 1999.
8. B. Von Herzen and A. Barr. Accurate triangulations of deformed intersecting surfaces. *Computer Graphics*, 21(4):103–110, 1987.

9. P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proc. of SIG-GRAPH 96*, pages 109–118, 1996.
10. J. M. Maubach. Local bisection refinement for N -simplicial grids generated by reflection. *SIAM J. Sci. Stat. Comput.*, 16:210–227, 1995.
11. D. Moore. The cost of balancing generalized quadtrees. In *Proc. ACM Solid Modeling*, 1995.
12. R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *Proc. IEEE Visualization'98*, pages 19–26, 1998.
13. A. Weiser. *Local-Mesh, Local-Order, Adaptive Finite Element Methods with a Posteriori Error Estimators for Elliptic Partial Differential Equations*. PhD thesis, Yale University, 1981.
14. Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In *Proc. IEEE Visualization'97*, pages 135–142, 1997.

Applications

Patient-Specific Vascular NURBS Modeling for Isogeometric Analysis of Blood Flow*

Yongjie Zhang¹, Yuri Bazilevs¹, Samrat Goswami¹, Chandrajit L. Bajaj², and Thomas J.R. Hughes¹

¹ Institute for Computational Engineering and Sciences, The University of Texas at Austin, United States.

² Department of Computer Sciences and Institute for Computational Engineering and Sciences, The University of Texas at Austin, United States.

Abstract. We describe an approach to construct hexahedral solid NURBS (Non-Uniform Rational B-Splines) meshes for patient-specific vascular geometric models from imaging data for use in isogeometric analysis. First, image processing techniques, such as contrast enhancement, filtering, classification, and segmentation, are used to improve the quality of the input imaging data. Then, luminal surfaces are extracted by isocontouring the preprocessed data, followed by the extraction of vascular skeleton via Voronoi and Delaunay diagrams. Next, the skeleton-based sweeping method is used to construct hexahedral control meshes. Templates are designed for various branching configurations to decompose the geometry into mapped meshable patches. Each patch is then meshed using one-to-one sweeping techniques, and boundary vertices are projected to the luminal surface. Finally, hexahedral solid NURBS are constructed and used in isogeometric analysis of blood flow. Piecewise linear hexahedral meshes can also be obtained using this approach. Examples of patient-specific arterial models are presented.

Keywords: Patient-specific vascular models, hexahedral mesh, skeleton-based sweeping, NURBS, isogeometric analysis, blood flow.

1 Introduction

Recently, patient-specific modeling was proposed as a new paradigm in simulation-based medical planning. Physicians, using computational tools, construct and evaluate combined anatomical/physiological models to predict the outcome of alternative treatment plans for an individual patient. A comprehensive framework has been developed to enable the conduct of computational vascular research [1, 2]. Blood flow simulations provide physicians with physical data to help them devise treatment plans.

*<http://www.ices.utexas.edu/~jessica/paper/vascularmodel>

Email addresses: jessica@ices.utexas.edu (Yongjie Zhang), bazily@ices.utexas.edu (Yuri Bazilevs), samrat@ices.utexas.edu (Samrat Goswami), bajaj@cs.utexas.edu (Chandrajit L. Bajaj), hughes@ices.utexas.edu (Thomas J.R. Hughes)

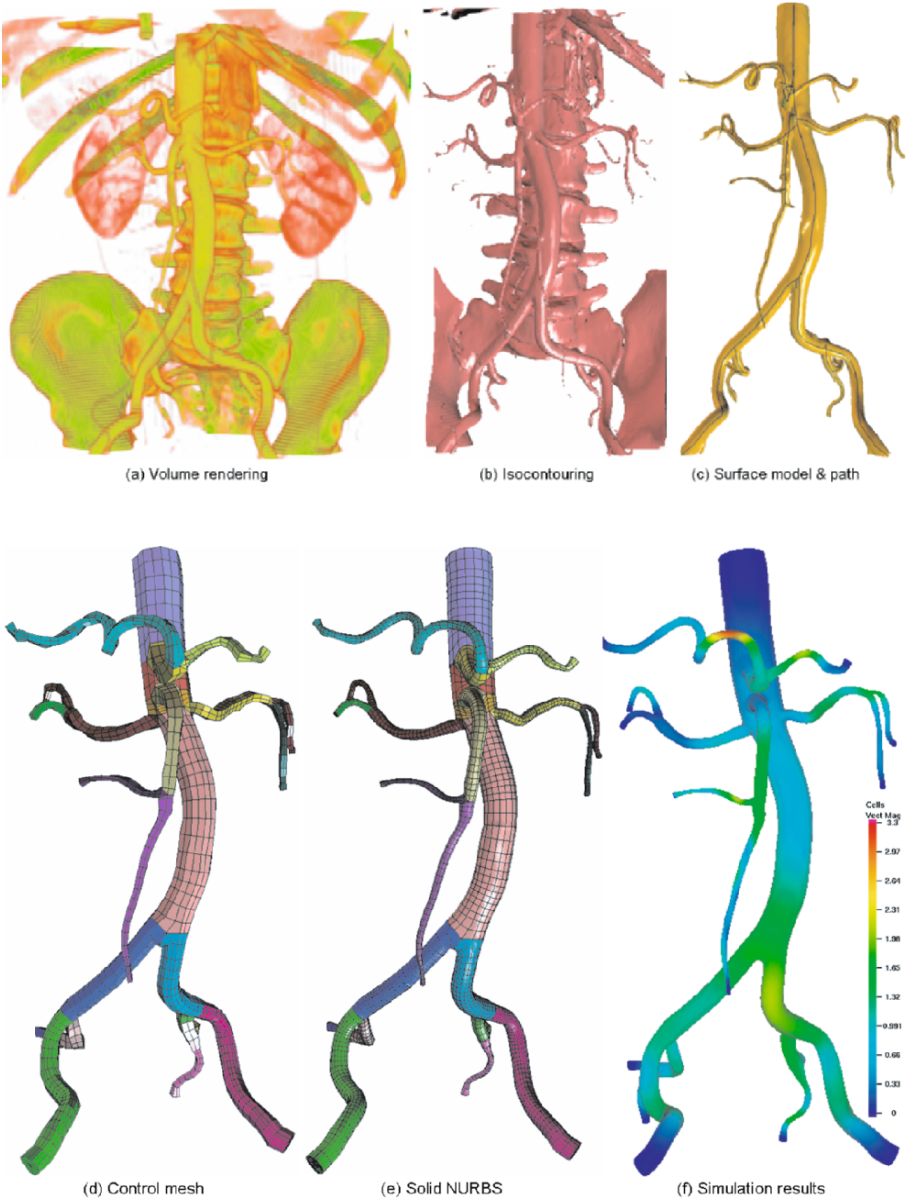


Fig. 1. The abdominal aorta model is divided into 26 patches, and each color represents one different patch. (a) - volume rendering result; (b) - isocontouring result; (c) - surface model and its path after removing unnecessary components; (d) - control mesh; (e) - solid NURBS mesh after refinement (73,314 elements); (f) - fluid-structure interaction simulation results: contours of the arterial wall velocity (cm/s) during late systole plotted on the current configuration. Only major branches are kept in (d-f).

Isogeometric analysis is a new computational technique that improves on and generalizes the standard finite element method. It was first introduced in [3], and expanded on in [4]. In an effort to instantiate the concept of isogeometric analysis, an analysis framework based on NURBS was built. Mathematical theory of this NURBS-based approach was put forth in [5]. NURBS is not the only possible basis for isogeometric analysis but it is certainly the most highly developed and widely utilized. For an introductory text on NURBS, see Rogers [6]. A more advanced treatment of the subject is given in Piegl and Tiller [7]. Other geometric modeling techniques that have potential as a basis for isogeometric analysis include: A-patches [8], T-splines [9], and subdivision [10]. These warrant further investigation.

Figure 1 shows one such model, obtained from patient-specific imaging data. We have designed a set of procedures which allows us to create solid NURBS vascular models directly from patient-specific data. We have named this process the vascular modeling pipeline, which can be divided into four main steps:

1. Preprocessing – in scanned Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) data, the intensity contrast may not be clear enough, noise exists, and sometimes the blood vessel boundary is blurred. Therefore, we use image processing techniques to improve the quality of CT/MRI data, such as contrast enhancement, filtering, classification, and segmentation.
2. Path Extraction – The goal is to find arterial pathes. Vascular surface models can be constructed from the preprocessed imaging data via isocontouring. The skeleton is then extracted from the surface model using Voronoi and Delaunay diagrams. This skeletonization scheme is suitable for noisy input and creates one-dimensional clean skeletons for blood vessels.
3. Control Mesh Construction – a skeleton-based sweeping method is developed to construct hexahedral NURBS control meshes by sweeping a templated quad mesh of a circle along the arterial path. Templates for various branching configurations are presented which decompose the geometry into mapped meshable patches using the extracted skeleton. Each patch can be meshed using one-to-one sweeping techniques. Some nodes in the control mesh lie on the surface, and some do not. We project nodes lying on the surface to the vascular surface. The blood vessel wall can be built by radially extending the surface outside 10%-15% of the distance to the center line.
4. NURBS Construction and Isogeometric Analysis – after generating hexahedral control meshes, we construct solid NURBS geometric models and employ isogeometric analysis to simulate blood flow. Piecewise linear hex meshes can also be obtained. Three numerical examples, coronary, thoracic and abdominal arteries, are presented.

The remainder of this paper is organized as follows: Section 2 reviews related previous work. Section 3 describes the meshing pipeline and preprocessing for our geometric modeling approach. Section 4 talks about solid NURBS construction and isogeometric analysis. Section 5 explains the skeleton-based sweeping method and decomposition templates for various branching configurations. Section 6 presents three numerical examples. Section 7 draws conclusions and outlines planned future work.

2 Previous Work

Sweeping Method: Sweeping, or $2\frac{1}{2}$ -D meshing, is one of the most robust techniques to generate semi-structured hexahedral meshes. One-to-one sweeping requires that the *source* and *target* surfaces have similar topology. The source surface is meshed with quadrilaterals [11], which are swept through the volume using *linking* surfaces as a guide [12].

However, few geometries satisfy the topological constraints required by one-to-one sweeping. In the CUBIT project [13] at Sandia National Labs, a lot of research has been done to automatically recognize features and decompose geometry into mapped meshable areas or volumes. Various many-to-one and many-to-many sweeping methods have been developed [14, 15, 16, 17]. Care should also be taken in locating internal nodes during the sweeping process [18, 19].

Medial Axis-based Mesh Generation: Medial axis is the locus of points that are minimally equidistant from at least two points on the geometry's boundary. The medial axis transform provides an alternative representation of geometric models that has many useful properties for analysis modeling [20]. Applications include decomposition of general solids into subregions for mapped meshing, identification of slender regions for dimension reduction and recognition of small features for suppression. The medial surface subdivision technique [21] decomposes the volume into map-meshable regions, which are then filled with hex elements using templates.

Medial axis has been used to construct hexahedral meshes for CAD objects. The skeleton-based modeling methods were developed for solid models [22]. Quadros et al. used a skeleton technique to control finite element mesh size [23]. Besides other unstructured mesh generation methods [24, 25, 26], a skeleton-based subdivision method has also been used in biomedical applications, such as a below-knee residual limb and external prosthetic socket [27], and bifurcation geometry in vascular flow simulation [28]. However, trifurcations and more complex branchings also exist in the human artery tree. Therefore, decomposition templates for arbitrary branching configurations are desirable and are constructed in this paper.

NURBS in Mesh Generation and Analysis: As the most highly developed and widely utilized technique, NURBS [6, 7, 29] has evolved into an essential tool for a semi-analytical representation of geometric entities. Sometimes NURBS solid models are taken as input for finite element mesh generation [30]. Anderson et al. proposed a fast generation of NURBS surfaces from polygonal mesh models of human anatomy [31]. An enhanced algorithm was developed for NURBS evaluation and utilization in grid generation [32]. In isogeometric analysis [3], NURBS basis functions are used to construct the exact geometry, as well as the corresponding solution space.

3 Meshing Pipeline and Preprocessing

The input images are often of poor quality which makes it difficult to generate quality meshes for regions of interest. To circumvent this problem we pass the raw imaging data through a preprocessing pipeline where the image quality is improved by enhancing the contrast, filtering noise, classifying, and segmenting regions of various materials. The surface model is then extracted from the processed imaging data, and

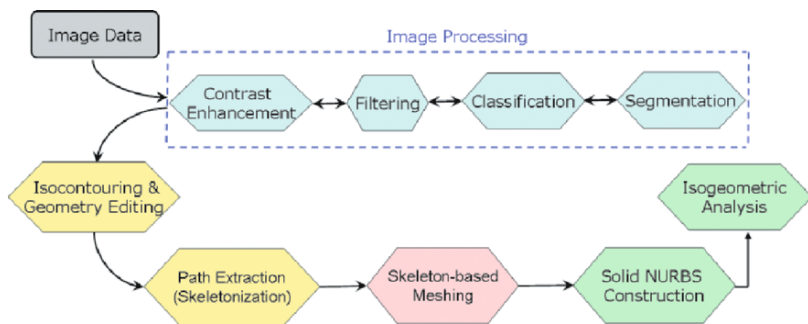


Fig. 2. A schematic diagram of the meshing pipeline. Preprocessing includes three modules: image processing, isocontouring and geometry editing, and path extraction.

the vessel path is obtained after skeletonizing the volume bounded by the surface. First we modify the geometry by removing unnecessary components, then extract the skeleton. The generated path can also be edited according to simulations, e.g., adding a path for a Left Ventricle Assist Device (LVAD) in the thoracic aorta model (Figure 12). A skeleton-based sweeping method is then used to generate hexahedral control meshes for solid NURBS construction and isogeometric analysis. Figure 2 shows the meshing pipeline. The preprocessing step of our skeleton-based meshing approach is described below, including image processing, isocontouring and geometry editing, and path extraction.

Image Processing: We choose a fast localized method for image contrast enhancement [33]. The basic idea is to design an adaptive transfer function for each individual voxel based on the intensities in a suitable local neighborhood. A bilateral pre-filtering coupled with an evolution driven anisotropic geometric diffusion PDE (partial differential equation) [34] is utilized to remove noise. Sometimes we need to classify the voxels into several groups, each of which corresponds to a different type of material. We choose an approach which relies on identification of the contours by membership of seed points which are located by the gradient vector diffusion [35]. A variant of the fast marching method is adopted [36] to segment the imaging data to find the clear boundary of each voxel group belonging to a certain category.

Isocontouring and Geometry Editing: There are two main isocontouring methods from imaging data: Primal Contouring (or Marching Cubes [37]) and Dual Contouring [38]. In this application we choose Dual Contouring to extract the isosurface, because it tends to generate meshes with better aspect ratios. We then modify the model to suit our particular application. This can be done in various ways, for example, by removing unnecessary components, adding necessary components which are not constructed from imaging data, denoising the surface, etc. After getting the vessel path, we can edit it according to simulation requirements. For example, we can add a path for the left ventricle assist device (LVAD) in the thoracic aorta model (Figure 12).

Path Extraction: The vertex set of the extracted and possibly repaired geometry is then used to create an interior path lying in the middle of the blood vessels. We define a squared distance function which assigns to any point $x \in \mathbb{R}^3$, the minimum

square distance to the vertex set. We further compute the index 1 and index 2 saddle points of this distance function and compute the unstable manifold of these two types of critical points. The identification of the critical points along with their indices and the computation of the unstable manifold are done efficiently via the Voronoi and its dual Delaunay diagram of the point set. The details of this method can be found in [39]. We adopt this method of path generation because it has several advantages which are useful for the patient specific modeling of blood vessels. One advantage is that it can handle noisy input gracefully. Often the noise present in the data is not fully eliminated after the preprocessing stage. In the path generation step we employ another stage of filtering which helps to construct a clean skeletal path for the extracted geometry. Secondly, the extracted geometry may have flat regions where it is not straight forward to obtain a linear path. Fortunately our starring scheme, as described in [39], eliminates these spurious features and create the one-dimensional path. The results of this path generation step are shown on various datasets (Figures 1, 11, 12).

4 Solid NURBS Construction and Isogeometric Analysis

In a NURBS-based isogeometric analysis a physical domain in \mathbb{R}^3 is defined as a union of patches. A patch, denoted by Ω , is an image under a NURBS mapping of a parametric domain $(0, 1)^3$

$$\Omega = \{\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \mid \mathbf{x} = \mathbf{F}(\xi, \eta, \zeta), 0 < \xi, \eta, \zeta < 1\}, \quad (1)$$

where

$$\mathbf{F}(\xi, \eta, \zeta) = \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^l R_{i,j,k}^{p,q,r}(\xi, \eta, \zeta) \mathbf{C}_{i,j,k}, \quad (2)$$

$$R_{i,j,k}^{p,q,r} = \frac{N_{i,p}(\xi) M_{j,q}(\eta) L_{k,r}(\zeta) w_{i,j,k}}{\sum_{\hat{i}=1}^n \sum_{\hat{j}=1}^m \sum_{\hat{k}=1}^l N_{\hat{i},p}(\xi) M_{\hat{j},q}(\eta) L_{\hat{k},r}(\zeta) w_{\hat{i},\hat{j},\hat{k}}}. \quad (3)$$

In the above, $R_{i,j,k}^{p,q,r}(\xi, \eta, \zeta)$'s are the rational basis functions, and $\mathbf{C}_{i,j,k}$'s $\in \mathbb{R}^3$ are the control points. In the definition of the rational basis, $N_{i,p}(\xi)$'s, $M_{j,q}(\eta)$'s, and $L_{k,r}(\zeta)$'s, are the univariate B-spline basis functions of polynomial degree p , q , and r ; $w_{i,j,k}$'s, strictly positive, are the weights.

In isogeometric analysis the geometry generation step involves construction of a control mesh, which is a piecewise multi-linear interpolation of control points, and the corresponding rational basis functions. The initial mesh encapsulates the 'exact geometry' and, in fact, defines it parametrically.

For the purposes of analysis, the isoparametric concept is invoked (see Hughes [40]). The basis for the solution space in the physical domain is defined through a push forward of the rational basis functions defined in (2) (see [5] for details). Coefficients of the basis functions, defining the solution fields in question (e.g., displacement, velocity, etc.), are called control variables.

As a consequence of the parametric definition of the 'exact' geometry at the coarsest level of discretization, mesh refinement can be performed automatically

without further communication with the original description. This is an enormous benefit. There are NURBS analogues of finite element h - and p -refinement, and there is also a variant of p -refinement, which is termed k -refinement, in which the continuity of functions is systematically increased. This seems to have no analogue in traditional finite element analysis but is a feature shared by some meshless methods. For the details of the refinement algorithms see [3].

The isogeometric approach is fundamentally higher-order. For example, in order to represent circles, cylinders and spheres, rational polynomials of at least quadratic order are necessary. The generation of refined NURBS bases of all orders is facilitated by simple recursion relationships. The versatility and power of recursive NURBS basis representations are truly remarkable. Equation systems generated by NURBS tend to be more homogeneous than those generated by higher-order finite elements and this may have some benefit in equation solving strategies. NURBS satisfy a ‘variation diminishing’ property. For example, they give monotone fits to discontinuous control data and become smoother as order is increased, unlike Lagrange interpolation polynomials which oscillate more violently as order is increased. NURBS of all orders are non-negative pointwise. This means that every entry of the NURBS mass matrix is non-negative. These properties are not attained in finite element analysis. On the other hand, NURBS are not interpolatory. They are fit to nets of control points and control variables. This aspect is less transparent to deal with than the corresponding finite element concepts of interpolated nodal points and nodal variables but somewhat similar to the situation for meshless methods. There are many robust algorithms to create very complex geometries with NURBS.

5 The Skeleton-based Sweeping Method

Blood vessels are tubular objects, therefore we choose the sweeping method to construct hexahedral control meshes for NURBS-based isogeometric analysis.

5.1 Sweeping Method

In the sweeping method, a templated quadrilateral mesh of a circle is projected onto each cross-section of the tube, then corresponding vertices in adjacent cross-sections are connected to form a hexahedral mesh. A hexahedral NURBS control mesh should satisfy the following four requirements:

1. Any two cross-sections can not intersect with each other.
2. Each cross-section should be perpendicular to the path line.
3. In the intersection region of several branches, each cross-section should remain perpendicular to the vessel surface.
4. In order to achieve a G^1 -continuous surface, the boundary vertex shared by two patches in the control mesh should be collinear with its two neighbors along the axial direction, and the boundary vertex shared by three or more patches should be coplanar with all of its neighboring boundary vertices. This is because, for a so-called open knot vector, a NURBS curve is tangent to the control polygon at the first and the last control nodes.

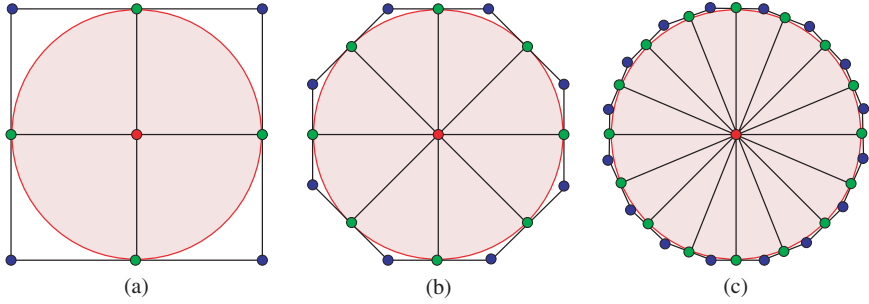


Fig. 3. Multi-resolution templates for cross-sections. (a) Level-1-template (9 control nodes); (b) Level-2-template (17 control nodes); (c) Level-3-template (25 control nodes). Red points are circle centers, green points are interpolatory control nodes on the circle, and blue points are non-interpolatory control nodes defined as the intersection point of two tangent lines at its two neighboring green points.

We choose to parameterize the template cross-section as follows. One parametric direction is associated with a closed circular curve, while another parametric direction is associated with a radial coordinate. Rational quadratic basis is used to define the circular curve with a control polygon given by the linear interpolation of the green and blue points shown in Figure 3. For the template shown in Figure 3a, the control polygon is a square consisting of 8 control nodes, while in Figure 3b, it is an octagon. Note that the circular cross-section is unchanged geometrically and parametrically as more control points are chosen for its representation. The green control points lie on the circle, while the blue control points do not. This is due to the fact that the rational basis is interpolatory at the green points and is not interpolatory at the blue points. Also note that each interpolatory control point has two neighboring non-interpolatory points that are collinear with it. This construction guarantees the resultant circular curve to be G^1 -continuous. Later, when we discuss data fitting, it is only the interpolatory points that get projected onto the true surface. The non-interpolatory points are adjusted to preserve the collinearity in order to obtain a G^1 -continuous cross-section.

In the process of sweeping, we translate the cross-section template to the selected locations on the path, and rotate it to make its normal vector pointing in the direction tangent to the path as shown in Figure 4. This gives the third parametric direction for the solid NURBS representation. The hexahedral control mesh is constructed by connecting the corresponding control nodes in adjacent cross-sections. Piecewise linear hexahedral meshes can also be generated at the same time by projecting all boundary vertices to the vessel surface, or by interpolating the elements of the solid NURBS geometry.

5.2 Branching Templates

One-to-one sweeping requires that the source and target surfaces have similar topology. Generally, arterial models do not satisfy this requirement, therefore we need to decompose arterial networks into mapped meshable regions. In this section, we will

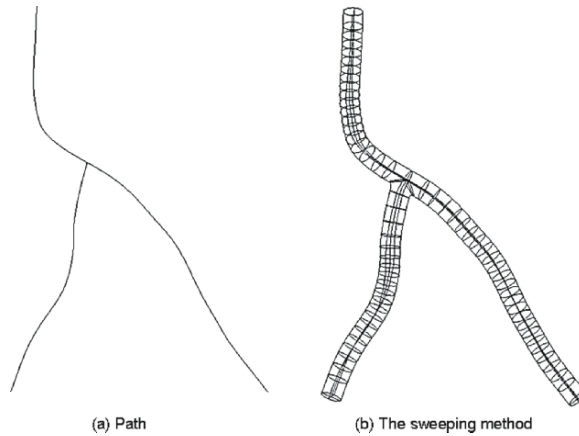


Fig. 4. The skeleton-based sweeping method. (a) - a blood vessel skeleton; (b) - a templated circle is translated and rotated to each cross-section. A bifurcation is shown.

discuss various decomposition templates for different branching configurations. An n -**branching** is formed when n branches join together, where $n \geq 3$. When $n = 3$, it is a **bifurcation**; when $n = 4$, it is a **trifurcation**; when $n > 4$, we call this situation **higher order branching**.

In the human vascular system, most branchings are bifurcations. However, trifurcations or higher order branchings also exist. For example, there are several trifurcations in the coronary arteries (Figure 11) and the abdominal aorta (Figure 1). In the following, we will discuss decomposition templates for all possible branching configurations.

Bifurcation

For every intersection, a so-called master arterial branch is chosen. Typically, it is an artery with the largest diameter. Suppose the master branch consists of two sub-branches (Branch 1 and Branch 2), and the slave branch is Branch 3, as shown in Figure 5a. The axes of Branch 1, 2 and 3 are Axis 1, 2 and 3 respectively (Axis 1 and Axis 2 may not be collinear). There is one basic case, shown in Figure 5, and all bifurcations can be decomposed into three map-meshable regions by a variant of this basic template.

Figure 5 shows the path, the constructed hexahedral control mesh, the solid NURBS mesh, and the piecewise linear hexahedral meshes of the bifurcation template. The bifurcation geometry is decomposed into three patches: the master branch contains two patches (red and green), and the slave branch has one patch (yellow). Here we choose Level-1-template (Figure 3a) for each cross-section, as the master and slave branches have similar diameters. The bifurcation template also works for finer cross-sections.

When the master branch and the slave branch have different diameters, the control nodes of some cross-sections are distributed unevenly in order to generate better intersection regions. Figure 6 shows two control meshes and their corresponding

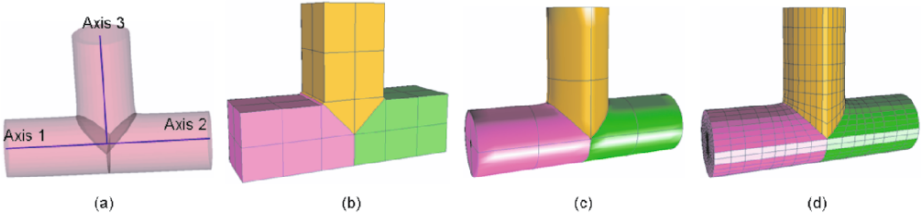


Fig. 5. The bifurcation decomposition template. (a) - path; (b) - control mesh; (c) - solid NURBS; (d) - a piecewise linear hex mesh. The bifurcation geometry is decomposed into 3 patches, and each patch is rendered with a different color.

solid NURBS meshes. The master branch control polygon is deformed from a square to a trapezoid so as to accommodate a slave branch with a smaller diameter. Note that the NURBS basis changes accordingly so as to preserve the circular cross-section, and the quality of the intersection geometry is improved as can be seen in Figure 6 and Figure 7, where the axes of the master and the slave branches are non-orthogonal, or non-coplanar. Although deforming the control polygon of the master branch gives better results as compared to the non-deformed case, for the intersection of branches with high diameter ratios we advocate the use of a finer template for the master branch, such as a Level-2-template or a Level-3-template.

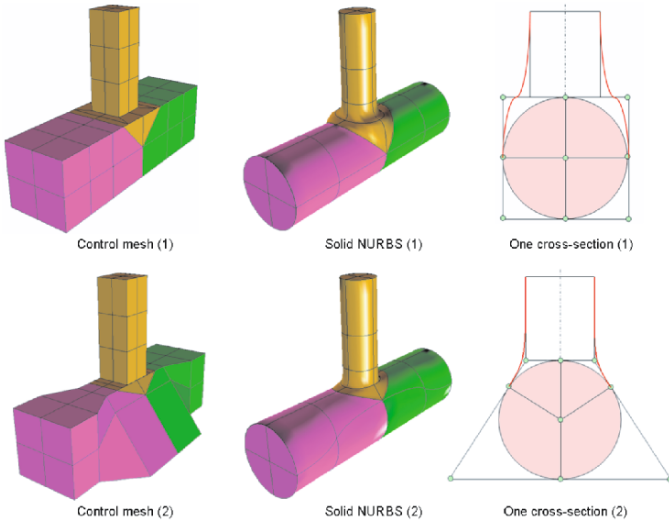


Fig. 6. Comparison of two meshes for the situation when the master branch and the slave branch have different diameters. Control nodes on cross-sections are distributed evenly in Mesh (1) (the top row), and unevenly in Mesh (2) (the bottom row). The red curves in the right two pictures are transition curves.

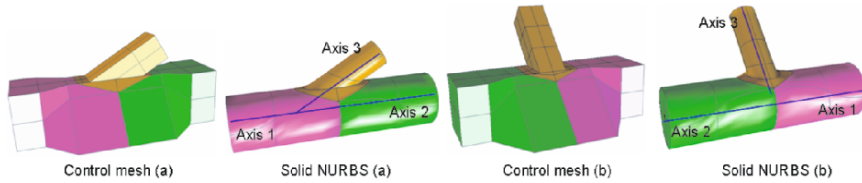


Fig. 7. Control mesh and solid NURBS (a) - the axes of the master and slave branches are not perpendicular to each other; Control mesh and solid NURBS (b) - the axes of the master and slave branches are not coplanar.

Trifurcation

Trifurcation has one master branch and two slave branches. According to the position of slave branches relative to the master branch, we classify all possible trifurcations to fall into five irreducible cases. All other trifurcations can be decomposed into map-meshable regions by extending the five basic decomposition templates.

Case 1: The two slave branches are distributed along the peripheral direction of the master branch, and they are in opposite relative to the master branch (the angle between them is around 180°). The same cross-section template can be used for the master and slave branches.

Case 2: The two slave branches are distributed along the peripheral direction, and the angle between them is arbitrary. Finer cross-section template is chosen for the master branch.

Case 3: The two slave branches are distributed along the axial direction of the master branch, and they intersect with each other.

Case 4: The two slave branches are distributed along the axial direction of the master branch, and they do not intersect with each other. This situation degenerates into two bifurcations.

Case 5: The two slave branches do not intersect with the master branch at the same point, but they intersect with each other. In this situation, two bifurcations merge into one trifurcation.

If a Level-1-template is selected as the cross-section of the master branch, then there are at most two slave branches along the peripheral direction as shown in Figure 8 (Case 1). If the two slave branches are not opposite relative to the master branch, or the two slave branches have different diameters from the master branch, then a Level-1-template is not suitable, and we need to choose finer cross-section templates, such as a Level-2-template or a Level-3-template. Similarly, if a Level-2-template is selected as the cross-section of the master branch, then there can be at most four slave branches along the peripheral direction. A Level-3-template allows at most eight slave branches along the peripheral direction. In Case 2 of Figure 8, the two slave branches are distributed along the peripheral direction and they are not opposite, therefore we choose the finer cross-section template for the master branch (Level-2-template), while the slave branch may have coarser cross-sections (Level-1-template).

Case 3 and Case 4 have the same path, but Case 4 degenerates into two bifurcations because its two slave branches do not intersect with each other even though their

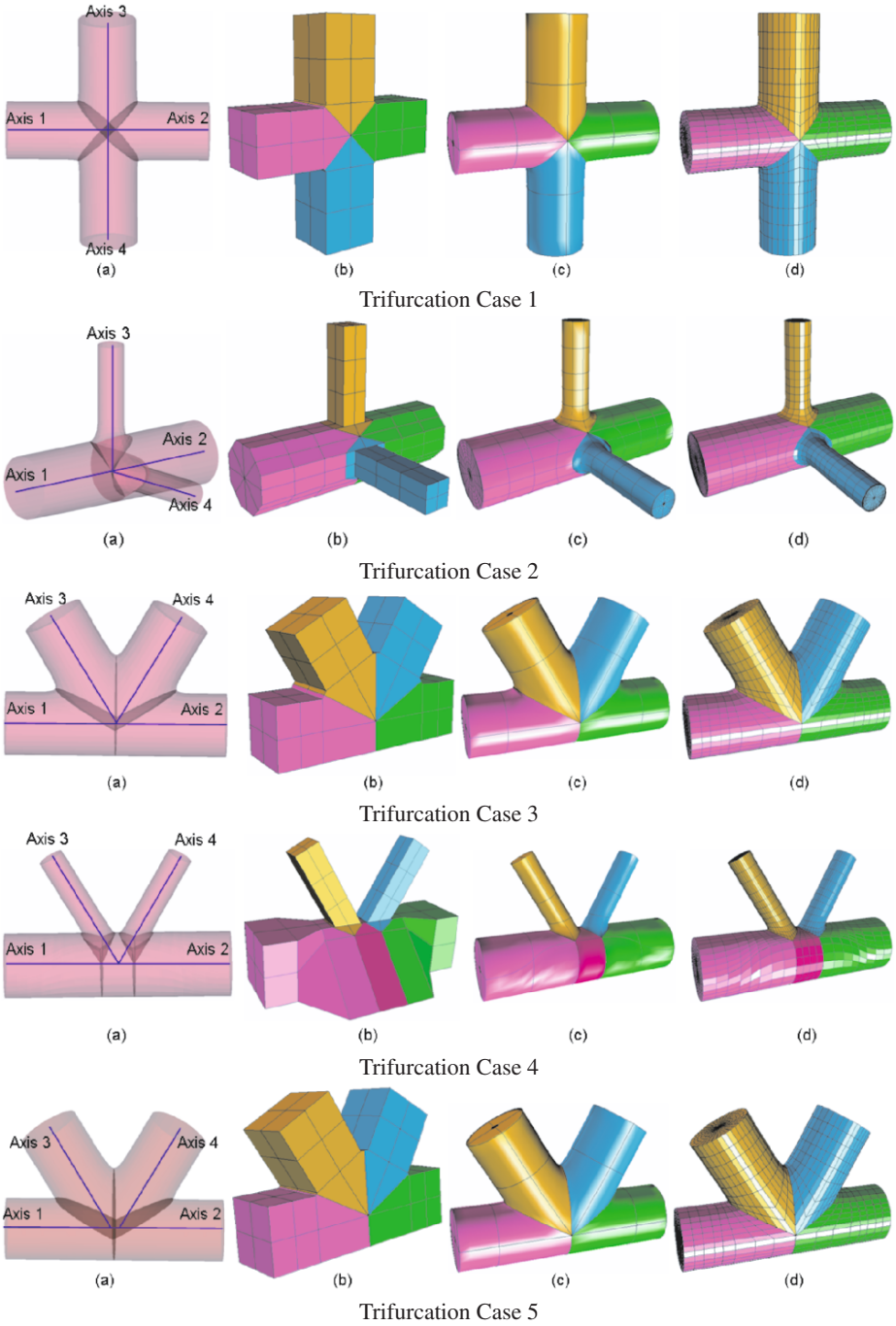


Fig. 8. The trifurcation decomposition templates of Case 1-4. (a) - path; (b) - hex control mesh; (c) - solid NURBS; (d) - piecewise linear hex mesh. The Trifurcation geometry is decomposed into 4 patches (Case 1, 2, 3, 5) or 5 patches (Case 4). Each patch is rendered with a different color.

axes intersect. There is another special situation (Case 5) where two slave branches do not intersect with the master branch at the same intersection point in the skeleton, but the two intersection points are very close and the two slave branches intersect with each other. This situation contains two bifurcations in the skeleton, but it should be considered as one trifurcation. Therefore, when we choose branching configurations, both the path and the vessel size should be considered.

Remark: In n -branching, n should be decided not only by the path, but also by the diameter of each slave branch. In other words, if neighboring slave branches intersect with each other, then it is n -branching. Otherwise, it degenerates into several m -branchings, where $m < n$. On the other hand, several m -branchings may merge into one n -branching if its slave branchings intersect with each other.

Higher Order Branching

Here we discuss three basic templates for n -branching when $n > 4$. Relative to the master branch, there are only two directions to arrange slave branches, the peripheral and axial directions of the master branch. All other n -branching configurations can be obtained by combining the three basic ones.

Case 1: There are three or more slave branches distributed along the peripheral direction of the master branch. Figure 9 shows one example of four slave branches along the peripheral direction. Level-2-template is selected for the master branch. If there are more than four slave branches, the master branch needs to have a finer cross-section. The cross-section template of slave branches can be coarser.

Case 2: There are three or more slave branches distributed along the axial direction of the master branch. Neighboring slave branches intersect with each other.

Case 3: There are three or more slave branches distributed along the axial direction of the master branch. Slave branches do not intersect with each other. n -branching degenerates into several m -branchings ($m < n$).

Several lower order branchings may merge into a higher order one, for example, one bifurcation and one trifurcation can merge into a 5-branching. Case 1, Case 2 and Case 3 can be combined together to form more complex configurations. Figure 9 shows one example of 7-branching. It has four slave branches along the peripheral direction and two slave branches along the axis of the master branch.

5.3 Data Fitting

After the sweeping step, each circular cross-section needs to be projected onto the vessel surface as shown in Figure 10. First the interpolatory control points (green points) are moved in the radial direction to the true surface. Then, the non-interpolatory points (blue points) are placed at the intersection of the lines tangent to the true surface passing through the two neighboring interpolatory points.

There are situations when the tangent lines do not intersect inside the fan region defined in Figure 10b, or do not even intersect with each other when they are parallel. This may occur when the cross-section template is not sufficiently fine to capture features of the true surface, or when the true surface is noisy. This situation will also result in an overlap in the geometry. In order to avoid overlap, we force the

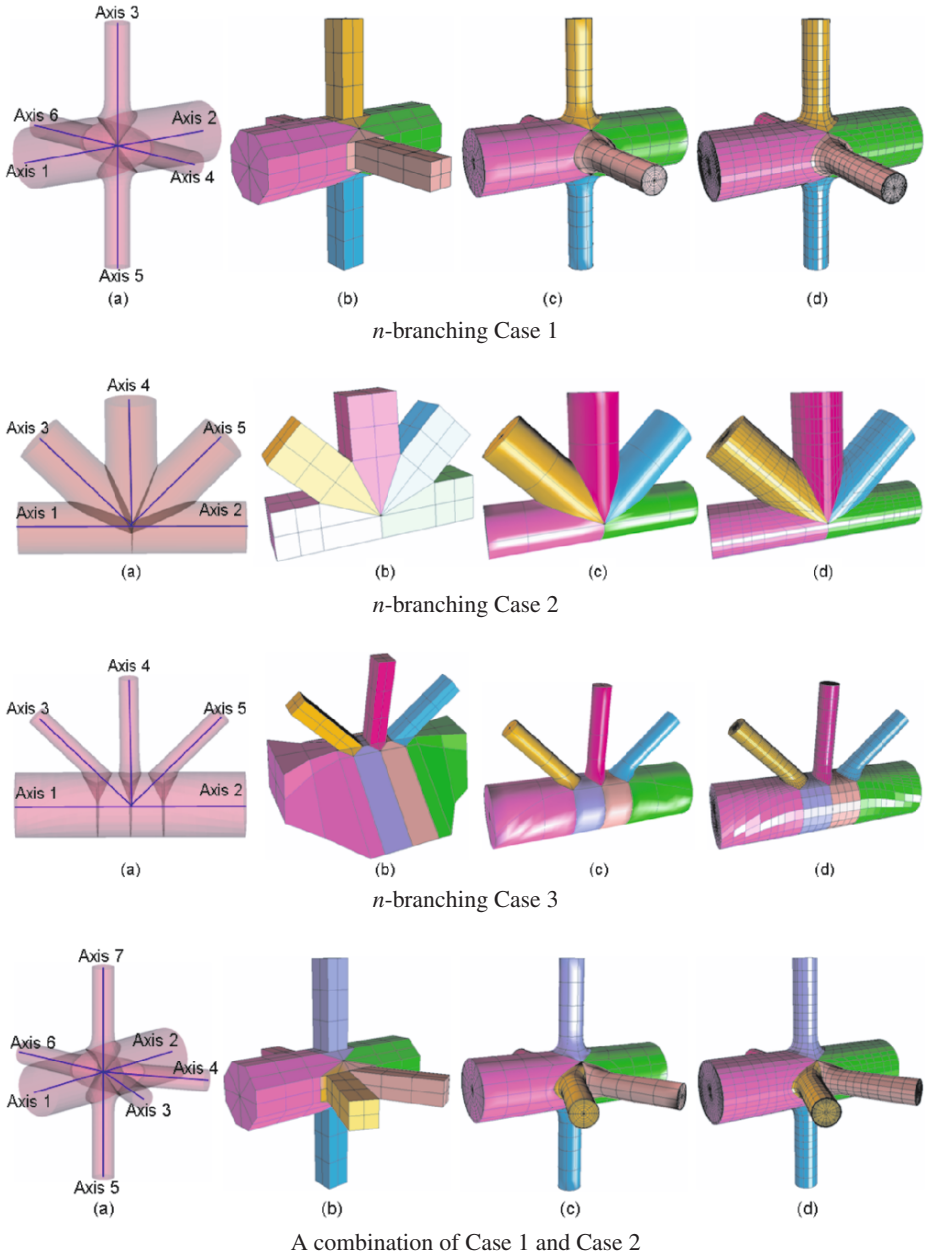


Fig. 9. The n -branching templates of Case 1-3 and a combination of Case 1-2. (a) - path; (b) - control mesh; (c) - solid NURBS; (d) - piecewise linear hex mesh. The Trifurcation geometry is decomposed into 6 patches (Case 1), 5 patches (Case 2) or 7 patches (Case 3, a combination of Case 1 and Case 2). Each patch is rendered with a different color.

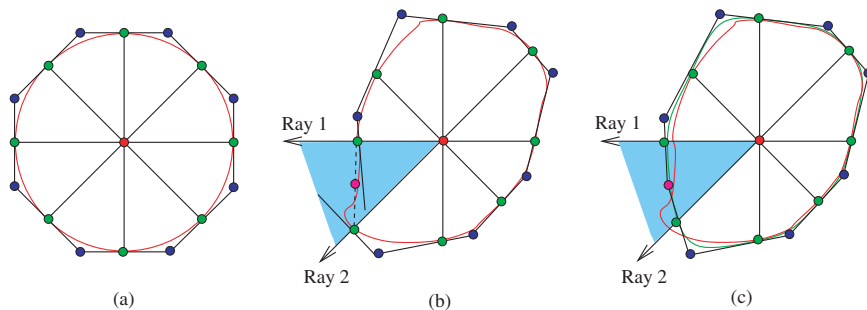


Fig. 10. One cross-section template is projected to the vessel surface. (a) Level-2-template for one circular cross-section; (b) The red curve is the vessel curve. In the blue fan region, the two tangent lines do not intersect with each other, and the magenta point is the calculated control node; (c) The two neighboring green control nodes are adjusted. The green curve is the constructed spline curve. Green control nodes are interpolatory points lying on the vessel surface, and blue points are non-interpolatory.

non-interpolatory point to stay inside the fan region (the sector between two radial rays) by placing it at the midpoint (indicated by the magenta color) of the segment connecting the two interpolatory points. Finally, the location of the interpolatory points is changed so as to preserve G^1 -continuity of the surface.

After projecting each cross-section to the vessel surface, we construct hexahedral control meshes and generate solid NURBS for patient-specific vascular models. The geometric error can be reduced by choosing a finer template for each cross-section.

5.4 Implications for Analysis of Blood Flow in Arteries

The proposed construction of a NURBS solid mesh for isogeometric analysis has the following implications:

1. Parametric definition of the NURBS mesh allows one to refine the boundary layer region near the arterial wall in order to accurately capture flow features.
2. In the case of a flow in a straight circular pipe driven by a constant pressure gradient, NURBS basis of quadratic order gives rise to a point-wise exact solution to the incompressible Navier-Stokes equation system. This also has implications on the overall accuracy of the approach.
3. The choice of the parameterization of the cross-section template gives rise to a singularity in the geometrical mapping at the center. This singularity does not seem to affect the accuracy of the computational results. Other parameterizations of the circular cross-section, containing multiple patches, are also possible.

6 Numerical Examples

In this section we present applications of the meshing pipeline to three patient-specific vascular models: a model of a portion of the coronary tree, a model of the

thoracic aorta, and a model of the abdominal aorta. Isogeometric analysis is then used to compute blood flow in the models. In all cases, time-dependent, viscous, incompressible Navier-Stokes equations were used as the blood model. The fluid density and dynamic viscosity were chosen to be representative of blood flow. The first example makes use of the Casson model for the dynamic viscosity while in other examples viscosity was set to a constant value. All models are subjected to a time-periodic inflow boundary condition, which simulates the input from a beating heart. The arterial wall is assumed rigid in the first example. Examples two and three present fluid-structure interaction calculations in which the wall is assumed to be elastic (see Bazilevs *et al.* [2] for the details of the mathematical formulation). The rigid wall simulation was performed on a single processor, while the elastic wall simulations were done in parallel.

A model of a portion of the coronary tree: Data for this model was obtained from CT Angiography imaging data of a healthy male, over 55 years of age. Large motions of the heart, as it supplies blood to the circulatory system, decreases the quality of the imaging data, and makes construction of patient-specific coronary models a challenging task. Nevertheless, we managed to extract a portion of the coronary tree for the purposes of creating an analysis suitable model. Results of the isocontouring algorithm are shown in Figure 11a. Figures 11b-11d show the path, the control mesh, and the solid NURBS model of the arterial segment. The model was used to study drug delivery processes in arteries. The drug concentration in the blood is modeled as a passive scalar governed by an unsteady advection-diffusion equation. Figure 11e shows the isosurface of the drug concentration at 50% colored by the blood velocity magnitude, revealing that the flow is unsteady, and has many complex features.

Thoracic aorta model: Data for this model was obtained from CT Angiography imaging data of a healthy male over-30 volunteer. A patient-specific model of the thoracic aorta was constructed by running through the meshing pipeline. An extra branch, representing a left ventricular assist device (LVAD), was added to the arterial model. Evaluation of LVADs, as well as other electromechanical devices used to support proper blood circulation, is of great interest to the cardiovascular community. The path, the control mesh, and the solid NURBS model are shown in Figures 12a-12c. Figure 12d shows a result of the fluid-structure interaction simulation. Note that the inlet and the three smaller outlet branches were extended for the purposes of analysis.

Abdominal aorta: Data for this model was obtained from 64-slice CT angiography of a healthy male over 55 years of age. Various stages of the meshing pipeline are illustrated in Figure 1a-1f. Figure 1g shows a result of the fluid-structure simulation. A computational study using a truncated geometrical model of this aorta was performed in [2]. We used 85 seconds for path extraction and 8 seconds for control mesh construction on a 64-bit dual-AMD 2GHz linux system, and 20 seconds for solid NURBS generation on an Intel 3GHz linux system.

7 Conclusions and Future Work

We have developed a four-stage process to construct analysis suitable geometric models from patient-specific vascular imaging data with a goal of using them in

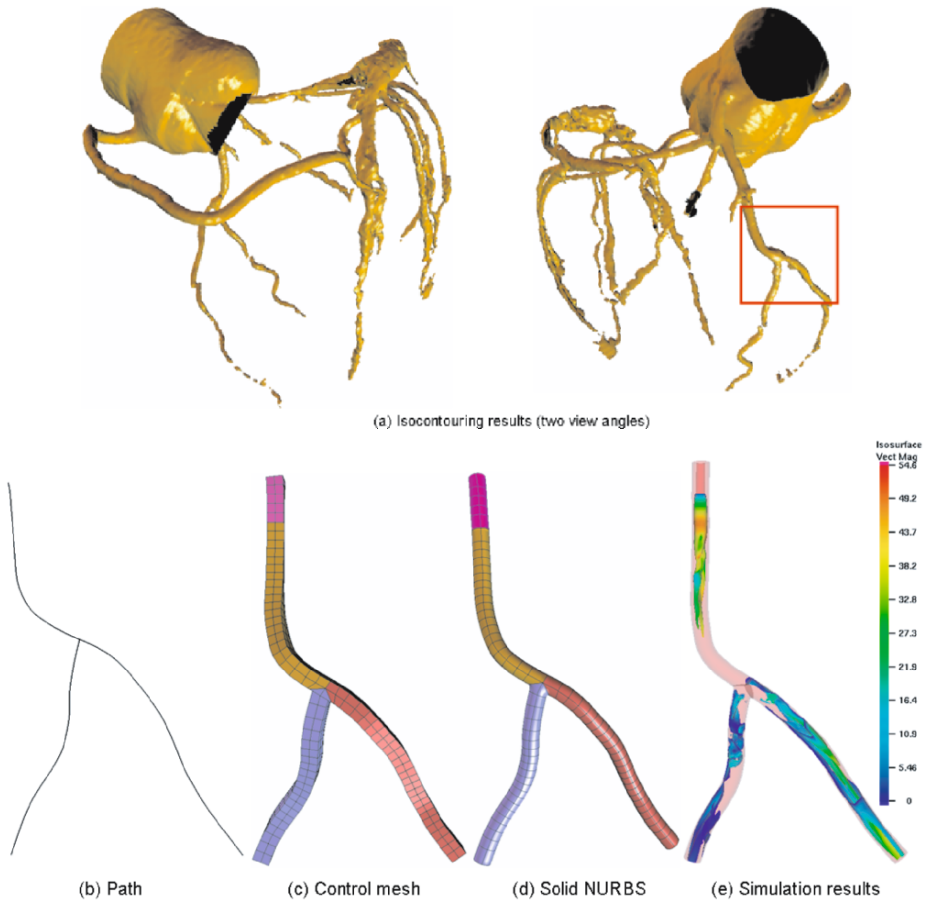


Fig. 11. Coronary artery. (a) - isocontouring results (two different view angles); (b) - path; (c) - control mesh; (d) - solid NURBS model (20,824 elements); (e) - rigid wall simulation results: isosurface of the drug concentration at 50% colored by the blood velocity magnitude (cm/s).

isogeometric analysis of blood flow in arteries. We have focused on the NURBS modeling, and did not treat other geometrical modeling technologies, such as A-patches, T-splines, and subdivision. We would like to investigate these techniques in the future.

We have successfully applied our method to three patient-specific examples, which involve a model of a part of the coronary arterial tree, a thoracic aorta model, and an abdominal aorta model. As part of the future work, we would like to apply the techniques described here to modeling and analysis of the human heart.

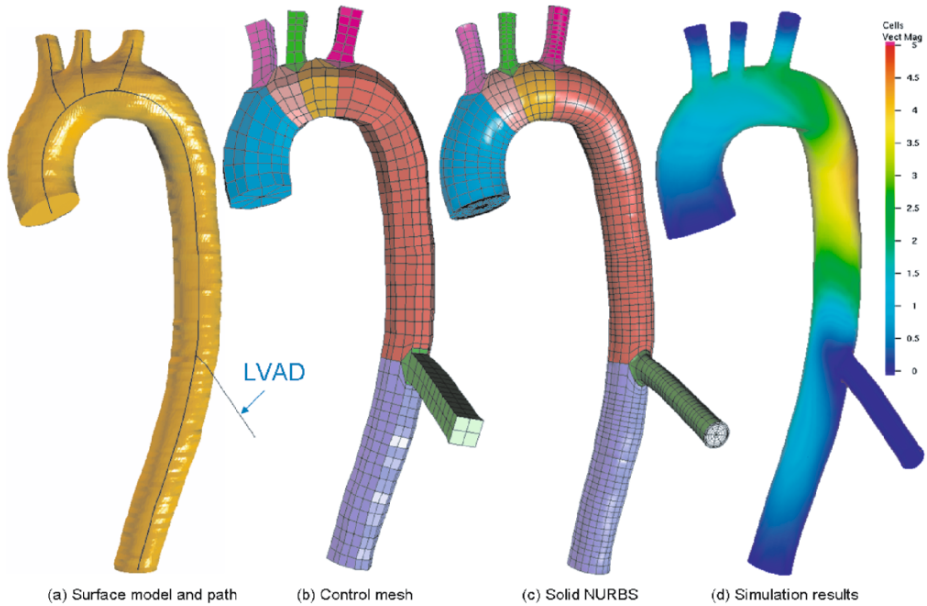


Fig. 12. Thoracic aorta. (a) - surface model and the path, a LVAD is inserted; (b) - control mesh; (c) - solid NURBS (41,526 elements); (d) - fluid-structure interaction simulation results: contours of the arterial wall velocity (cm/s) during late diastole plotted on the current configuration.

Acknowledgement

Y. Zhang was partially supported by the J. T. Oden ICES Postdoctoral Fellowship at the Institute for Computational Engineering and Sciences. This research of Y. Zhang, S. Goswami, and C. Bajaj was supported in part by NSF grants EIA-0325550, CNS-0540033, and NIH grants P20-RR020647, R01-GM074258, R01-GM073087. This support is gratefully acknowledged. We would also like to thank Fred Nugen, Bob Moser, and Jeff Gohean for providing us with the data for the thoracic aorta model.

References

1. Taylor C.A., Hughes T.J., Zarins C.K. "Finite element modeling of blood flow in arteries." *Computer Methods in Applied Mechanics and Engineering*, vol. 158, 155–196, 1998
2. Bazilevs Y., Calo V., Zhang Y., Hughes T.J. "Isogeometric Fluid-Structure Interaction Analysis with Applications to Arterial Blood Flow." *Computational Mechanics*, 2006
3. Hughes T.J., Cottrell J.A., Bazilevs Y. "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement." *Computer Methods in Applied Mechanics and Engineering*, vol. 194, 4135–4195, 2005
4. Cottrell J., Reali A., Bazilevs Y., Hughes T. "Isogeometric analysis of structural vibrations." *Computer Methods in Applied Mechanics and Engineering*, 2005. In press

5. Bazilevs Y., da Veiga L.B., Cottrell J., Hughes T., Sangalli G. "Isogeometric analysis: Approximation, stability and error estimates for h -refined meshes." *Mathematical Models and Methods in Applied Sciences*, 2006. Submitted, available as ICES report 06-04
6. Rogers D.F. *An Introduction to NURBS With Historical Perspective*. Academic Press, San Diego, CA, 2001
7. Piegl L., Tiller W. *The NURBS Book (Monographs in Visual Communication)*, 2nd ed. Springer-Verlag, New York, 1997
8. Bajaj C., Chen J., Xu G. "Modeling with Cubic A-Patches." *ACM Transactions on Graphics*, vol. 14, 103–133, 1995
9. Sederberg T.W., Cardon D.L., Finnigan G.T., North N.S., Zheng J., Lyche T. "T-Spline Simplification and Local Refinement." *ACM Transactions on Graphics (TOG), SIG-GRAPH*, vol. 23, 276–283, 2004
10. Cirak F., Scott M.J., Antonsson E.K., Ortiz M., Schröder P. "Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision." *Computer-Aided Design*, vol. 34, 137–148, 2002
11. Blacker T. "A New Approach to Automated Quadrilateral Mesh Generation." *Int. J. Numer. Meth. Engng*, vol. 32, 811–847, 1991
12. Cook W.A., Oakes W.R. "Mapping methods for generating three-dimensional meshes." *Computers in Mechanical Engineering*, pp. 67–72, 1982
13. "CUBIT Mesh Generation Toolkit. Web site: <http://sass1693.sandia.gov/cubit>."
14. Blacker T. "The Cooper Tool." *5th International Meshing Roundtable*, pp. 13–29, 1996
15. Shepherd J., Mitchell S., Knupp P., White D. "Methods for MultiSweep Automation." *9th International Meshing Roundtable*, pp. 77–87, 2000
16. Knupp P. "Next-Generation Sweep Tool: A Method for Generating All-Hex Meshes on Two-And-One-Half Dimensional Geometries." *7th International Meshing Roundtable*, pp. 505–513, 1998
17. White D., Saigal S., Owen S. "Automatic Decomposition of Multi-Sweep Volumes." *Engineering With Computers*, vol. 20, 222–236, 2004
18. Staten M., Canaan S., Owen S. "BMSweep: Locating Interior Nodes During Sweeping." *7th International Meshing Roundtable*, pp. 7–18, 1998
19. Scott M., Earp M., Benzley S. "Adaptive Sweeping Techniques." *14th International Meshing Roundtable*, pp. 417–432, 2005
20. Armstrong C., Robinson D., McKeag R., Li T., Bridgett S., Donaghy R., McGleenan C. "Medials for Meshing and More." *4th Int. Meshing Roundtable*, pp. 277–288, 1995
21. Price M.A., Armstrong C.G., Sabin M.A. "Hexahedral Mesh Generation by Medial Surface Subdivision: I. Solids with Convex Edges." *Int. J. Numer. Meth. Engng.*, vol. 38, 3335–3359, 1995
22. Storti D., Turkiyyah G., Ganter M., Lim C., Stal D. "Skeleton-based modeling operations on solids." *ACM Symposium Solid Modeling Applications*, pp. 141–154, 1997
23. Quadros W.R., Owen S.J., Brewer M., Shimada K. "Finite Element Mesh Sizing for Surfaces Using Skeleton." *13th International Meshing Roundtable*, pp. 389–400, 2004
24. Zhang Y., Bajaj C., Sohn B.S. "3D Finite Element Meshing from Imaging Data." *The special issue of Computer Methods in Applied Mechanics and Engineering (CMAME) on Unstructured Mesh Generation*, vol. 194, no. 48-49, 5083–5106, 2005
25. Zhang Y., Bajaj C. "Adaptive and Quality Quadrilateral/Hexahedral Meshing from Volumetric Data." *Computer Methods in Applied Mechanics and Engineering (CMAME)*, vol. 195, no. 9-12, 942–960, 2006
26. Ito Y., Shum P.C., Shih A., Soni B., Nakahashi K. "Robust generation of high-quality unstructured meshes on realistic biomedical geometry." *Int. J. Numer. Meth. Engng.*, vol. 65, 943–973, 2006

27. Zachariah S.G., Sanders J.E., Turkiyyah G.M. "Automated Hexahedral Mesh Generation from Biomedical Image Data: Applications in Limb Prosthetics." *IEEE Transactions on Rehabilitation Engineering*, vol. 4, no. 2, 91–102, 1996
28. Verma C.S., Fischer P.F., Lee S.E., Loth F. "An All-Hex Meshing Strategy for Bifurcation Geometries in Vascular Flow Simulation." *14th International Meshing Roundtable*, pp. 363–375, 2005
29. Thompson J.F., Soni B.K., Weatherill N.P. *Grid Generation*. CRC Press LLC, 1999
30. Gursoy H.N. "Tetrahedral Finite Element Mesh Generation from NURBS Solid Models." *Engineering with Computers*, vol. 12, no. 19, 211–223, 1996
31. Anderson C.W., Crawford-Hines S. "Fast Generation of NURBS Surfaces from Polygonal Mesh Models of Human Anatomy." *Technical Report CS-99-101, Colorado State University*, 2000
32. Yu T.Y., Soni B.K. "NURBS Evaluation and Utilization for Grid Generation." *5th International Conference on Numerical Grid Generation in Computational Field Simulations*, pp. 323–332, 1996
33. Yu Z., Bajaj C. "A Fast and Adaptive Algorithm for Image Contrast Enhancement." *IEEE International Conference on Image Processing (ICIP'04)*, vol. 2, pp. 1001–1004, 2004
34. Bajaj C., Wu Q., Xu G. "Level Set Based Volumetric Anisotropic Diffusion." *ICES Technical Report 301, the Univ. of Texas at Austin*, 2003
35. Tomasi C., Madcuchi R. "Bilateral filtering for gray and color images." *IEEE International Conference on Computer Vision*, p. 839, 1998
36. Yu Z., Bajaj C. "Image Segmentation using Gradient Vector Diffusion and Region Merging." *16th International Conference on Pattern Recognition*, vol. 2, pp. 941–944, 2002
37. Lorensen W., Cline H. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." *SIGGRAPH*, pp. 163–169, 1987
38. Ju T., Losasso F., Schaefer S., Warren J. "Dual Contouring of Hermite Data." *SIGGRAPH*, pp. 339–346, 2002
39. Goswami S., Dey T.K., Bajaj C.L. "Identifying Flat and Tubular Regions of a Shape by Unstable Manifolds." *11th ACM Sympos. Solid and Physical Modeling, to appear*, 2006
40. Hughes T.J.R. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, Mineola, NY, 2000

Rapid Meshing of Turbomachinery Rows Using Semi-Unstructured Conformal Grids

Manuel A. Burgos², Roque Corral^{1,3}, Jaime Fernández-Castañeda¹, and Carlos López¹

¹ Industria de Turbo Propulsores S.A.,

² Universidad Politécnica de Cartagena

³ Universidad Politécnica de Madrid

Summary. A semi-unstructured grid generation method especially tailored for the meshing of turbomachinery blade passages and their associated cavities is presented. The method is based on a smart combination of quasi-3D methods, an ad-hoc block decomposition of the domain and a grid-based reconstruction of the computational domain.

Key words: Grid, Hybrid, Delaunay, Semi-Unstructured, Turbomachinery

Introduction

The aerodynamic design of uncooled turbomachinery rows is based, among other things, on the repeated simulation of three-dimensional blade rows with a fast turn-around time. Traditionally blade rows have been simulated using clean aerodynamic surfaces and simplified computational domains consisting of just the aerodynamic surface and an approximated main annulus. A more accurate and realistic representation of the flow geometry would require the inclusion of the tip-shroud and under platform cavities, the airfoil platforms themselves and the sealing flows, which on the other hand have been shown to have a large influence on the development of secondary flows (see Figs. 1 and 2).

The motivation for addressing this specific topic is two-fold. First, the computational resources have traditionally limited the overall number of grid points in the simulations and hence their level of complexity. Second, and more important, the efficient setting of turbomachinery simulations has been traditionally a problem, which is aggravated by the presence of tip-shroud and platform cavities.

The steady growth of computational resources has paved the way to overnight simulations of blade rows including its cavities with a small number of CPUs, however if these simulations have to be included in a design process, the generation of the associated grids need to be performed in the order of tens of minutes or at most a few hours.

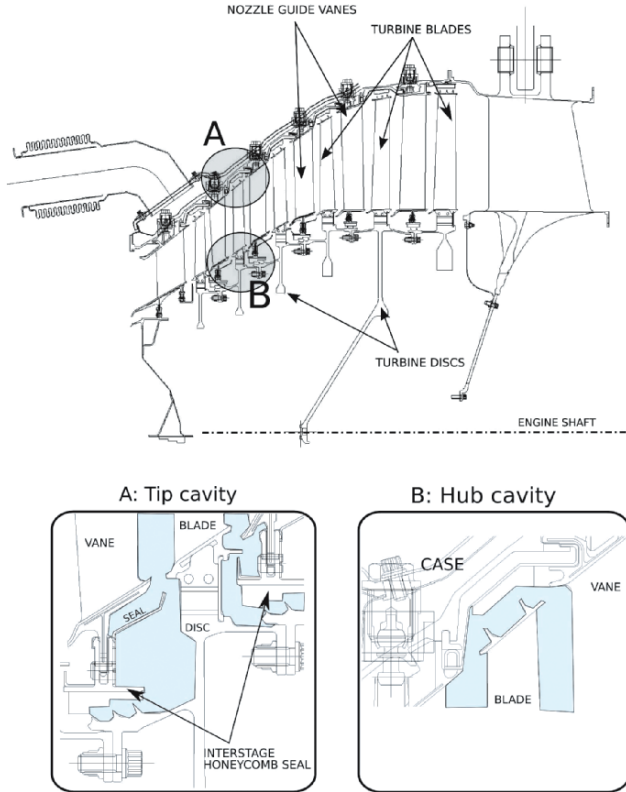


Fig. 1. Real geometry and turbine terminology.

Configurations as the one outlined in Fig. 3 may be meshed using multi-block structured methods. The departing point is usually a CAD 2D drafting model that do not define the fluid domain. The solid model is usually constructed by another specialist and many details, usually irrelevant from a fluid dynamics point of view, are included on it. Obtaining a proper mesh for such configurations may be a one-week task.

Multi-block structured meshing requires to accommodate the geometry in the block interfaces to concentrate nodes in high gradient regions and determine the block topology. The whole process is very slow and not suitable in a design environment.

The use of fully unstructured methods presents several problems as well. The airfoil surface has a large curvature in the leading edge region in the downstream direction, while in the span-wise direction the curvature is much smaller. The meshing process is slow, not reliable enough and tends to generate more nodes than necessary in the radial direction.

The generation of fast and robust *ad hoc* methods for this type of configurations is hence very attractive.

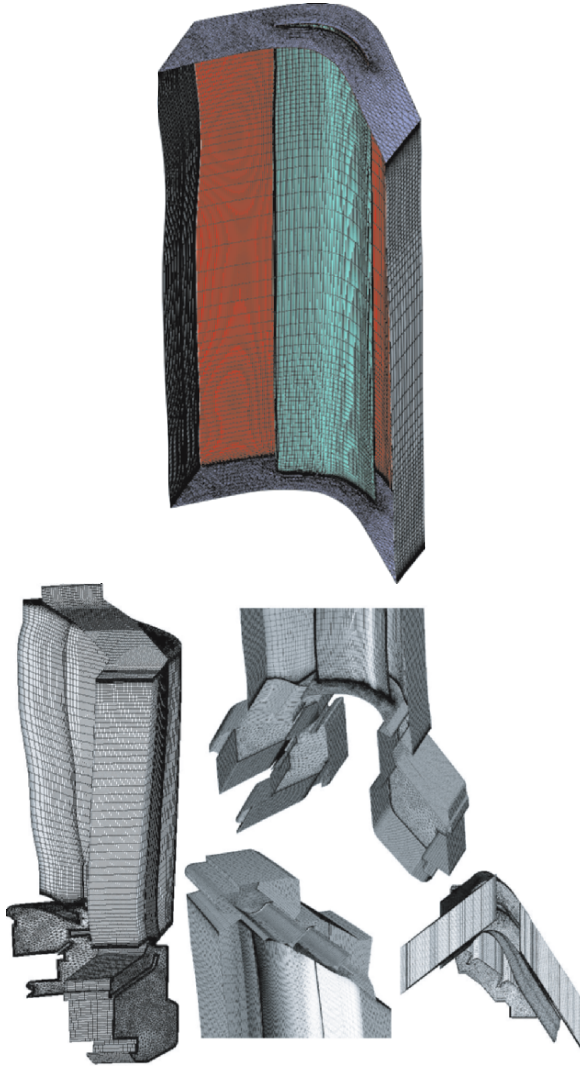


Fig. 2. Traditional design grids (top) and new grids (bottom).

Method Overview

The method exploits several particularities of the configuration to significantly ease, not only the meshing procedure, but the post-processing of the simulations. First the computational domain is conceptually subdivided in sub-domains as it is sketched in Fig. 4 left. However the 3D solid model is not strictly needed and the grid is constructed making extensive use of the freedom that provides the periodicity condition in the azimuthal direction and the use of semi-structured grids.

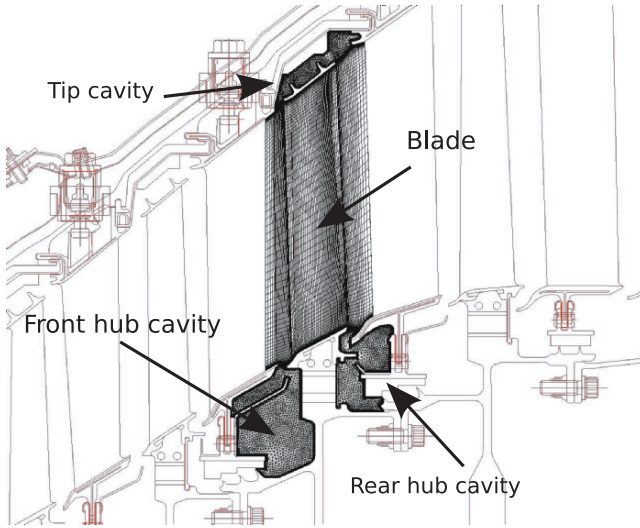


Fig. 3. General arrangement of a turbo-machine and grid of the computational domain

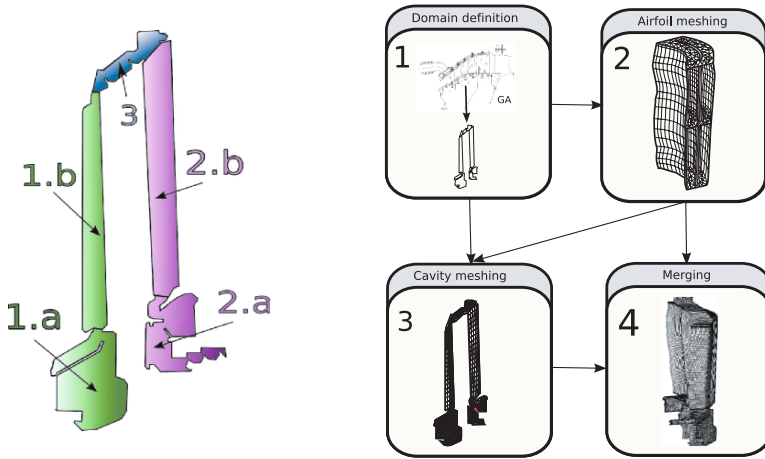


Fig. 4. Conceptual block decomposition (left) and scheme of the meshing process (right)

It is important to highlight that the only truly three-dimensional feature in the domain is the blade airfoil. The tip-shroud cavity and the under-platform cavities are essentially axi-symmetric configurations where the azimuthal position of the lateral boundaries is free, provided that the blade pitch is covered, and do not need to be fixed in advance. Both parts may be meshed using semi-unstructured methods. The unstructured part of the method copes with the more complex part of the geometry (e.g.: The definition of the cavity in the meridional plane) while the structured part deals with the easiest part of the meshing process (e.g: in the cavity the 3D mesh is

obtained revolving the 2D unstructured grid). The difficult part of the problem is to combine all the blocks in a consistent way without generating non-conformal grids and satisfying the designer's requirements in terms of point distribution.

The method considers two types of geometry. The airfoil surface that is created by a specific application and that will be considered given in this work, and the two dimensional geometry, that defines the cavities and the inlet/outlet boundary conditions, and whose revolution defines the computational domain that contains the blade surface. The latter is usually constructed in a CAD environment and need to be cleaned to eliminate details which are not relevant for the simulations.

The method needs to handle information coming from different sources. In order to integrate the process in a design environment a specific application has been developed to keep track of all the information and control the different interfaces. Turbomachinery design is conducted in a hierarchical way and this philosophy has been kept in the actual process.

First a semi-unstructured grid about the airfoil is generated using a system especially designed for the meshing of blade passages that is currently used routinely[1]. This method produces hexahedral and prismatic cells that are easy to merge with revolved two-dimensional grids. The sub-domain is shorter than the one used in the absence of cavities, which spans from the trailing edge of the previous row to the leading edge of the next (see Fig. 3). The inlet and outlet angles are obtained from through-flow data (meridional analysis).

In second place the cavity sub-domains are defined and meshed using hybrid grids. Then the rest of the blade passage is meshed using a structured grid. Especial care has to be taken to ensure the compatibility of the node distribution of this mesh with the airfoil and cavity grids.

Finally the 2D grids are revolved and the resulting grid merged with the semi-unstructured grid of the airfoil. A design environment has been developed, which manages all information, ensures the compatibility of the grids, merges them automatically and creates boundary conditions and consistent initial flow solution.

Geometry Generation

A state-of-the-art turbomachinery design environment must deal with complex configurations and build simulation models in just a few hours. To obtain this objective, a trade off between generic and tuned tools to speed-up the overall process is mandatory. In the context of the present work, all final simulation models are three-dimensional, although advantage is taken, whenever it is possible, of the quasi-axisymmetric nature of the problem.

A CAD-like geometric kernel is used to support all the geometry translation, creation and manipulation, as well as the grid generation. The underlying entities supporting all the geometry are Non-Uniform Rational B-Splines (NURBS) and the solid model topological data structure is a Radial Edge Non-Manifold Boundary Representation (RENMB-Rep).

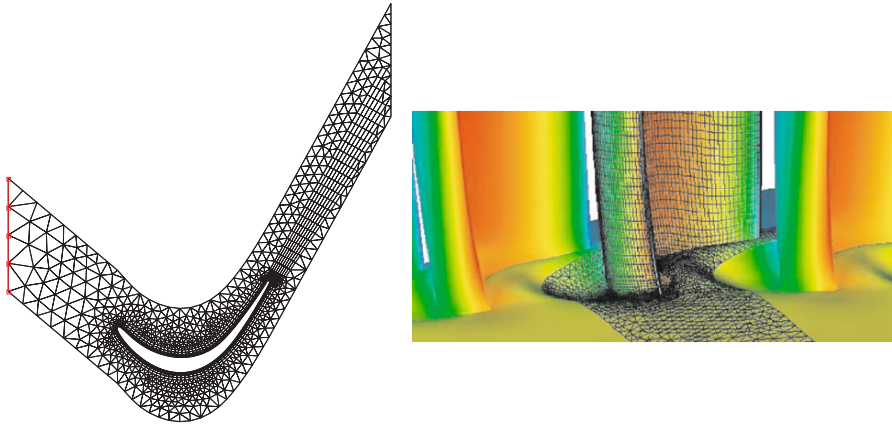


Fig. 5. Hybrid grid on an airfoil (left) and semi-structured grid with a profiled end-wall detail (right)

Passage and Cavities

Including the cavities in the simulation increases significantly the amount of information needed to set up the grid. The cavities are complex and are defined in a CAD tool, together with the rest of the engine, by a designer. The general arrangement (GA) contains usually more information that required including a higher level of details than needed for the simulations, drafting information, other layers, etc.

The GA is imported as an IGES file. The micro-edges are eliminated and the geometry cleaned, which is more efficient than redraw the full geometry. If needed this can be redesign within the same environment. At this point it is possible to decide how to subdivide the domain in blocks. Three blocks are necessary for the latter merging (see Fig. 4) but it is recommended to subdivide into sub-blocks to improve the quality of the grid (meshing zones 1.b and 2.b with structured grids).

Aerodynamic Surfaces

The airfoil surface is provided by the aerodynamic designer and is considered as given in the present work. The baseline computation considers the end-wall as a smooth axi-symmetric surface. The real platform geometries considered here require a modification of the baseline flow-path since it presents discontinuities (gaps) with the rest of the passage end-wall. This problem is illustrated in Fig. 7. When non-axisymmetric end-wall (NAEs) are considered to reduce secondary losses (see Fig. 5, right) the geometry is considered given as well, the main reference with the previous case is that this surface is continuous with respect the baseline axi-symmetric end-wall.

Meshing Techniques

Two-Dimensional Unstructured Triangulation

The computational domain of some blocks (i.e. 1.a and 2.a and 3 in figure 4) and a reference section of the blade grid is tessellated using Delaunay triangulations. Some approaches emphasise constructions based on edge swappings (Lawson's algorithm), whereas others rely on the in-circle property (Bowyer-Watson method). The present algorithm makes use of both methods to increase the robustness of the system. Enforcing the prescribed geometry is usually done in two post-processing steps. First, the edges that define the geometry are required to exist. In the present approach an arbitrary set of edges, not necessarily conforming to a boundary, is forced to exist by executing a sequence of swappings [2]. In a second step, all of the triangles placed outside of the domain are flagged and removed, making use of the orientation associated to the closed circuits that define them and of a consistent ordering of the nodes that constitute the triangles.

Interior nodes of the domain are inserted making use of a function that provides a measure of the desired size of the triangles [3]. This element-size function is built just from the information contained in the point distribution of the domain boundaries and of the prescribed curves in its interior. The key idea of the procedure is to use as field point distribution function the solution of a Laplace equation with Dirichlet boundary conditions.

Mesh quality is improved in two different steps: grid smoothing and node-degree homogenisation. The latter process is useful to break some topological structures that tend to appear during the course of the triangulation and that prevent further improvement of the grid. The original two-dimensional procedure was generalised in order to obtain grids on arbitrary surfaces [4]. To reduce the computational time of edge-based solvers and increase the grid quality, triangles may be converted in an unstructured grid of quads [5].

Two-Dimensional Unstructured Viscous Grid

Grids with high-aspect-ratio cells are generated using an advancing normal technique [6] to build layers of stretched quadrilaterals in the boundary layer and wake regions. Normals are advanced from the selected boundaries where a distribution of points is assigned to each boundary node. Once a layer of normals has been constructed the normal directions are smoothed to improve the quality of the grid. Each single normal may be prevented from growing if either the associated cell reach an aspect ratio close to a certain threshold, usually of order unity, or another node or edge is found in the advancing direction.

Two-Dimensional Structured Grid

Some blocks (i.e 1.b and 2.b in figure 4) of our problem are meshed using structured grids based on the solution of elliptic partial differential equations (PDEs). Forcing

terms are used to construct stretched layers of the cells close to the domain boundaries. Control functions are computed using the boundary point spacing and then interpolated to the inner points [8][9]. The forcing terms are computed as

$$P = -\frac{\frac{\partial \mathbf{r}}{\partial \xi} \frac{\partial^2 \mathbf{r}}{\partial \xi^2}}{\left| \frac{\partial \mathbf{r}}{\partial \xi} \right|^2} \quad Q = \frac{\frac{\partial \mathbf{r}}{\partial \eta} \frac{\partial^2 \mathbf{r}}{\partial \eta^2}}{\left| \frac{\partial \mathbf{r}}{\partial \eta} \right|^2}$$

Once P and Q are obtained at each boundary the values for the inner points are obtained interpolating along lines of constant ξ and η :

$$P(\xi, \eta) = (1 - \eta) P_1(\xi) + \eta P_2(\xi) \quad 0 \leq \xi \leq 1$$

$$Q(\xi, \eta) = (1 - \xi) Q_1(\eta) + \xi Q_2(\eta) \quad 0 \leq \eta \leq 1$$

Grid control of orthogonality at boundaries is introduced adding a second term in P and Q,

$$P = -\frac{\frac{\partial \mathbf{r}}{\partial \xi} \frac{\partial^2 \mathbf{r}}{\partial \xi^2}}{\left| \frac{\partial \mathbf{r}}{\partial \xi} \right|^2} - \lambda \frac{\frac{\partial \mathbf{r}}{\partial \xi} \frac{\partial^2 \mathbf{r}}{\partial \eta^2}}{\left| \frac{\partial \mathbf{r}}{\partial \eta} \right|^2} \quad Q = \frac{\frac{\partial \mathbf{r}}{\partial \eta} \frac{\partial^2 \mathbf{r}}{\partial \eta^2}}{\left| \frac{\partial \mathbf{r}}{\partial \eta} \right|^2} - \lambda \frac{\frac{\partial \mathbf{r}}{\partial \eta} \frac{\partial^2 \mathbf{r}}{\partial \xi^2}}{\left| \frac{\partial \mathbf{r}}{\partial \xi} \right|^2}$$

where $0 < \lambda < 1$ is a factor that relaxes the orthogonality at the boundaries. It has been observed that the range $\lambda \in [0.4-0.7]$ produces optimal results for our configurations.

The elliptic PDEs are solved using a multi-grid method and the smoother is based on a point-wise Newton solver. When the forcing terms are used the convergence of the algorithm deteriorates slightly.

Grid Topology Model

An efficient data structure was implemented to support the grid generation process in geometrical complex models. Solid models are stored in a Radial Edge Non-Manifold (RENM) data structure and numerical grid entities are stored in a Grid Topology Model (GTM) [7]. The GTM data structure explicitly defines the adjacency between grid entities, abstracts the grid from the geometry, eliminates duplicated grid points and provides surface direction normalisation for grid generation.

Grid Generation Methodology

Airfoil Passage Meshing

A semi-unstructured grid generation system was specially designed for the meshing of blade passages [1]. The main idea is to build an hybrid two-dimensional grid

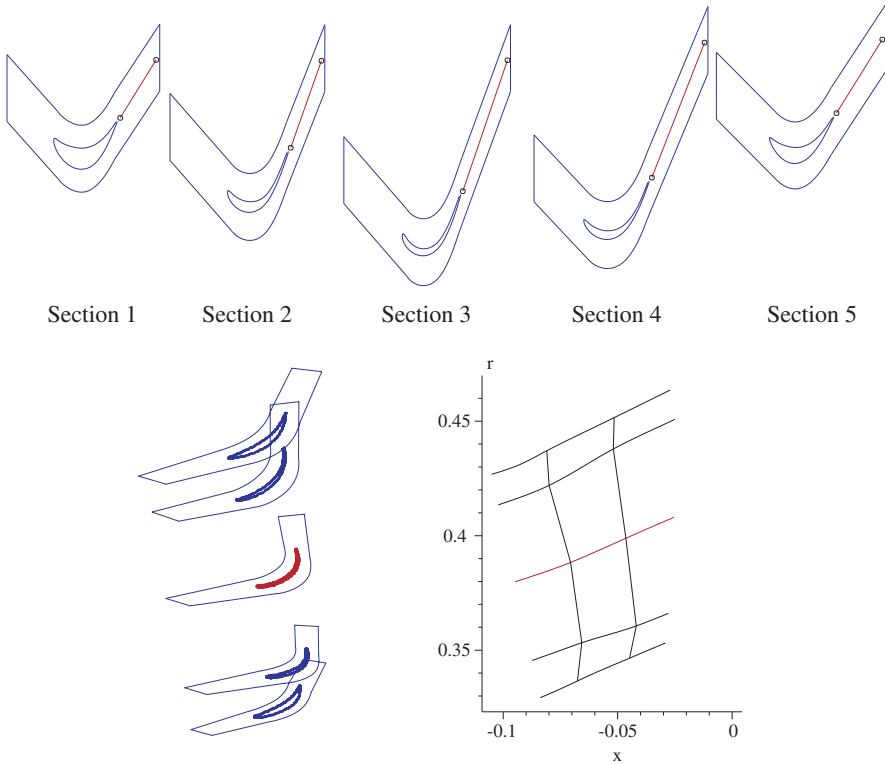


Fig. 6. Two-dimensional domains and data required for the generation of a semi-unstructured grid

for a reference blade-to-blade section and generate a continuous mapping from that section to the rest, keeping the same topology. Then, the triangles and quadrangles of the adjacent sections are connected to form triangular prisms and hexahedra.

The process is fast and robust and the quality of the resulting grid easy to monitor by inspecting each of the sections. The current approach uses the NURBS definition of the blade surface, the boundary conditions of the row and the streamlines from a through-flow program. The first step is to select the number and distribution of radial sections. Typically, sections are clustered in the hub and tip regions to provide adequate resolution of the end-wall boundary layers and the cores of the secondary vortex system. Once the blade sections have been fixed, the intersections of the NURBS definition of the blade and the stream-surfaces associated at each streamline are computed and stored. Simultaneously, the passage of each section is automatically computed. Then, one of the sections is chosen as the reference section and an hybrid grid is constructed in the $m' - \theta$ plane⁴ of the corresponding stream-surface.

⁴Parameter m' is defined as follows:

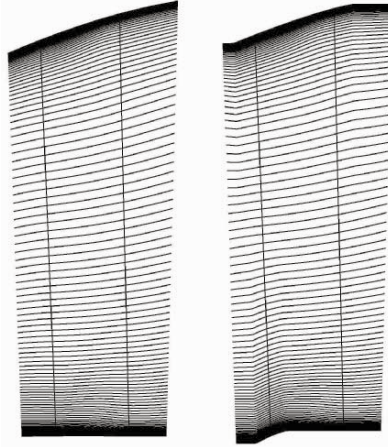


Fig. 7. Modification of the baseline geometry to introduce platform geometry and profiled end-walls.

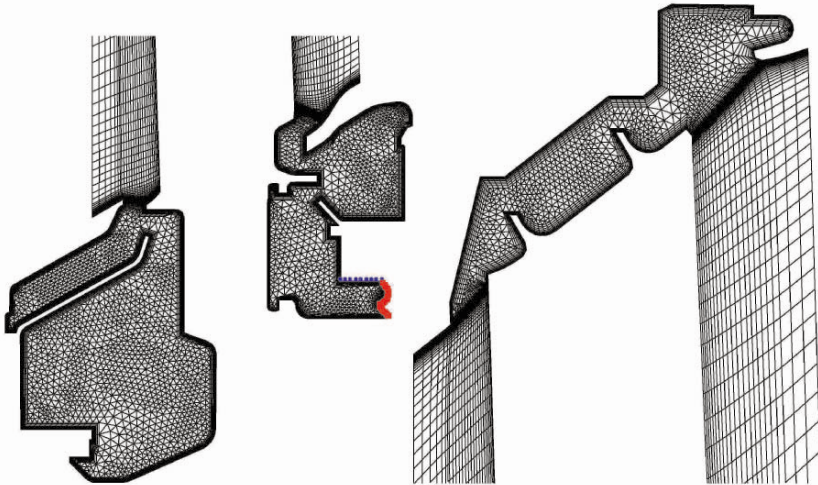


Fig. 8. Close-up of the 2D hybrid grids in the Cavities

At this stage the homologous points of the current section are selected, the existing 2D grid mapped to the rest of the section and a three-dimensional connectivity defined (see Fig. 6).

$$m'(x) = \int_{x_0}^x \frac{ds}{r} = \int_{x_0}^x \frac{\sqrt{1 + (r'(x))^2}}{r(x)} dx$$

where x is the axial axis of the engine. This parameterization is used in turbomachinery to deal with 2D CFD simulations on stream-surfaces due its properties i.e., preserves angles, is non-dimensional and locally length-coherent...

Non axi-symmetric end-walls and platform profiles are introduced as perturbations of the axi-symmetric nominal tip and hub stream-surfaces defined by the trough-flow analysis. The perturbations are gradually damped away from the walls until an axi-symmetric stream-surface is recovered.

When the cavities are included the length of the passage meshed by this method is reduced to an arbitrary distance from the blade contained within the upper and lower platforms.

Cavity Meshing

Cavities are meshed in the (x, r) plane using hybrid grids. Stretched layers in the vicinity of the wall are constructed using an advancing layer method and revolved later for merging. Node distribution of the passage in platform gap region is forced to be the same as the one of the cavity.

To generate a high quality mesh, the rest of the passage is meshed using a structured grid for which it is necessary to generate a block.. Continuity of the stretched cells in the blade passage block has to be provided here (see fig. 8). The cavities themselves are usually meshed using hybrid grids, thus, blocks 1.a and 2.a have structured grid and blocks 1.b, 3 and 2.b have hybrid grids (see Fig. 4). Triangles and quadrilaterals are transformed in hexahedral and prismatic cells respectively upon revolution, although the possibility of building an structured 2D multi-blocks grid remains for blocks 1.a, 2.a and 3 (but user will spend more time).

Merging

Once the 3D blade passage grid and the 2D cavity grid have been generated (see Fig. 9 left), the latter has to be revolved to form a 3D grid and then merged with the former.

The grid of the cavity is defined in the (x, r) plane and the inlet and outlet planes of the blade passage grid have a unique projection in the (x, r) plane defined by $r_b(x)$ and an arbitrary distribution of nodes in the azimuthal direction, although in practice, at the inlet the nodes are distributed uniformly in the azimuthal direction, while at the outlet a node clustering is typically seen in the middle of the passage to increase the definition in the wake region.

The creation of a conformal high-quality grid requires a point-wise definition of the extrusion of the 2D grid. To begin with we assume a reference angle θ_0 and a variation of the origin for each radial section $\theta_0 + \alpha_1(r)$ (see Fig. 9 right). Then the azimuthal variation in the passage is represented by θ_j . If the distribution were uniform $\theta_j = (j - 1)\Delta\theta$ with $\Delta\theta = \text{Blade Pitch}/(N_\theta - 1)$ where N_θ is the number of points in the azimuthal direction. The azimuthal angle in the interface plane is then

$$\theta_j(x_I) = \theta_0 + \alpha_1(r) + \theta_j.$$

The azimuthal position of the nodes is further modified to aligned the grid located in the main annulus with the flow as it is sketched in Fig. 10 where the 2D revolved

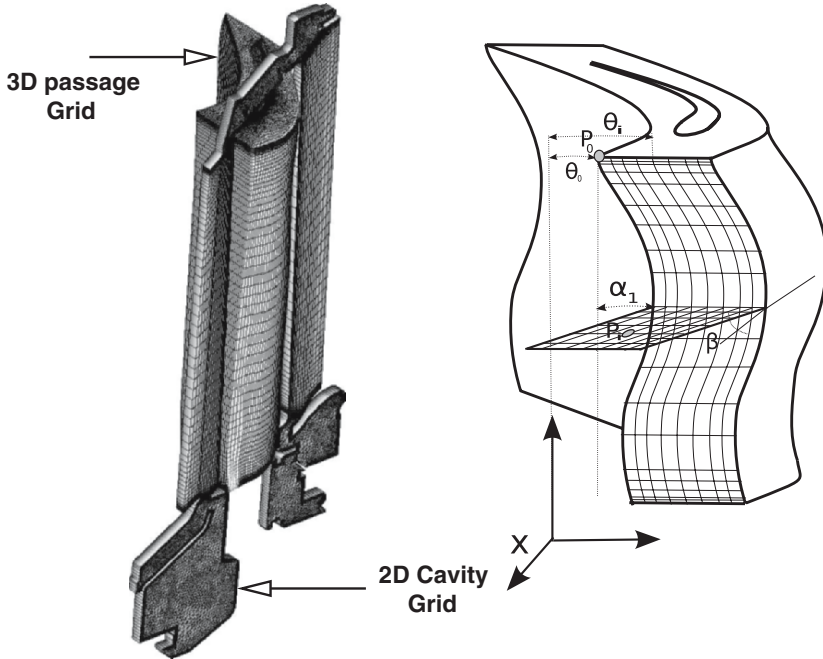


Fig. 9. Description of the merging problem (left) and adaption of the extruded 2D grid to the 3D grid (right)

grid is depicted. The azimuthal position of the extruded grid taking into account this feature is then

$$\theta(x, r) = \theta(x_I) + \frac{x - x_I}{r} \tan \beta$$

Once all the grids have been created are merged in a single grid, all the duplicated nodes are removed. Especial care has to be taken in the boundary layer regions where the grid spacing is very small and the generation of a conformal grid complex.

Preprocessing Environment

To integrate the present method in a design environment it was necessary to create a specific tool to manage all the information required, launch specific programs to mesh different blocks, ensure the compatibility among the different blocks and merge them automatically.

Automation of the whole process is possible due to the assignation by the user of attributes to the different geometric entities. This allows to mesh the redesigns reusing the grid parameters of the previous designs.

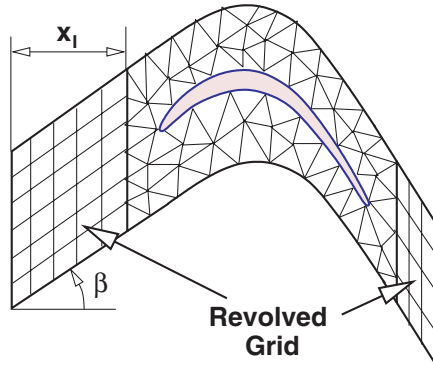


Fig. 10. Sketch of the grid in the main annulus

Initial flow solution and boundary conditions are set-up while meshing the different sub-blocks and transferred to the final grid. To ease the CFD convergence process the environment also handles the interpolation of the flow solution from other simulations, with or without cavities.

The main idea is to perform the whole work in a single environment that guides the user to set up of the grid and to where all the information is readily available.

Application Examples

The fifth stator of a low-pressure turbine has been chosen to test the capabilities of the method. The same geometry had been previously meshed using a multi-block structured method (see Fig. 11). The main difference between the structured grid and our method (Fig. 12) is due to the propagation of the stretching in the boundary layer region on the cavities and the passage to obtain a conformal grid. These additional points are not only undesirable from a computational point of view but represent a 30% of the total number of points.

Although the tip cavity has not been included in this mesh the number of nodes of the mesh with cavities approximately doubles the one of the single passage (about 1.6×10^6 points). The final grid is not only of better quality, but the time required to set the grid is reduced by one order of magnitude (from one week to a few hours)

Concluding Remarks

A method to set up CFD simulations for turbomachinery rows including its adjacent cavities in a few hours has been presented. The method constructs the grid departing from the general arrangement, the blade surface and the trough-flow data, that contain value data to construct the grid as the inlet and outlet angles.

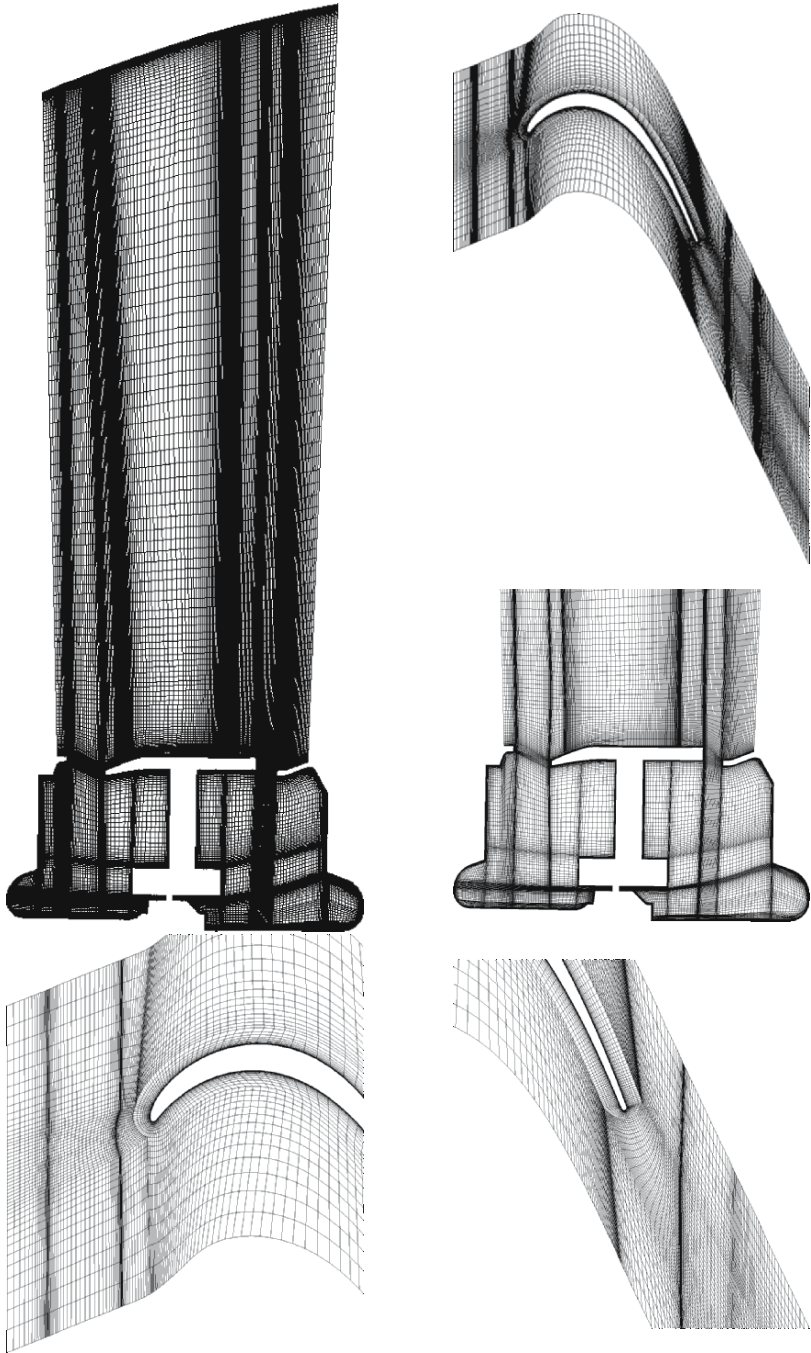


Fig. 11. Multi-block grid of the fifth rotor of the Trent 1000 engine low-pressure turbine (3.8×10^6 nodes)

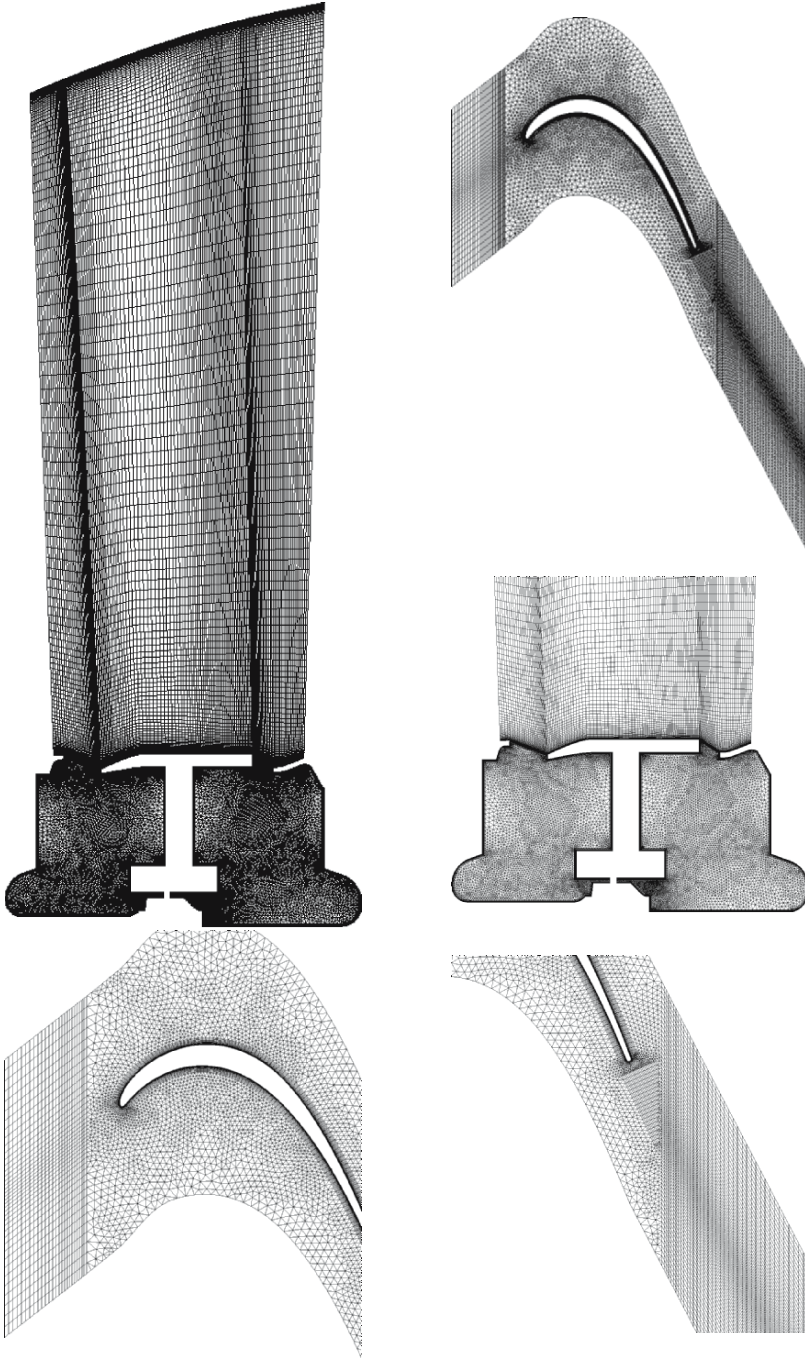


Fig. 12. Semi-structured grid of the fifth rotor of the low-pressure turbine of the Trent 1000 engine (3.0×10^6 nodes)

The current approach constructs the grid without making use a solid model. Instead directly uses the 2D geometry to construct the cavity grids revolving unstructured grids. The blade passage mesh is constructed using as well a 3D semi-structured grid.

Approximately the number of points in the cavities is the same than the number of nodes in the blade passage, however the complexity in the generation of the grid does not increase significantly.

Today the largest limitation of the method consists in the accommodation of the fillets between the blade and the platforms, that are known to significantly impact the development of secondary flows.

References

1. J. Contreras, R. Corral, J. Fernandez-Castañeda, G Pastor and C. Vasco, *Semi-Unstructured Grid Methods for Turbomachinery Applications*, ASME Paper 2002-GT-30572 presented at 47nd ASME Gas Turbine and Aeroengine Congress, Exposition and Users Symposium June 3-6, 2002, Amsterdam, The Netherlands.
2. S.W., Sloan, *A Fast Algorithm for Generating Constrained Delaunay Triangulations*, Computers & Structures, Vol 47, No. 3, pp. 441-450, 1993
3. R. Corral and J. Fernández-Castañeda, *Surface Mesh Generation by Means of Steiner Triangulations*, AIAA paper 98-3013, 1998
4. R. Corral, J. Fernandez-Castañeda, *Surface Mesh Generation by Means of Steiner Triangulations*, AIAA Journal Vol. 39, No. 1, pp. 178-180, 2001
5. S.J. Owen, M.L. Staten, S.A. Canann, S. Saigal, *Q-Morph an Indirect Approach to Advancing Front Quad Meshing*, Int. J. Numer. Meth Enging, Vol 44, 1317-1340, 1999
6. Pirzadeh, S., *Unstructured Viscous Grid Generation by the Advancing Layers Method*, AIAA Paper 93-3453, 1993
7. A. Gaither, *A Boundary Representation Solid Modelling Data Structure for General Numerical Grid Generation*, Master Thesis, Mississippi State University, 1997
8. Thomas P., and Middlecoff J. *Direct control of the Grid Point Distribution in Meshes Generated by Elliptic Equations*, AIAA Journal Vol. 18, No. 6., 1980
9. Sorenson R. L. and Steger J. L. *Numerical Generation of Two dimensional Grids by the Use of Poisson Equations with Grid Control*, in Numerical Grid Generation Techniques, R. E Smith, ed.. NASA CP 2166, NASA Langley Research Center, Hampton, VA, USA, 1980

Hybrid Mesh Generation for Viscous Flow Simulation

Yuanli Wang¹ and Samuel Murgie²

¹ ALGOR, Inc., 150 Beta Dr., Pittsburgh, PA USA, 15238

ywang@algor.com

² ALGOR, Inc., 150 Beta Dr., Pittsburgh, PA USA, 15238

smurgie@algor.com

Abstract. This paper presents a robust and automated approach to generate unstructured hybrid grids comprised of prismatic and tetrahedral elements for viscous flow computations. The hybrid mesh generation starts from a triangulated surface mesh. The prismatic elements are extruded based on the weak solutions of the Eikonal equation to generate anisotropic elements at boundaries, and finally the isotropic tetrahedral grids are generated to fill the rest of the domain. The presented hybrid meshing algorithm was validated using a ball valve model under both steady and unsteady conditions.

1 Introduction

A great challenge for viscous flow simulations is to gain anisotropic elements at the vicinity of the boundary area, especially for complex geometric solid surfaces. When such domains are discretized, it is important that the grid fits the boundaries well, and no conflict occurs during boundary mesh generation processes. These issues become even more difficult to achieve with strongly curved boundaries. The first issue relates to surface discretization or surface mesh generation, which is not discussed in this paper.

To effectively avoid the warping of normal directions during boundary meshing, it is critical to generate good-quality, high-aspect-ratio cells in the vicinity of boundaries for wall-dominated phenomena. For this purpose, a new approach based on a hybrid meshing methodology was proposed, in which normal directions are calculated using a weak solution of the Eikonal equation at each solid surface node of the boundaries to be propagated [1], [5], and [6].

The purpose of this paper is to validate the quality and effectiveness of the proposed new hybrid mesh generation strategy using a ball valve model. This paper is arranged as follows: Section 2 describes the problem to be solved. Section 3 gives a brief introduction of the methodology used in this work.

Section 4 illustrates the mesh results. Validation of the flow using the presented method is demonstrated in Section 5. Finally, Section 6 summarizes the presentation.

2 Problem Description

In this paper, the steady and unsteady fluid flows in a three-dimensional (3D) model of a ball valve are computed. Figure 1 shows the geometric image of a pipe section including a ball valve. The valve is in the half open position. The goal of the fluid flow analysis is to determine the velocity and the pressure of the fluid as it exits the section.

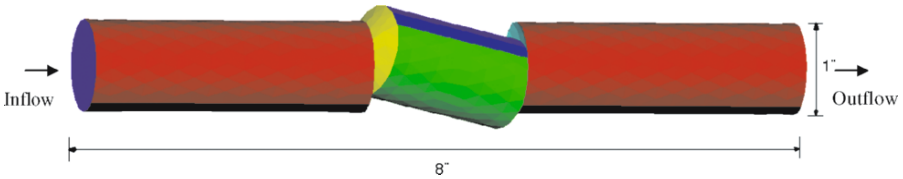


Fig. 1. Diagram of the pipe section

Topologically, the illustrated geometry is very simple and is equivalent to a cylinder. Geometrically, this configuration has both convex and concave shapes in its body. The most difficult consideration in this test case is how to correctly calculate normal directions at the four singularity points as shown in Figure 2. In this case, there are four surfaces that pass through and share the singularity point. The traditional normal calculation method uses the geometric information surrounding the point to be propagated, i.e., the weighted average normal vectors of neighboring surfaces. The definition of the normal vector may become ambiguous at special cases when normal direction is performed in this way. For example, in Figure 2, the definition of normal vectors for Surface 2 and Surface 4 are almost in opposite directions. It is difficult to determine a compromise normal vector at this point using the average normal vectors of neighboring surfaces.

Instead of using the traditional normal calculation, this paper uses a new way to calculate normal vectors to prevent the above-mentioned problem. The propagation strategy is described in Section 3, and the resulting meshes at singularity points are presented in Section 4.

To analyze the flow pattern within this system, geometric sizes are set as illustrated in Figure 1. The total length of the pipe section is 8.0 *in* and the diameter at intake areas and valves is 1.0 *in*. Several physical characteristics of the fluid are also pre-defined, including the flow rate at the inlet area as 0.5 *in/s* and the fluid as water.

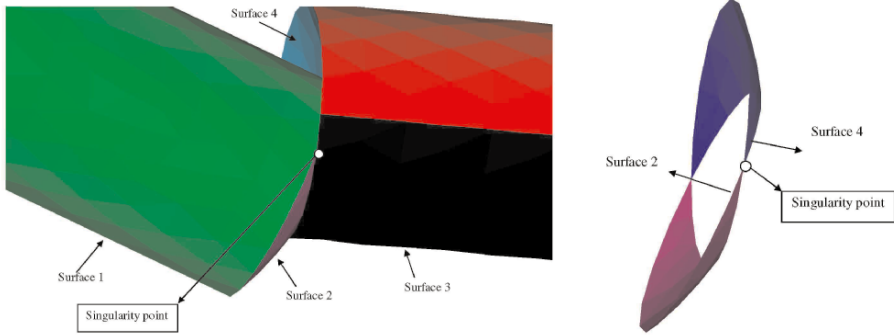


Fig. 2. Singularity point

3 Methodology

A primary challenge that exists in dealing with viscous flow concerns boundary mesh generation. Many boundary mesh generation methods have been proposed, and two important problems still exist. The first problem relates to the potential self-intersection of the front as it grows from the original surface. *Local* self-intersection may occur during propagation when the offset distance is greater than the local curvature radius in concave regions. *Global* self-intersection, on the other hand, arises when the distance between two distinct points on the curve or surface reaches a local minimum. The second difficulty, which is a more recent issue in offset construction, is the establishment of a common connectivity between the original and offset surfaces.

Two major types of methods in dealing with the computation of offset curves and surfaces are proposed: *direct offset methods (DOM)*, which propagate curves or surfaces directly based on a geometric construction; and *indirect offset methods (IOM)*, which cast the curve or surface offset problem into a set of partial differential equations (PDE), in which, geometric information is implicitly represented.

The advantage of DOM is that the entire or partial original parameterization information can be preserved (which is a rather attractive merit to boundary meshing), but the self-intersection problem cannot be avoided, and extra care is required for removing self-intersections. The ability to effectively eliminate these self-intersections is an important criterion in the applicability of such methods in the context of an automated procedure. One representative attempt to eliminate self-intersections is the *Advancing Front Method* developed by Pirzadeh [9] based on a grid-marching strategy. The solution is to simply stop the advancement of the front before self-intersections occur. Based on a similar marching idea, Sullivan [10] presented a self-intersection

detection and removal algorithm in 2D. The 3D algorithm developed by Guibault [8] eliminates self-intersections by first detecting tangled-loops and then re-locating the points located within this area. In the algorithm described by Glimm et al [11], a hybrid algorithm is applied to resolve self-intersections by either re-triangulating triangles after removing unphysical surfaces, or reconstructing the interface within each rectangular grid block in which crossing is detected. Other types of techniques to eliminate self-intersections use the properties of curves and surfaces, i.e., control points, derivatives, curvature, etc. Blomgren [12], Tiller and Hanson [13], and Coquillar [14] approached the problem by offsetting the control polygon for NURBS curves. Nachman [15] extended this idea to propagate surfaces by offsetting control points. Piegl and Tiller [16] sampled offset curves and surfaces based on bounds on the second derivatives to avoid self-intersections. In the method developed by Sun et al. [17], control points are repositioned to reduce local curvature in areas where local self-intersections may occur, while the rest of the control points remain unchanged. Farouki [18] described an algorithm which first decomposes the original surfaces into parametric patches, and then uses Hermite interpolation to construct the offset surfaces.

The representative work of IDM is the *level set method* developed by Sethian and Osher [19, 20, 21, 22] which models front propagation problems as a hyperbolic differential equation. In this method, self-intersection problems can be avoided, but at the cost of completely losing the connectivity information stored in the original geometric front. In general, to restore a similar connectivity between the original and offset fronts is not a trivial task.

The present work proposes to use a boundary mesh generation method discussed in [1], [5] and [6], which combines the advantages of the two types of methods to build a new offset construction method that maintains parametric connectivity between the original and offset surfaces, and still avoids self-intersections through the use of a weak solution to the shortest distance problem. Figure 3 shows the outline of the proposed mesh generation process. The details for each step are explained below.

Surface Mesh Generation

In the present hybrid meshing strategy, the surfaces are discretized into triangles [23, 24]. Figure 4 shows the initial surface mesh used in this test. There are no restrictions for the shape of surface mesh elements. The proposed boundary mesh process can accept any type of elements, e.g., triangular, quad, etc.

Boundary Mesh Generation

There are four essential steps in the boundary meshing process: (1) calculate the ϕ value for each grid node using the fast sweeping algorithm; (2) calculate normal directions for the front grid nodes; (3) propagate front points along

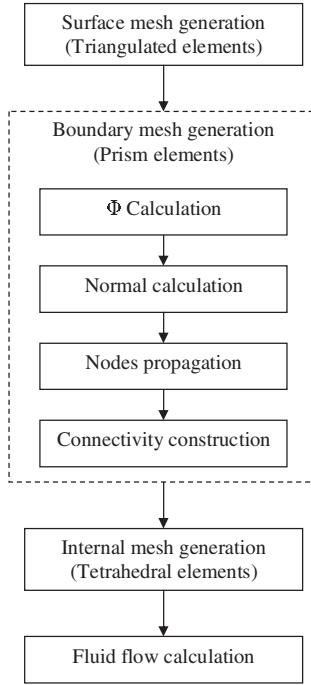


Fig. 3. Outline of mesh generation process

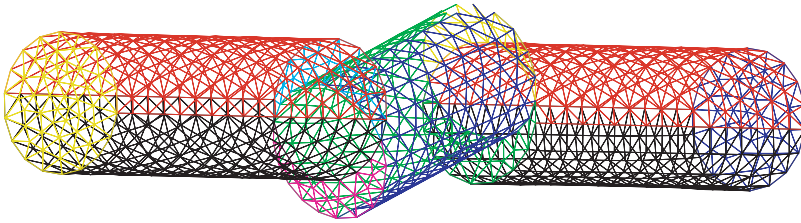


Fig. 4. Surface mesh

their local normal directions according to the given distance; (4) construct blocks or the relative topological connectivity around the boundary area.

1. ϕ calculation

The present work proposes using the *Offset Distance Equation*, which is a variation of the Eikonal equation, to model the surface offset problem. This equation is given below:

$$\begin{cases} \nabla\phi \cdot \nabla\phi = 1 \\ \phi = 0 \quad \text{if } \mathbf{P} \in \Gamma \end{cases} \quad (1)$$

where ϕ is the minimum Euclidean distance from an arbitrary point (P) in the computational domain to the front (Γ) to be propagated. Equation 1

expresses the condition for the shortest Euclidean distance from any space position to the boundary. To enforce the boundary condition, $\phi = 0$ when $\mathbf{P} \in \Gamma$, exact values are first assigned to boundary nodes. Then, Gauss-Seidel iterations with alternating sweeping orderings are used to update the ϕ s at the rest of the grid nodes. The details of this fast sweeping algorithm can be found in [2].

2. Normal calculation

At each grid node, the normal vector is represented by the equation

$$\mathbf{n} = \frac{\nabla\phi}{|\nabla\phi|} \quad (2)$$

where \mathbf{n} is the normal direction and ϕ is the weak solution of the Offset Distance Equation. The term $\nabla\phi/|\nabla\phi|$ can be viewed as a unit propagation speed in the normal direction: positive for an outward propagation, and negative for an inward propagation.

3. Nodes propagation

The propagation equation at each node is:

$$\mathbf{x}_t = F\mathbf{n} \quad (3)$$

Setting the propagation speed F at each node to 1.0, the new propagated points are obtained by iteratively solving Equation 3 using the fourth-order Runge-Kutta method [3].

4. Connectivity construction

All the propagated points are sequentially connected according to their original connectivities. The final propagation surface forms the input surface mesh for the volume mesh generation process. Figures 5 and 6 show the overall boundary mesh interface and the final boundary mesh of this ball valve model.

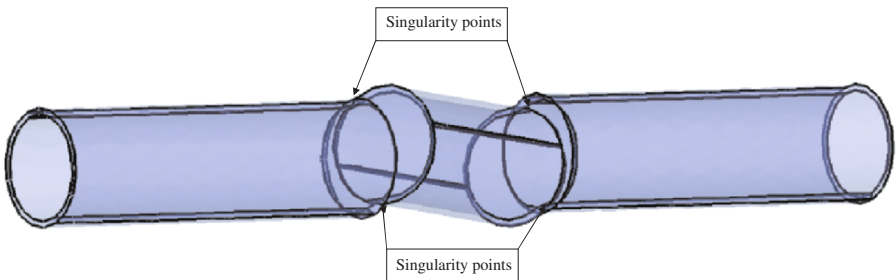


Fig. 5. Boundary mesh interface

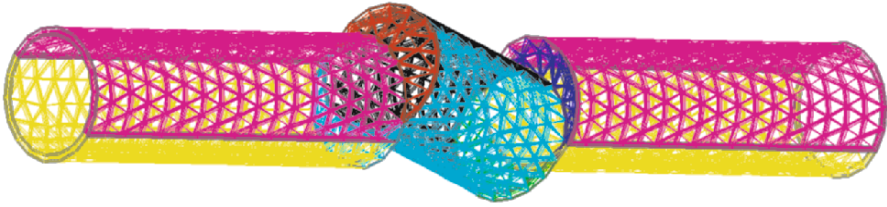


Fig. 6. Boundary mesh

Internal Mesh Generation

After the boundary meshing process, a new surface mesh is generated, which possesses exactly the same topological definition (triangle connectivity) as the original solid surface. This new surface mesh becomes the input surface mesh for tetra mesh generation. The entire computational domain defined by this surface mesh is tessellated by isotropic tetrahedra [7, 4]. Figures 7 and 8 illustrate the final mesh generated for this model, and Figures 9 shows the final mesh at the middle section.

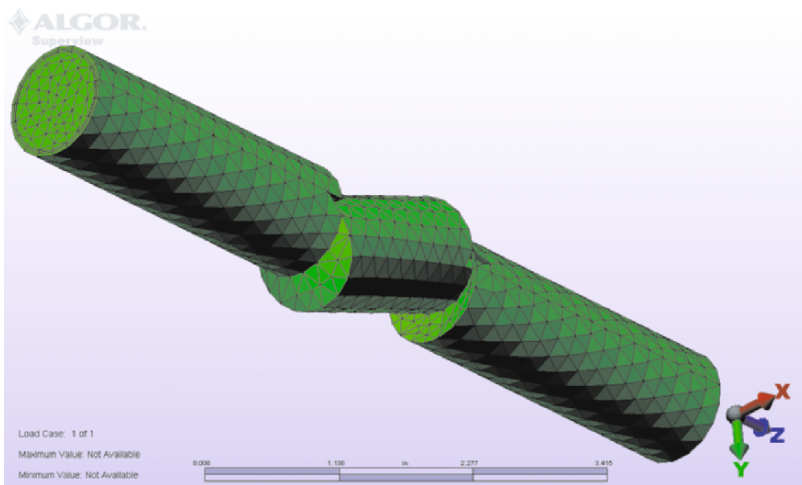


Fig. 7. Final mesh (outside view)

4 Mesh validation

This section will only evaluate the quality of the resulting mesh for prismatic elements.

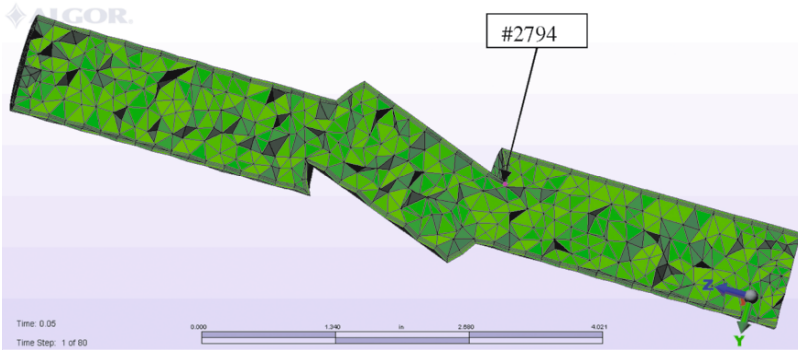


Fig. 8. Final mesh (internal view)

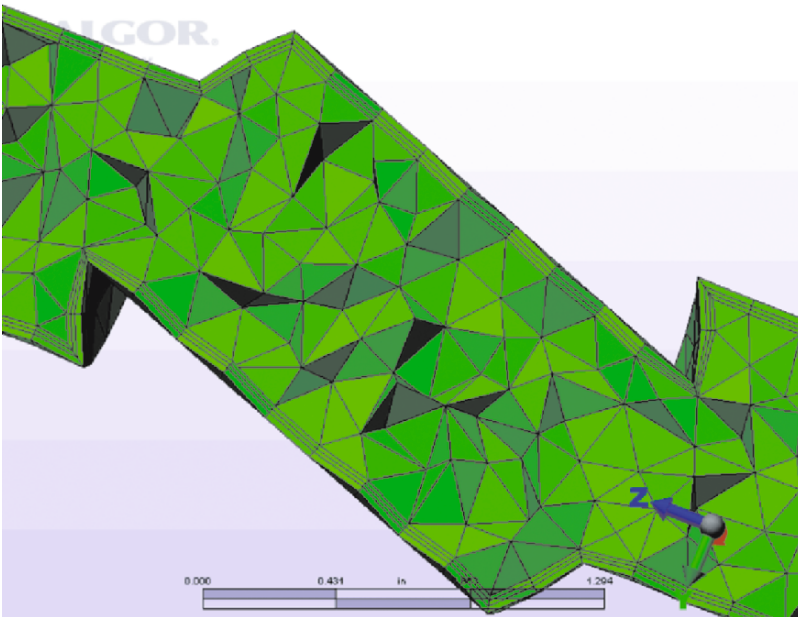


Fig. 9. Final mesh (in the middle section)

Volume Distribution

For validation, the original boundary is decomposed into 12 surfaces, and three layers are generated after propagation. Table 1 illustrates the size of the minimum and maximum volume for prismatic elements at each layer. From this table, we can see that the volume of the prisms is reduced when marching proceeds toward the internal direction.

Figure 10 illustrates partial volume distribution of the prismatic elements. Because the normal directions at the surface mesh are defined toward the

Table 1. Prismatic element volume range

Layer 1		Layer 2		Layer 3	
Min. Size	Max. Size	Min. Size	Max. Size	Min. Size	Max. Size
4.776e-5	4.516e-4	2.374e-5	2.851e-4	1.504e-5	2.701e-4

outward direction of the domain, all the volume values are represented in negative values. In this figure, blue color represents maximum volume, and red represents minimum when absolute values are used. This figure shows that the quality of the surface mesh can greatly affect the quality of the boundary mesh. As a more uniform distribution is generated for the original surface mesh, the quality of the prismatic elements improves. Table 2 shows the overall volume range generated for this model.

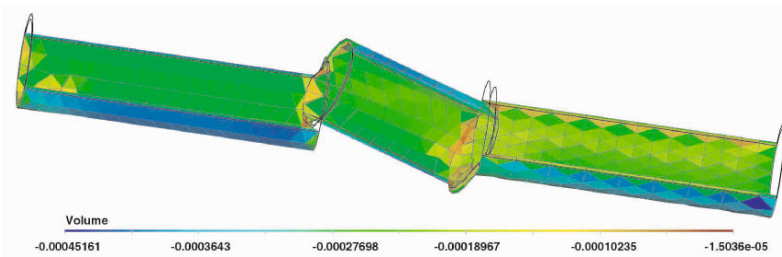

Fig. 10. Volume distribution of prismatic elements

Table 2. Overall volume range

Type	Min. Size	Max. Size
Prismatic element	1.503e-5	4.516e-4
Tetrahedral element	2.904e-5	1.802e-3

Aspect Ratio

The definition of aspect ratio for a prism element is:

$$aspect\ ratio = \frac{averaged\ beam\ length}{averaged\ circumcircle\ radius} \quad (4)$$

Table 3 illustrates the histogram of aspect ratio for prismatic elements. From the table we can see that the averaged aspect ratio is 0.012632. The reason of this boundary mesh has such a small aspect ratio value is that the initial surface mesh is very coarse. Refining the surface mesh size or increasing the offset distance will help to increase the aspect ratio.

Table 3. Histogram of aspect ratio

Range	Number of prisms	%
0.003443 - 0.007660	2241	42.395006
0.007660 - 0.011877	1010	19.107075
0.011877 - 0.016093	562	10.631858
0.016093 - 0.020310	423	8.002270
0.020310 - 0.024527	362	6.848278
0.024527 - 0.028743	293	5.542944
0.028743 - 0.032960	183	3.461975
0.032960 - 0.037177	68	1.286417
0.037177 - 0.041393	64	1.210745
0.041393 - 0.045610	80	1.513432
total	5286	100
Averaged aspect ratio = 0.012623		

Table 4. Histogram of normalized equiangular skewness

Value of normalized equiangular skewness	Number of prisms	%	Cell quality
0.000000 - 0.250000	710	13.431706	Excellent
0.250000 - 0.500000	3827	72.398789	Good
0.500000 - 0.750000	391	7.396897	Fair
0.750000 - 0.900000	332	6.280742	Poor
0.900000 - 1.000000	26	0.491865	Bad
total	5286	100	
Averaged normalized equiangular skewness = 0.16386			

Normalized Equiangular Skewness

The definition of normalized equiangular skewness (NES) and cell quality can be found in [25]. According to [25], 0 indicates the ideal element, 1 indicates the bad element, and the acceptable NES range in 3D is 0-0.4. Table 4 illustrates the histogram of NES. From this table, we can see that the averaged normalized equiangular skewness is 0.16386 which falls in the acceptable range.

Sharp Corner Behavior

Theoretically, this boundary mesh generation method allows the propagation to proceed to any distance without losing original connectivity at sharp corners. However, the cost is that some of the elements may degrade down to zero-volume elements under certain circumstances.

For example, Figure 11 shows the propagation behavior at one sharp corner in two dimensions. The boundary line is propagated toward its inner direction

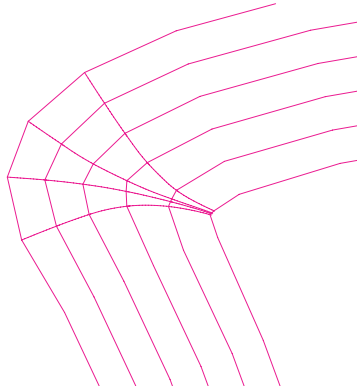


Fig. 11. Propagation behavior at a sharp corner

(from left to right), the propagation paths are skewed together after a certain distance. The post-redistribution algorithm is required to avoid zero-volume elements during the boundary mesh generation process.

Singularity Point Propagation Behavior

Figure 12 shows the propagation behavior at singularity points. The upper figure illustrates the enlarged view of the boundary mesh interface at the part

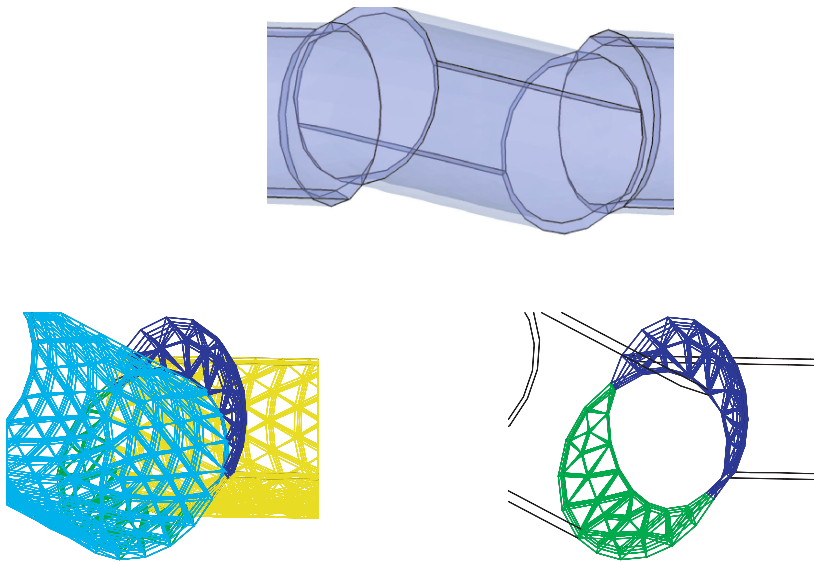


Fig. 12. Mesh at singularity points

with singularity points. The two bottom figures show the mesh at one singularity point. From these figures, we can see that there is only one propagation path at the singularity point, which is shared by the four surfaces' meshes. The reason we can get unique normal directions at the singularity points is because the ϕ value at any point in the computational domain is a unique value. This means the normal vector, $\frac{\nabla\phi}{|\nabla\phi|}$, is a unique value at any point of the computational domain. Thus, in this case, the proposed method can avoid the problem caused by the traditional normal calculation method.

5 Flow Analysis

The flow calculation is performed by ALGOR Professional Multiphysics, which is finite element analysis (FEA) software developed by ALGOR, Inc. Fluid enters the pipe, which is connected to a ball valve in the half-open position. Since the Reynolds number is very low - the viscous force is dominant - a viscous layer on model boundaries is expected in the flow. Two analyses (steady and unsteady flow) are performed using this mesh. Their velocity and pressure analysis results are shown below, respectively. In the illustrated results, dark blue color represents low value, and red represents high value.

Stead Flow

The flow at the inlet is set as a uniform velocity in the direction of the z -axis. The expected velocity solution in the domain is shown in Figures 13 and 14. As predicted, a vortex flow pattern develops in the downstream area. Figure 15 illustrates an enlarged view of Figure 14. Figure 16 is the internal pressure distribution which is coincident with the anticipated result.

Unsteady Flow

In this case, the unsteady solver is applied. Figures 17 and 18 show the velocity result plotted by magnitude and vector, respectively. From these figures, we can see that the final fluid velocity distribution demonstrates a similar behavior as the result calculated by the steady solver when a sufficient physical time is predefined. Figure 19 is an enlarged view of Figure 18 which illustrates the vortex pattern developed in the downstream area. Figure 20 shows the velocity variation at node #2794, which is a singularity point. Its position is illustrated in Figure 8. All the other nodes demonstrate the same convergent behavior. After a certain iteration time, the variation of velocity at any computational node converges to a steady value. Figure 21 illustrates the internal pressure distribution of this model.

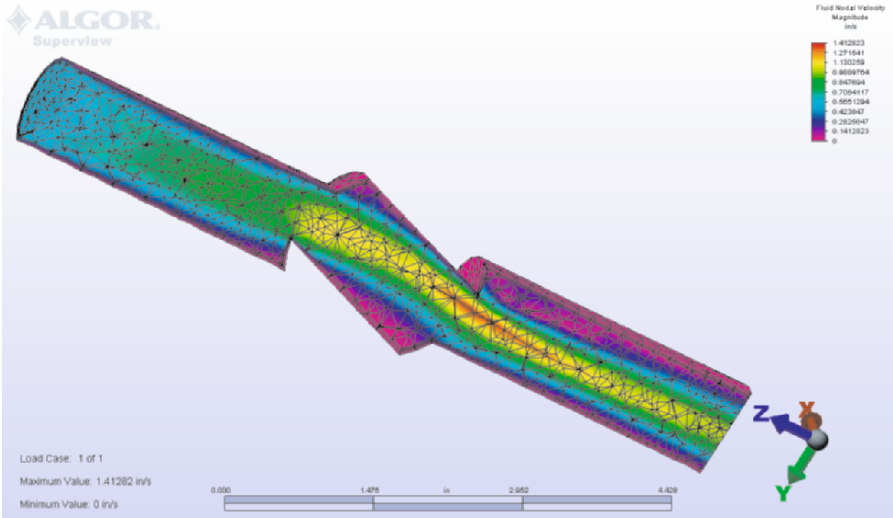


Fig. 13. Velocity distribution (plot by magnitude)

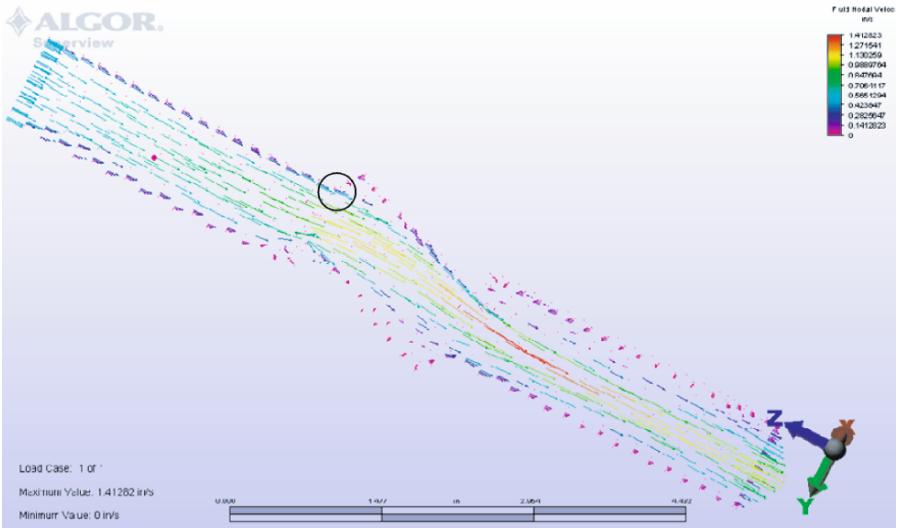


Fig. 14. Velocity distribution (plot by vector)

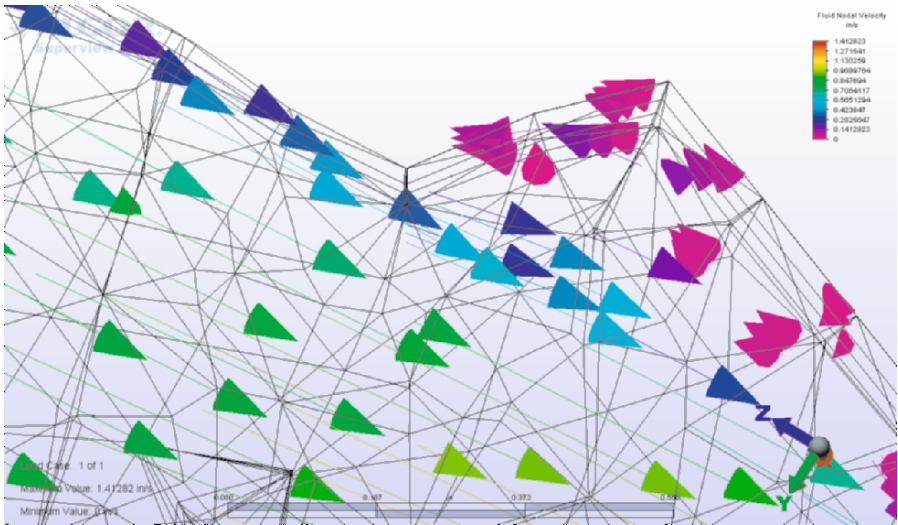


Fig. 15. Vortex pattern (enlarged view)

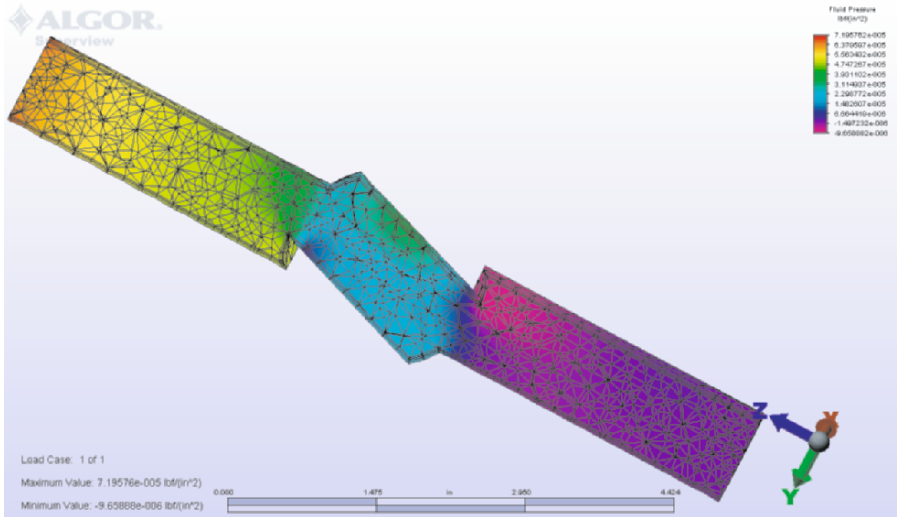


Fig. 16. Pressure distribution

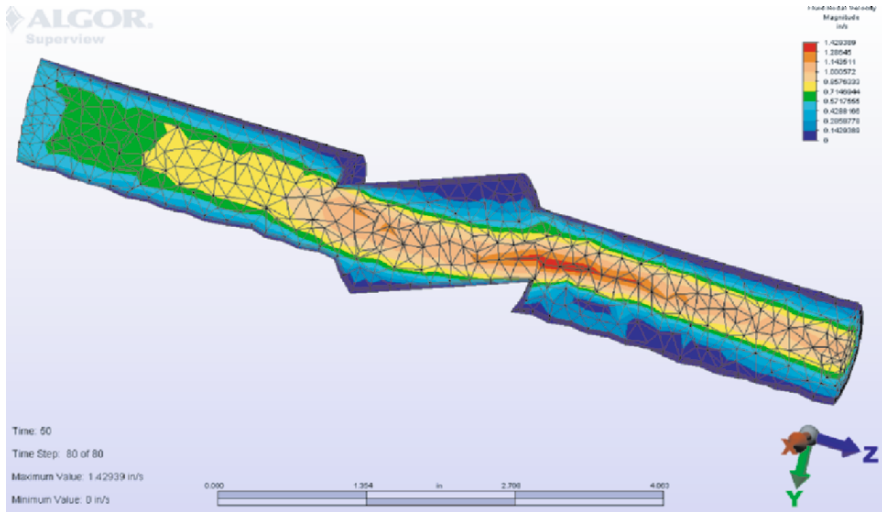


Fig. 17. Velocity distribution (plot by magnitude)

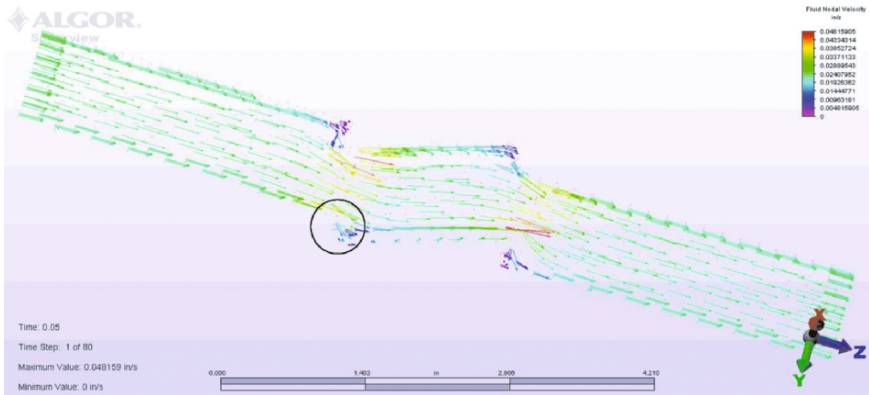


Fig. 18. Velocity distribution (plot by vector)

6 Conclusion

An automated hybrid grid generation method has been developed. The boundary anisotropic elements are extruded along the normal directions which are represented by the weak solution of the Eikonal equation. The method was applied to the ball valve model for both steady and unsteady fluid flow analysis using ALGOR Professional Multiphysics software. The preliminary results show that the proposed method practically generated well-qualified grid distribution for viscous flow.

Further research and development work needs to be done in the following areas: 1) A higher-order numerical solution of the Eikonal equation is to be

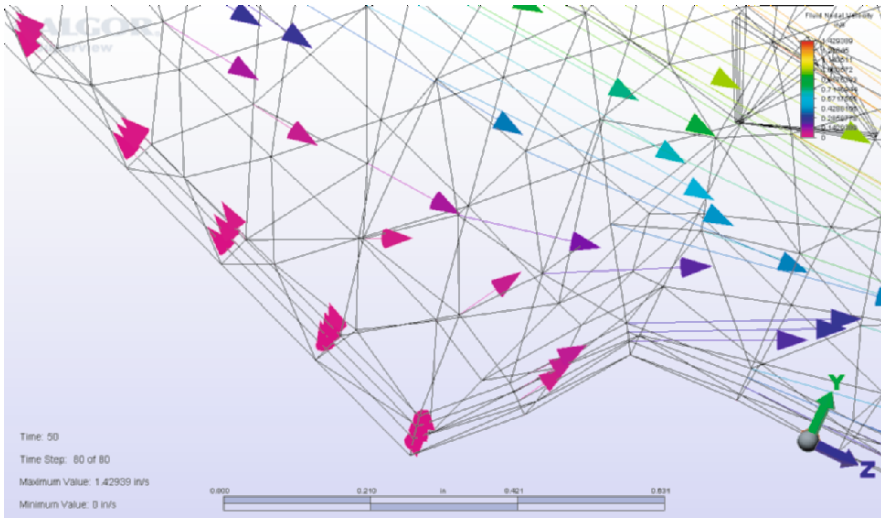


Fig. 19. Vortex pattern (enlarged view)

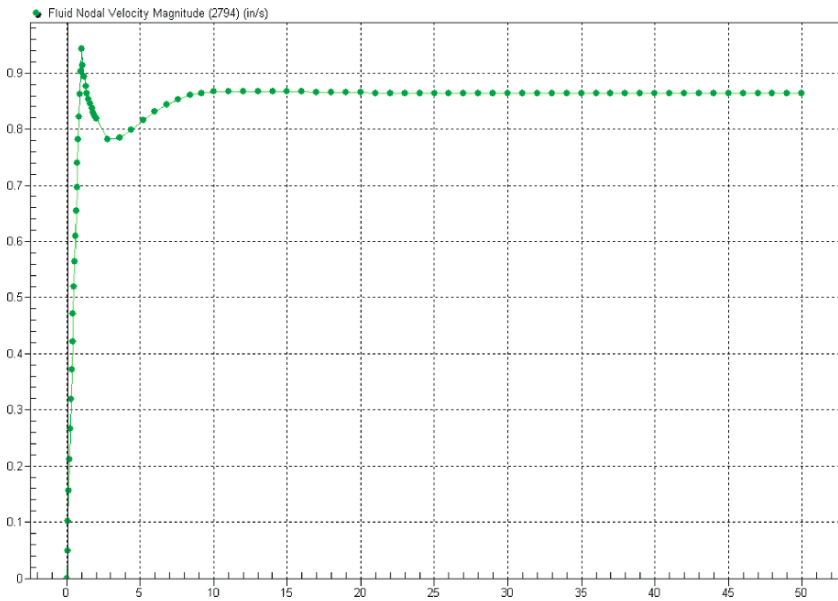


Fig. 20. Velocity variation at node #2794

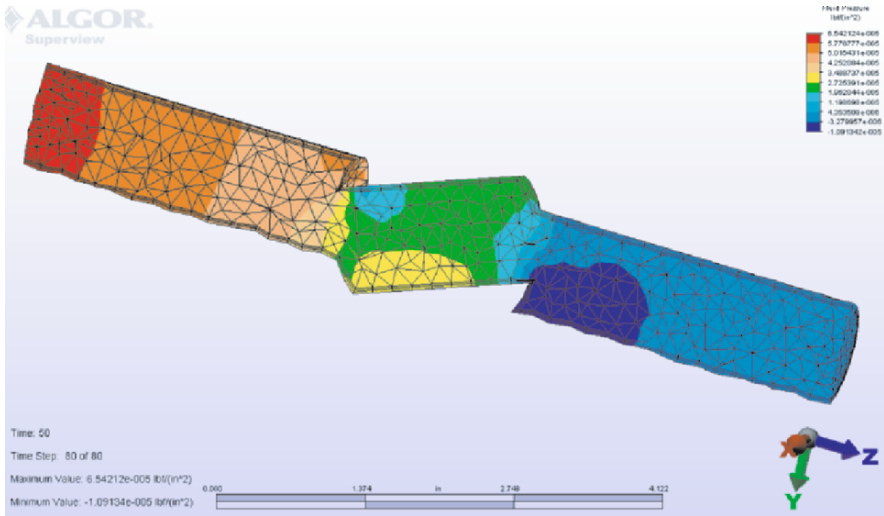


Fig. 21. Pressure distribution

explored. The benefit of doing this is to get more accurate normal calculation which will obviously strengthen control ability during the propagation process; 2) In this work, the propagation stops after three propagations in order to avoid zero-volume elements. Thus, a mesh optimization procedure is to be added as a complementary work of the presented method.

Acknowledgements

Many thanks to Zhongman Ding from ALGOR, Inc. for his discussions, and Minli Chi for his help and suggestions in writing this paper.

References

1. Y. L. Wang, F. Guibault, R. Camarero, and K. -E Tchon, "A parametrization transporting surface offset construction method based on the eikonal equation," in *17th AIAA CFD Conf.*, June 2005.
2. H. K. Zhao, "A fast sweeping method for eikonal equations," *Mathematics of Computation*, vol. 74, no. 250, pp. 603–627, 2005.
3. J. Hoffman, *Numerical Methods for Engineers and Scientists*. McGraw-Hill Inc., 1992.
4. H. Borouchaki, Hecht, F., E. Saltel and P. L. George, "Reasonably Efficient Delaunay Based Mesh Generator in 3 Dimensions," in *Proceedings 4th International Meshing Roundtable*, Sandia National Laboratories, pp. 3–14, 1995
5. Y. L. Wang, F. Guibault, and R. Camarero, "Automatic near-body domain decomposition using the eikonal equation," in *Proceedings of the 14th International Meshing Roundtable*, Sandia National Laboratory, Sept. 2005.

6. Y. L. Wang, F. Guibault, and R. Camarero (2006) *Int. J. Numer. Meth. Engng*, Submitted.
7. P. L. George, F. Hecht, and E. Saltel, "Automatic Mesh Generator with Specified Boundary," in *Computer Methods in Applied Mechanics and Engineering*, North-Holland, Vol 92, pp. 269–288, 1991
8. F. Guibault, *Un mailleur hybride structuré/non structuré en trois dimensions*. PhD thesis, École Polytechnique de Montréal, 1998.
9. S. Pirzadeh, "Unstructured viscous grid generation by advancing-layers method," in *AIAA 11th Applied Aerodynamics Conference*, no. AIAA-93-3453, (Monterey, CA), pp. 420–434, Aug. August 9–11, 1993.
10. J. Sullivan and J. Zhang, "Adaptive mesh generation using a normal offsetting technique," *Finite Elements in Analysis and Design*, vol. 25, pp. 275–295, 1997.
11. J. Glimm, J. Grove, X. Li, and D. Tan, "Robust computational algorithms for dynamic interface tracking in three dimensions," *SIAM J. Sci. Comp.*, vol. 21, no. 6, pp. 2240–2256, 2000.
12. R. Blomgren, "B-spline curves, boeing document, class notes, b-7150-bb-wp-2811d-4412," 1981.
13. W. Tiller and E. Hanson, "Offsets of two-dimensional profiles," *IEEE Computer Graphics and Applications*, vol. 4, no. 9, pp. 36–46, 1984.
14. S. Coquillart, "Computing offsets of b-spline curves," *Comp.-Aided Design*, vol. 19, no. 6, pp. 305–309, 1987.
15. A. Kulczycka and L. Nachman, "Qualitative and quantitative comparisons of b-spline offset surface approximation methods," *Comp.-Aided Design*, vol. 34, pp. 19–26, Jan. 2002.
16. L. A. Piegl and W. Tiller, "Computing offsets of nurbs curves and surfaces," *Comp.-Aided Design*, vol. 31, pp. 147–156, Feb. 1999.
17. Y. F. Sun, A. Y. C. M. Nee, and K. S. Lee, "Modifying free-formed nurbs curves and surfaces for offsetting without local self-intersection," *Comp.-Aided Design*, vol. 36, no. 12, pp. 1161–1169, 2004.
18. R. T. Farouki, "The approximation of non-degenerate offset surfaces," *Comp.-Aided Geom. Design*, vol. 3, no. 1, pp. 15–43, 1986.
19. S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations," *J. Comp. Phys.*, vol. 79, pp. 12–49, 1988.
20. J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 93, no. 4, pp. 1591–1595, 1996.
21. J. A. Sethian, *Level set methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.
22. J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.
23. T. D. Blacker and M. B. Stephenson, "Paving: A new approach to Automated Quadrilateral Mesh Generation," *SIAM Journal on Numerical Analysis*, vol. 32, pp. 811–847, 1991.
24. R. J. Cass and S. E. Benzley and R. J. Meyers and T. D. Blacker, "Generalized 3-D Paving: An Automated Quadrilateral Surface Mesh Generation Algorithm," *SIAM Journal on Numerical Analysis*, vol. 39, no. 9, pp. 1475–1489, 1998.
25. "TGrid 3.5 User's Manual," *Fluent Inc.*, April, 2003.

A Remeshing Procedure for Numerical Simulation of Forming Processes in Three Dimensions

L. Giraud-Moreau, H. Borouchaki and A. Cherouat

Institut Charles Delaunay (FRE CNRS 2848)
University of Technology of Troyes
12 rue Marie Curie BP2060
10010 Troyes Cedex - France
laurence.moreau@utt.fr

Abstract. This article presents a remeshing procedure of thin sheets for numerical simulation of metal forming process in three dimensions. During simulation of metal forming processes, where large plastic deformations are possible, severe mesh distortion occur after a few incremental steps. Hence an automatic mesh generation with remeshing capabilities is essential to carry out the finite element analysis. This paper gives the necessary steps to remesh a structure in finite element simulation of forming processes. The proposed remeshing technique based on geometrical criteria includes adaptive refinement and coarsening procedures. It has been implemented with triangular and quadrilateral elements. The proposed method has been integrated in a computational environment using the ABAQUS solver. Numerical examples show the efficiency of the proposed approach.

Keywords: Adaptive remeshing, forming process, geometrical error estimator, thin sheet.

1 Introduction

The finite element method has been very successful in the numerical simulation of metal forming processes like deep-drawing, hydro-forming or forging [1-2]. However, due to the imposition of large

plastic strains and friction, the finite element mesh representing the workpiece undergoes severe distortion and hence it is necessary to generate a new mesh for the deformed domain [3]. It is therefore convenient to update the mesh in such a way that it conforms to the new deformed geometry and becomes dense enough in the critical region while remaining reasonably coarse in the rest of the domain. The remeshing procedure must be automatic and robust. Several remeshing methods have been proposed during the last years. Globally, three different types of adaptive remeshing strategies can be distinguished: r-adaptivity, p-adaptivity and h-adaptivity [4-5]. Strategies based on r-adaptivity consist of keeping the number of special grid points fixed, but allowing them to move into regions where a finer spatial discretization is needed. This type of adaptation is particularly powerful on problems where a large domain is needed to capture a time varying solution which has steep slopes over only a small fraction of that domain [6]. The remeshing techniques presented by Zienkiewicz et al [7], Fourment et al [8], Coupez [9], Borouchaki et al [10] are based on the computation of a size map to govern a global remeshing of the part at each iteration. Strategies based on p-adaptivity consist of changing the degree of the interpolating polynomials in appropriate regions of the mesh. This method is preferred for (linear) smooth solutions or over subregions where the solution is smooth [11]. Strategies based on h-adaptivity consist of adapting the number of grid points and changing the mesh connectivity. Grid points are added to areas where more accuracy is demanded (the interpolation will be enriched) and can be deleted where the solution is accurate enough. As part of these methods, remeshing techniques based on the computation of a size map to govern a global remeshing of the part at each iteration have been proposed [7-10]. Cho and Yang [12] have proposed a refinement algorithm based on h-adaptivity which consists in splitting each deformed element in two elements along an edge. This procedure drags to the creation of small edges and consequently degenerates

elements during repetitive refinement iterations. Moreover, all similar refinement methods only based on the break of edges lead to the formation of small edges or poor shaped elements.

This paper presents a new remeshing technique for the numerical simulation of thin sheet metal forming processes. This method is based on a geometrical criterion. It is applied to computational domain after each small displacement step of forming tools. It allows, in particular to refine the current mesh of the part under the numerical simulation of the forming process in the curved area with preserving shape quality element and to coarsen this mesh in the flat area.

The mesh refinement is necessary to avoid large element distortions during the deformation. It ensures the convergence of the computation and allows an adequate representation of the geometry of the deformed domain. The mechanical fields are simply induced from the old mesh into the new mesh.

The proposed remeshing method looks like to the remeshing method presented by Meinders [13] in the case of a triangular mesh. Compared to Meinders method, the proposed remeshing technique generates a smaller number of elements, it has been implemented with triangular and quadrilateral elements and a coarsening technique is considered, in addition to the refinement technique.

This paper gives the different steps of the proposed remeshing method. Some application examples are presented in order to show the pertinence of our approach.

2 General Remeshing Scheme

The simulation of the forming process is based on an iterative process. At first, a coarse initial mesh of the part is generated with triangular or quadrilateral elements. At each iteration, a finite element computation is then realized in order to simulate numerically the forming process for a small displacement step of forming tools. This displacement step must be sufficiently small with respect to the specified minimal size of mesh elements.

Then, remeshing is applied after each deformation increment, if necessary, according to the following scheme:

- coarsening procedure applied to elements which are in flat area,
- iterative refinement to restore mesh conformity,
- refinement procedure applied to elements which are in curved area (the refinement is applied in the vicinity of nodes for which the shape of the surface is changed and only if the minimal element size is not reached),
- iterative refinement to restore mesh conformity.

This process (simulation of the forming process for a small displacement step of forming tools, remeshing of the part) is repeated until the final tool displacement is reached.

The computation convergence is principally based on the mesh refinement and coarsening procedures. The applied refinement must in particular not introduce a mesh distortion, which could increase during iterations and stop the forming process simulation.

2.1 Geometrical Criterion

During the remeshing procedure, a geometrical criterion is used to refine the current mesh of the part in the curved area, and to coarsen this mesh in the flat area. For a given element, this geometrical criterion represents the maximal angular gap between the normal to the element and the normals at its vertices. An element is thus considered to be “curved” (resp. “flat”) if the corresponding angular gap is greater (resp. smaller) than a given threshold (for example 8 degrees). The geometrical refinement and coarsening methods based on the same geometrical criterion are thus consistent.

The normal vector \vec{v} at node P can be defined as the weighted average of the unit normal vectors \vec{N}_i ($i = 1, \dots, m$) to elements sharing node P:

$$\vec{v} = \frac{\sum_{i=0}^m \alpha_i \vec{N}_i}{\left\| \sum_{i=0}^m \alpha_i \vec{N}_i \right\|} \quad (1)$$

where α_i is the angle at P of the i th element sharing P.

The computation of normal vector to the element depends on the element shape (triangle or quadrilateral). The normal vector \vec{N} to a triangle $P_1P_2P_3$ is the unit normal vector to its supporting plane :

$$\vec{N} = \frac{\overrightarrow{P_1P_2} \wedge \overrightarrow{P_2P_3}}{\left\| \overrightarrow{P_1P_2} \wedge \overrightarrow{P_2P_3} \right\|} \quad (2)$$

The normal vector to a quadrilateral element is the average of the normal vectors to the four triangles defined by joining its barycentre to its edges.

The geometrical criterion applied to an element can be written as:

$$\max_i (\arccos \langle \vec{v}_i, \vec{N} \rangle) \geq \omega_g \quad (3)$$

Where \vec{v}_i is the normal at vertex i of the element, \vec{N} is its normal and ω_g is an angular gap threshold. In this case, the element must be refined.

2.2 Mesh Refinement and Coarsening Methods

The adaptive remeshing technique consists in improving the mesh in order to conform to the geometry of the current part surface during deformation. In the following, the mesh refinement and coarsening methods are detailed.

The refinement technique consists in subdividing mesh elements. An element is refined if it is a “curved” element (geometrical criterion). There is only one element subdivision which allows to preserve the element shape quality: the uniform subdivision into four new elements. In the case of a triangle, three new nodes are added : one in the middle of each edge. In the case of a quadrilateral, five nodes are added : one in the middle of each edge and one in the element barycentre. Figure 1 shows the triangular and quadrilateral element refinements.



Fig. 1. Triangular and quadrilateral element refinements

After each refinement procedure, an iterative refinement to restore mesh conformity is necessary. Indeed, after applying the subdivision according to the geometrical criterion, adjacent elements to subdivided elements must be modified. As the edges of the subdivided elements are divided in two, there is a node in the middle of the edges common to the subdivided element and its adjacent elements. The mesh is then not conforming. To retrieve the mesh conformity, adjacent elements to subdivided elements must be also subdivided. This last subdivision can not be a homothetic subdivision in four elements because it would result in the systematic homothetic subdivision of all mesh elements.

There are three different configurations for adjacent elements which must be subdivided in order to ensure the mesh conformity:

- no edge is saturated (i.e. containing a new added node),
- only one edge is saturated,
- at least two edges are saturated.



Fig. 2. Subdivision of elements with one saturated edge

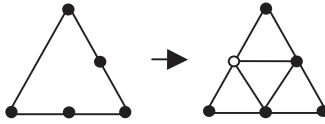


Fig. 3. Subdivision of triangle with two saturated edges

Depending on the configuration, a subdivision is applied if necessary. In the first case (no saturated edge), the element is not subdivided and is not modified. In the second case (one saturated edge), a triangular element is subdivided in two triangles and a quadrilateral element in three triangles (see figure 2). This subdivision allows to stop the propagation of the homothetic subdivision. In the third case, if all the edges are saturated the element is subdivided in four homothetic elements. Otherwise, in the case of triangular elements (having two saturated edges), all possible subdivisions lead to the formation of poor shaped elements (stretched elements). It is then necessary to add a new node in order to subdivide also this element into four homothetic elements (see figure 3). In the quadrilateral case, when only two edges are saturated and are adjacent, the quadrilateral is subdivided in four triangles (see figure 4). This subdivision allows to stop the propagation of the homothetic subdivisions. In the other cases, the element is subdividing into four homothetic quadrilateral elements (see figure 5) by adding nodes in the middle of non-saturated edges and in the barycentre of the element.

This refinement procedure is iteratively applied until no new node is added.

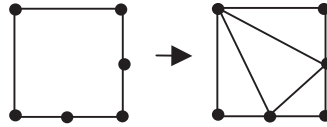


Fig. 4. Subdivision of quadrilateral element with two adjacent saturated edges

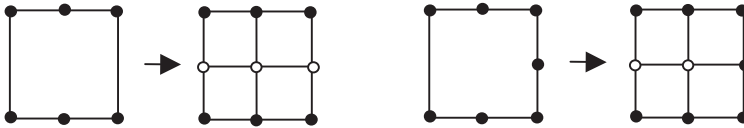


Fig. 5. Other cases of subdivision of quadrilateral element

From an algorithmic point of view, the mesh is composed of two types of element: ordinary and extraordinary. An ordinary element is a triangle or a quadrilateral without saturated edges (see figure 6). An extraordinary triangle is a triangle with one and only one saturated edge. An extraordinary quadrilateral is a quadrilateral with only one saturated edge or two adjacent saturated edges. Figure 7 shows extraordinary triangle and quadrilaterals. The remeshing algorithm must take into account these two element types. During the refinement operation, the geometrical criterion is applied to elements of both types. An ordinary or extraordinary element which is curved is then subdivided into four ordinary elements. After this operation, all ordinary elements with at least two saturated edges, except the case of two adjacent edges for quadrilateral elements, are iteratively subdivided into four ordinary elements. Then, all the elements with at least one saturated edge are transformed to extraordinary elements and the other elements remain unchanged.

At the end of the refinement operation, for the mechanical computational purpose, the extraordinary elements of the resulting mesh are transformed : an extraordinary triangle is divided in two triangles, an extraordinary quadrilateral with one middle node is divided in three triangles and an extraordinary quadrilateral with two middle nodes is divided in four triangles.



Fig. 6. Ordinary elements

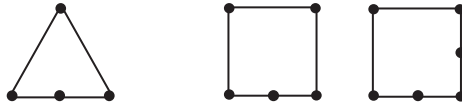


Fig. 7. Extraordinary elements

The coarsening technique is the reciprocal operation of the refinement procedure. It can only be applied to a set of four ordinary elements, obtained during a homothetic element refinement. Thanks to the coarsening technique, the initial element is restored when the area in which this element belongs, becomes flat (see figure 8).



Fig. 8. Triangular and quadrilateral elements coarsening

A quad tree structure can be considered to coarsening the mesh of the part. This structure allows to quickly localize associated elements. Each root of the tree is an element of the initial mesh and each edge is an intermediate element created during the refinement procedures. The leaves of the tree are elements of the current mesh and are brought together if they are associated. Figure 10 presents the quad tree structure associated to the mesh of the part on figure 9 at iteration 3. Elements whose number is underlined, are extraordinary elements.

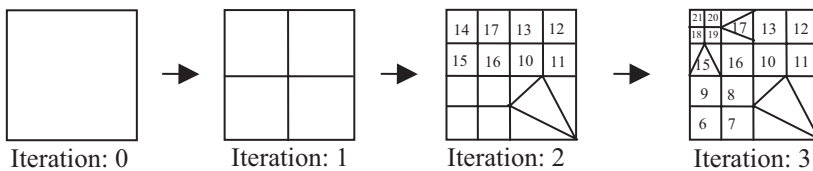


Fig. 9. Example of adaptive remeshing during three iterations

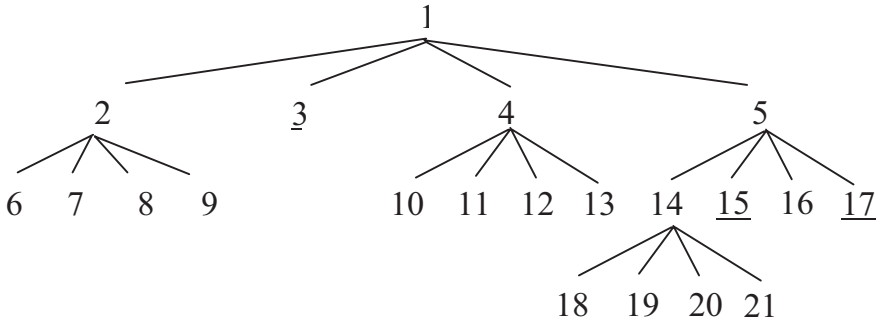


Fig. 10. Quad tree structure for the mesh of the part of figure 9

The coarsening technique is only applied to the leaves of a same level which are brought together. In the above example, associated elements which could be coarsened are: (6, 7, 8, 9) or (10, 11, 12, 13) or (18, 19, 20, 21). As in the refinement procedure, to ensure the mesh conformity, some coarsened elements could be refined if necessary. This last operation can only be applied when all the flat areas have been coarsened by the coarsening technique.

2.3 Transfer of Mechanical Fields

During the refinement procedure, the mechanical fields are simply induced from the current mesh to the new mesh. During the coarsening procedure, the mechanical fields associated to four associated elements are averaged and the result is associated to the new element. During the refinement procedure, the mechanical fields of curved elements are simply associated to the four new created elements from the subdivision.

3 Numerical Examples

3.1 Sheet Metal Stamping

An early application of adaptive mesh refinement was the simulation of 3-D sheet metal stamping example (Benchmark square box of Numisheet'93). According to Onate et al. [14], the geometric data of the square cup are: drawing depth 40 mm, sheet dimensions $150 \times 150 \text{ mm}^2$, thickness $h^0 = 0.78 \text{ mm}$, friction coefficient between the sheet and rigid tools is assumed to be $\mu = 0.144$ and the blank-holder force $F = 19600 \text{ N}$. The material model used is an isotropic elastoplastic von-Mises model with multi-linear isotropic hardening approximating a power law yield stress curve defined as $\sigma = 567.29(0.07127 + \bar{\epsilon}^p)^{0.2637}$. The punch velocity is 20 mm/s and its stroke is 80 mm. The tools (punch, die and blank-holder) are supposed rigid and modeled by discrete rigid surfaces. Two examples are presented: the first example concerns the stamping of square sheet in which the angle θ between initial sheet plane frame (X,Y) and the tools orientation (x,y,z) is $\theta = 0^\circ$ (see Figure 11a) and the second concerns the stamping of square sheet with $\theta = 45^\circ$ (see Figure 11b). In these two cases, the solver 3D ABAQUS/EXPLICIT has been used. The element size adaptive discretization of the deformable sheet uses $h_{\min} = 0.75 \text{ mm}$, geometrical criterion = 8° .

Meshes adapted to the part curvature corresponding to different punch displacement ($u = 6, 15, 24, 30, 36, 45$ and 48 mm) are shown in Figures 12 to 18 for $\theta = 0^\circ$ and $\theta = 45^\circ$. We can note that, the initial blank sheet is computed using an initial coarse mesh (100 quadrilateral elements), the mesh is again refined uniformly and the adaptive mesh refinement procedure is activated where elements are created automatically in regions of large curvature to even more accurately represent the complex material flow (large stretching) around the punch and die radii. The final contour of the sheet for 60 mm of punch displacement is presented in Figure 22 for $\theta = 0^\circ$ and in Figure 23 for $\theta = 45^\circ$. We can note the final shape of the sheet is completely different due the initial sheet orientation.

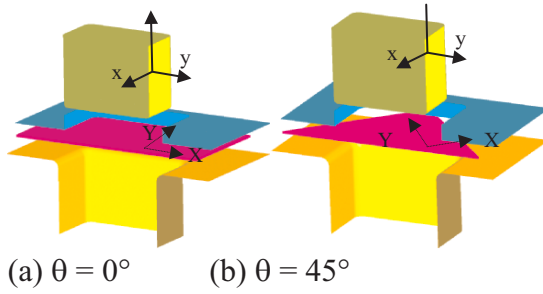


Fig. 11. Tools and initial sheet orientation

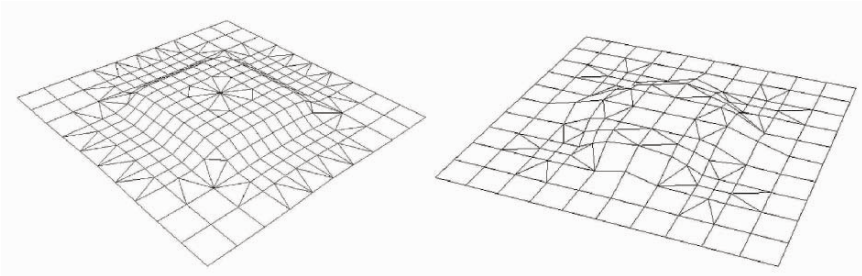


Fig. 12. Displacement $u = 6$ mm

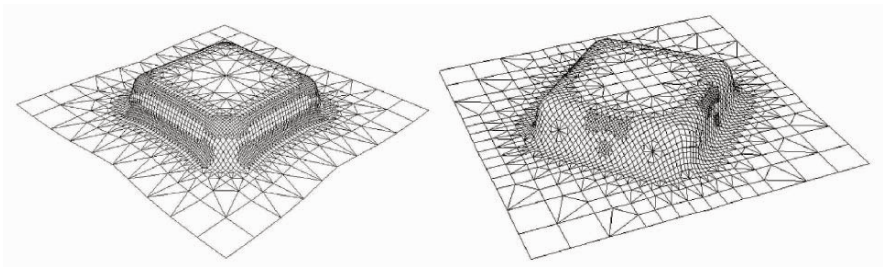


Fig. 13. Displacement $u = 15$ mm

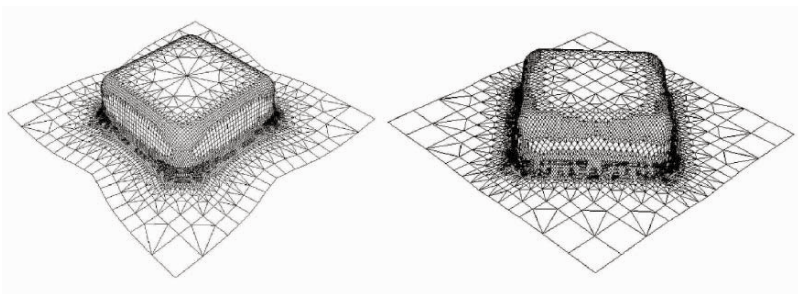


Fig. 14. Displacement $u = 24$ mm

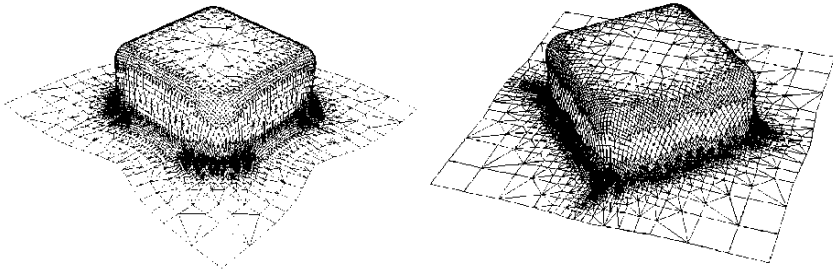


Fig. 15. Displacement $u = 30$ mm

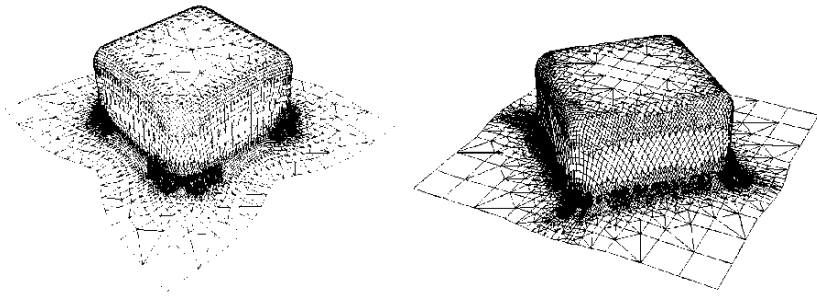


Fig. 16. Displacement $u = 36$ mm

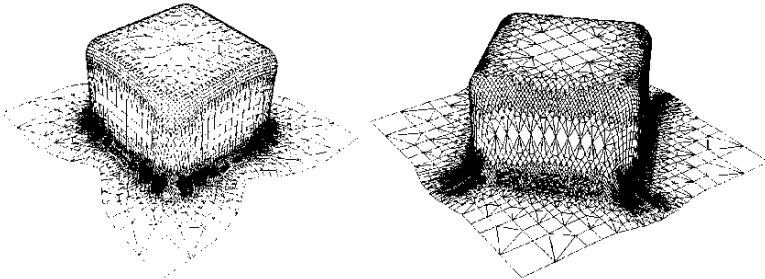


Fig. 17. Displacement $u = 45$ mm

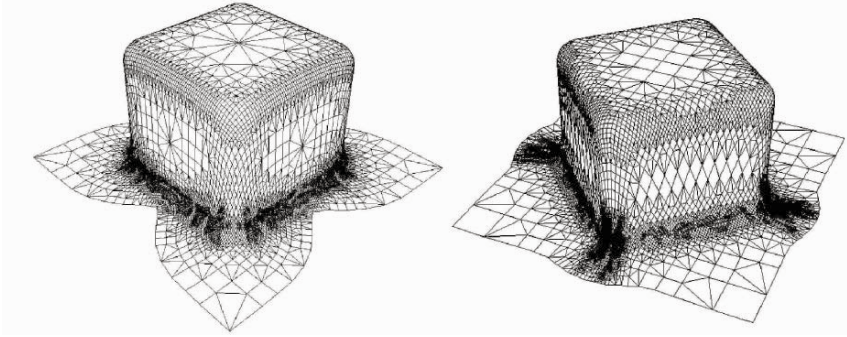


Fig. 18. Displacement $u = 48$ mm

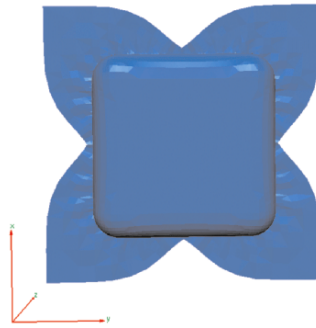


Fig. 19. Final shape for $\theta = 0^\circ$

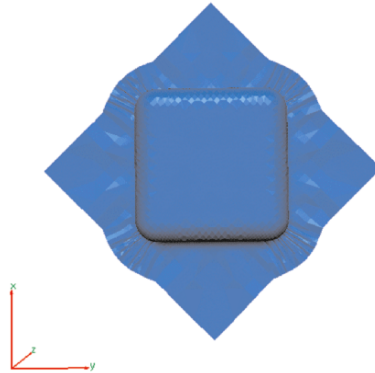
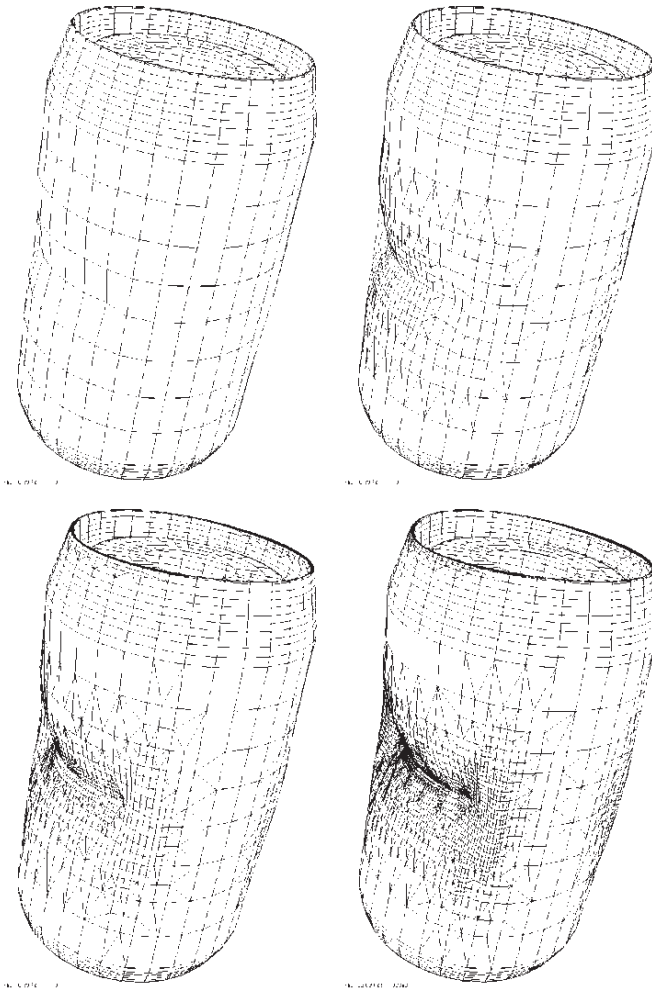


Fig. 20. Final shape for $\theta = 45^\circ$

3.2 Crushing of a Thin Cylinder

The second example is the crushing of a thin cylinder. The cylinder blank has initially 140 mm length, 44 mm diameter and 0.5 mm thickness. The initial mesh of the cylinder is constituted by 2048 quadrilateral sheet finite elements. Two concentrate loads diametrically opposite was prescribed using a linear ramp to simulate the crushing operation. The deformation evolution of the blank is illustrated in Figure 21. Here, the mesh refinement is localized on large deformed blank areas. The final mesh of the blank contains 30301 quadrilateral and 26714 triangular elements.



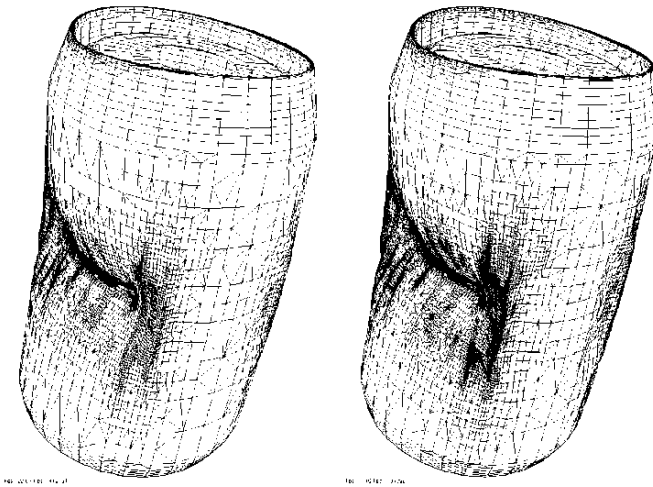


Fig. 21. Deformed cylinder for different crushing step

4 Conclusions

The different steps necessary to the remeshing of the computation domain in large elastoplastic deformations in three dimensions have been presented. The proposed adaptive remeshing technique is based on refinement and coarsening procedures using geometrical criterion. This approach has been implemented with triangular and quadrilateral elements in the ABAQUS code. Numerical simulations of thin sheet metal forming process in three dimensions have validated the proposed approach and proved its efficiency. The extension in three dimensions for massive structure metal forming is currently under progress.

References

1. H. Borouchaki, A. Cherouat, K. Saanouni, A. Cherouat and P. Laug, “Remaillage en grandes deformation. Applications à la mise en forme de structures 2D”, *Revue Européenne des éléments finis* (to appear), 2002.

2. K. Saanouni, A. Cherouat and Y. Hammi, "Optimization of hydroforming processes with respect to fracture", Esaform, Belgium, 1, 361-364, 2001.
3. P. Ladeveze and J.P. Pelle, "La maîtrise du calcul en mécanique linéaire et non linéaire, études en mécanique des matériaux et des structures", Hermès, Paris, France, 2001.
4. D. Djokovic, "Splines for approximating solutions of partial differential equations", Ph.D. Thesis, University of Twente, Enschede, ISBN 90-36510511, 1998.
5. A. Huerta, P. Diez, A. Rodriguez-Ferran, "Adaptivity and error estimation", Proceedings of the 6th International Conference on Numerical Methods in Industrial Forming Processes, J. Huétink and F.P.T. Baaijens (eds), Balkema, Rotterdam, 63-74, 1998.
6. R. Drake, V.S. Manoranjan, "A method of dynamic mesh adaptation", Int. J. Num. Meth. Eng., 39, 939-949, 1996.
7. O. C. Zienkiewicz and J. Z. Zhu, "Adaptivity and mesh generation", in Int. J. Numer. Methods Eng. 32, 783-810, 1991.
8. L. Fourment and J.-L. Chenot, "Adaptive remeshing and error control for forming processes", Revue européenne des éléments finis 3, 2, 247-279, 1994.
9. T. Coupez, "Génération de maillage et adaptation de maillage par optimisation locale", Revue européenne des éléments finis 9, 4, 403-422, 2000.
10. H. Borouchaki, A. Cherouat, P. Laug and K. Saanouni, "Adaptive remeshing for ductile fracture prediction in metal forming", C.R. Mecanique 330, 709-716, 2002.
11. Y. Li, I. Babuska, "A convergence analysis of a p-version finite element method for one-dimensional elastoplasticity problem with constitutive laws based on the gauge function method", J. Num. Anal., 33, 2, 809-842, 1996.
12. J.-W. Cho and D.-Y. Yang, "A mesh refinement scheme for sheet metal forming analysis", Proc. of the 5th International Conference, NUMISHEET'02, 307-312, 2002.
13. T. Meinders, "Developments in numerical simulations of the real life deep drawing process, Ph.D. Thesis, University of Twente, Enschede, ISBN 90-36514002, 2000.
14. E. Onate, J. Rojek, C. Garcia Garino, Numistamp: "A research project for assessment of finite element models for stamping process", Journal of Materials Processing Technology 50, 17-38, 1995.

Mesh Adaptation

A Solution-Based Adaptive Redistribution Method for Unstructured Meshes

Yasushi Ito, Alan M. Shih, Roy P. Koomullil and Bharat K. Soni

Dept of Mechanical Engineering
University of Alabama at Birmingham, Birmingham, AL, U.S.A.
{yito, ashih, rkoomul, bsoni}@uab.edu

Abstract. We propose an unstructured mesh redistribution method without using skewed elements for steady-state problems. The regions around solution features are indicated by a sensor function. The medial axes of the strong feature regions are calculated so that elements can be clustered around the most important solution features efficiently. Two approaches, a discrete surface-based approach using a Delaunay triangulation method and a mathematical-representation approach using least square fitting, are shown to calculate the medial axes. Remeshing of an initial volume mesh is performed around the medial axes using an advancing front method and/or an advancing layer method. Two examples are shown to present how our approach works.

Key words: Solution-based; mesh redistribution; remeshing; unstructured mesh

1. Introduction

Computational fluid dynamics (CFD) has become a crucial tool for prediction and analysis of flow field within a domain, providing engineers with a reliable means of understanding complex flow patterns. However, in order to obtain accurate results for highly complex flow fields, meshes must be clustered near the areas where the solution gradients are high. This is an arduous task the engineer must perform prior to the completion of the calculation. The meshes can be clustered in two ways; either a very fine

mesh is generated, or some solution-based mesh clustering is performed. The first approach can be very expensive in terms of computational costs. Although surface meshes can be adapted geometrically based on surface curvature and local volume thickness [1], it is often difficult to choose adaptation criteria for volume meshes before numerical simulations. The second approach can be achieved by mesh adaptation.

There are three mesh adaptation approaches: mesh refinement/de-refinement [2, 3], mesh redistribution [4] and the combination of these [5, 6]. Since structured meshes are not flexible for adding or deleting nodes locally, the mesh redistribution approach is widely used to move nodes toward solution features while the connectivity of the mesh is maintained.

Although solution features are adapted by unstructured meshes relatively easily, there are two issues needed to be addressed. One is the maintenance of valid elements. Hanging nodes can be created during a mesh refinement process. Local refinement of hybrid meshes for viscous flow simulations, which contain regular elements such as tetrahedra, prisms, pyramids and hexahedra, is difficult without creating low-quality elements to eliminate hanging nodes. To overcome this issue, an approach using generalized elements is promising [3].

The other issue is the quality of resulting refined meshes. Stretched elements may affect solution accuracy and cause a stiffness problem in numerical simulations. Mavriplis reports spanwise grid stretching, which is widely used in aircraft CFD simulations, may have substantial repercussions on overall simulation accuracy even at very high levels of resolution [7]. Since typical refinement and redistribution algorithms for unstructured meshes create highly stretched tetrahedra around solution features, the validation of the simulation process may be required. If a refined mesh does not have elements that have too small or too large angles even near solution features, we do not need to worry about these issues.

In this paper, we propose a solution-based redistribution method for unstructured volume meshes. The structured mesh redistribution methods only allow nodes to move towards solution features, while maintaining the mesh connectivity. In our unstructured mesh redistribution method, a mesh is remeshed around the solution features detected. The main objective here is to extract strong solution features as smooth surfaces (Section 2.3) based on sensor values (Section 2.2) and then to create high quality elements around them (Section 2.4). The entire domain can be remeshed with the embedded surfaces using an advancing front method with tetrahedra and an advancing layer method with prisms or hexahedra if needed. Alternatively, elements around the feature surfaces are removed from the initial volume mesh and only the resulting voids are remeshed to reduce

the required CPU time. Two examples are shown to present how our approach works (Section 4).

2. Meshing Methods

In this section, the methods for creating an initial mesh and redistributing nodes around solution feature surfaces are described.

2.1. Surface/Volume Mesh Generation

To generate surface meshes based on computer-aided design (CAD) defined geometries, a direct advancing front method is employed [8]. A modified decimation method is used for image-based geometries [1].

Tetrahedral meshes are created using an advancing front method [9]. For viscous flow simulations, a modified advancing layer method is used for the near-field mesh generation [10], which is followed by tetrahedral mesh generation to fill the rest of the domain using the advancing front method. The quality of the tetrahedral elements are enhanced using angle-based node smoothing, face swapping based on the Delaunay property, and removal of nodes that have an insufficient number of tetrahedra. A user can specify a stretching factor to control the mesh density.

The hybrid mesh generation method can be used to create layered meshes on solution features discussed in the next section to create high quality anisotropic adaptive meshes.

2.2. Feature Detection

After a numerical simulation using an initial mesh, the next step is the detection of solution features. The location of solution features is indicated by the weight function by Soni *et al.* [11] or the shock sensor by Lovely and Haines [12]. The weight function is calculated based on the conserved variables and indicates the regions of important flow features. It is defined at each element as follows:

$$\begin{aligned}
 W &= 1 + \frac{W^1}{\max(W^1, W^2, W^3)} \oplus \frac{W^2}{\max(W^1, W^2, W^3)} \oplus \frac{W^3}{\max(W^1, W^2, W^3)} \\
 W^k &= \sum_{\oplus i=1}^{nq} \left[\frac{q_{\xi k}^i}{|q^i| + \varepsilon} \Bigg/ \left[\frac{q_{\xi k}^i}{|q^i| + \varepsilon} \right]_{\max} \right] \quad (k = 1, 2, 3)
 \end{aligned} \tag{1}$$

where W^k ($k = 1, 2, 3$), $q_{\xi k}^i$ and q^i are x , y and z components of normalized gradient, the k^{th} component of the gradient calculated using i^{th} variable and the average variable at the centroid of the element, respectively. The symbol \oplus represents the Boolean sum, which, for two variables q_1 and q_2 , is defined as

$$q_1 \oplus q_2 = q_1 + q_2 - q_1 q_2 \quad (2)$$

The shock sensor is based on the fact that the normalized Mach number $M_n = 1$ at a shock.

$$M_n = \frac{\mathbf{V} \cdot \nabla p}{a |\nabla p|} = 1 \quad (3)$$

where a , \mathbf{V} and ∇p are the speed of sound, velocity vector and pressure gradient, respectively.

2.3. Extraction of Solution Feature Surfaces

To adapt high-quality elements around strong solution features, the next step is recognition of feature surfaces. Although this approach may need more meshing steps than a typical mesh redistribution method, much better quality elements can be generated around the solution feature surfaces. Marcum and Gaither propose a pattern recognition algorithm in 2D and mention the difficulty of extending it to 3D [13]. Although our approach needs user interaction during the process (to be discussed in Section 4), it enables feature surface extraction.

The direct extraction of solution feature surfaces is difficult from the initial mesh and solution data. At least two steps are needed. First, regions around the solution features are specified by selecting a certain sensor value. Although elements can be subdivided in the entire regions, the number of elements in the resulting mesh may become too big. The regions can be very thick if an initial volume mesh is coarse at the solution feature locations. To avoid this problem, the medial axis (also known as skeleton) of each region is extracted in the following step. Elements are clustered around the medial axes.

Two approaches can be considered to extract solution feature surfaces. One is a discrete surface-based approach. A medial axis is extracted from a triangulated closed surface using Delaunay triangulation [14]. Triangulated isosurfaces at a certain sensor value can be calculated easily and robustly, which enclose regions around solution features. For example, the shock features are surrounded by the isosurfaces at $M_n = 1$. A Delaunay

tetrahedral mesh can be obtained from a triangulated isosurface. The center of the circumsphere of each tetrahedron is considered to represent the medial axis. The quality of the resulting medial axes depends on the smoothness of the isosurfaces. However, isosurfaces are not usually smooth, and they may have bumps and holes due to truncation errors in the entire simulation process. User interaction is often required to fix the resulting surfaces.

The other approach is a mathematical-representation approach. A medial axis can be estimated using least square fitting directly from the nodes on an isosurface. Least square fitting methods often minimize the vertical offsets from a surface function instead of the perpendicular offsets to simplify an analytic form for the fitting parameters. Consequently, the least square fitting does not estimate the surface function well when the region defined by an isosurface is thick. Although a set of coordinates of nodes near a solution feature is needed as an input for a least square fitting method, the connectivity of the nodes is not required. Therefore, we define a solution feature as a set of nodes based on the following process:

1. Select nodes of a volume mesh that have a certain range of sensor values.
2. Also select nodes that are one-ring neighbors of the nodes in Step 1 to eliminate noise due to truncation errors.
3. Number each cluster of selected nodes, which can be defined as their connectivity, if the mesh has more than one solution features.
4. Calculate distance from the closest boundary at each selected node. The boundary is represented by the selected nodes that have at least one unselected node as their one-ring neighbor. The distance is defined as the number of edges from the boundary.
5. The nodes that have local maxima of the distance values are considered to form medial axes.

The coordinates of the nodes in Step 5 are fitted to functions, such as a plane, quadric and cone, using a least square fitting method. Local mesh size can be considered to be the error range of a data point. The reciprocal of the local mesh size is used for weighing. Suppose that a cluster of selected nodes \mathbf{x}_{m_j} ($j = 1, 2, \dots, n_m$) is fitted to a function $z = f(x, y)$.

$$E \equiv \sum_{j=1}^{n_m} \left(\frac{z_j - f(x_j, y_j)}{l_j} \right)^2 \quad (4)$$

where l_j is the maximum edge length connected to node j . E should be minimized.

The resulting function should be trimmed to define a surface in the computational domain.

2.4. Remeshing with Embedded Surfaces

Next step is to remesh the initial volume mesh (defined as V) around the medial axes created in the previous section. First, surface meshes (defined as S) are created on the medial axes based on user-specified mesh size. Second, elements of V are removed if they are close to S . Each node of S , S_i , should be in an element of V , V_i . A node of V , V_j , is removed if it is in a sphere defined at node S_i ($i = 1$ to the number of nodes in S) with the radius of $3 \max(l_{si}, l_{vj})$, where l_{si} is the maximum length of the edges connected to S_i and l_{vj} is the maximum length of the edges in V_j and if V_j is visible from S_i (*i.e.*, there is no boundary surface of V between V_j and S_i). Third, the void regions around the medial axes are filled using the meshing method described in Section 2.1.

The entire domain is remeshed with the embedded surfaces if a layered mesh is needed for no-slip walls or the shape of the outer boundary needs to be changed.

3. Flow Solver

The flow simulation system that is used for the current study is developed for a generalized grid framework, in which the discretization of the physical domain can be of structured, unstructured or an agglomeration of cells with an arbitrary number of faces (polytops) [15, 16]. The integral form of the Navier-Stokes equations is taken as the governing equations for the fluid flow. The spatial discretization of the governing equations is based on a cell-centered, finite volume upwind scheme. The convective fluxes at the cell-faces are evaluated using Roe's approximate Riemann [17]. Higher-order accuracy in the spatial domain is achieved using a Taylor series expansion of flow variables. The gradients at the cell center for the Taylor series expansion is estimated using either the Gauss theorem together with a weighted averaging procedure or a least-square fit of the variables in the neighboring cells. The least-square system resulting from the later approach is solved using the Gram-Schmidt method. A limiter function is added to the Taylor's series expansion to avoid the creation of local extrema during the reconstruction process. Limiters by Venkatakrisnan [18], and Barth and Jespersion [19] are implemented in the generalized grid framework.

4. Applications

In this section, two examples are shown to demonstrate and to discuss the solution-based mesh redistribution approach.

4.1. NACA0012 Wing

Figure 1a shows a mesh around a NACA0012 wing (50k nodes). An inviscid flow simulation is carried out at a freestream Mach number of 0.799 and an angle of attack, α , of 2.26° . Figures 1b and 1c illustrate pressure coefficient (C_p) distribution and weight function value distribution based on Eq. 1; the results indicate a shock on the wing.

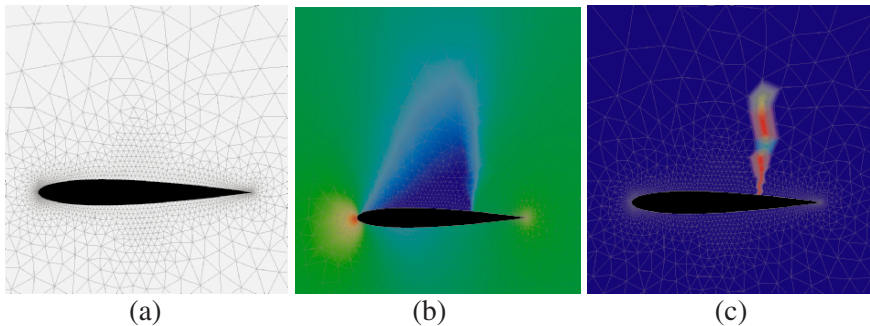


Figure 1. NACA0012 wing: (a) a tetrahedral mesh; (b) C_p distribution ($M = 0.799$, $\alpha = 2.26^\circ$); (c) weight function value distribution.

In this example, the shock location is estimated using the discrete surface-based approach based on Delaunay triangulation (Figure 2). First, an isosurface of a weight function value of 0.2 is extracted and smoothed using Visualization Toolkit (VTK) [20] (Figure 2a). Second, the medial axis of the isosurface is calculated (Figure 2b). The symmetry planes prevent creating a single medial axis. The medial axis is modified there manually (Figure 2c). It is sometimes difficult to generate an expected medial axis as single surface using existing algorithms even if a solution feature is simple.

Once the feature surface is computed, the surface mesh generation algorithm is applied to create a high quality mesh on it. Elements of the initial volume mesh, V^0 , near the solution feature are removed and the void is remeshed using the advancing front method. Figure 3b shows the resulting volume mesh (110k nodes; V^1). As a result, a high quality redistributed mesh is produced with alignment to the major flow feature. Figure 3c shows another redistributed volume mesh after the second simulation cycle (130k nodes; V^2).

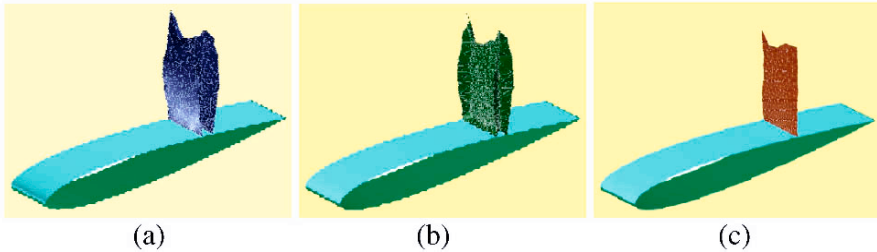


Figure 2. Extraction of a flow feature: (a) isosurface of a weight function value of 0.2 at the shock location; (b) medial axis of the isosurface; and (c) modified medial axis.

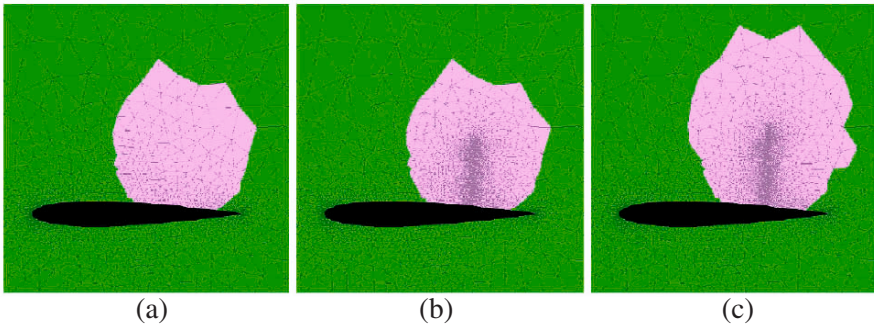


Figure 3. Redistributed volume meshes: (a) initial mesh (50k nodes); (b) redistribution #1 (110k nodes) – elements around the shock are replaced with finer elements; (c) redistribution #2 (130k nodes).

Figure 4 illustrates hybrid meshes for the same wing geometry to perform viscous flow simulations and C_p distribution. The shock location is estimated using the same approach from the initial hybrid mesh (Figure 4a), and then the entire domain is remeshed with the embedded surface (Figure 4b). To avoid creating skewed elements around the intersection between the wing upper surface and the embedded surface, the near-field mesh around the wing is generated first. The embedded surface close to or within the near-field mesh is trimmed automatically, and then the rest of the domain is filled with tetrahedral elements.

Figure 5 shows C_p distribution based on the redistributed mesh V^{n1} (Figure 3b), a reference tetrahedral mesh (110k nodes; Figure 5b), the hybrid meshes (Figure 4), and an experimental result. The shock locations of the numerical results do not agree with that of the experimental data well. The viscous flow simulations give better result, but further investigation for the turbulence model is required. In the inviscid flow case, although the reference mesh has almost the same number of nodes as V^{n1} , it is not adapted to the shock feature. The redistributed mesh V^{n1} represents the shock more clearly. In the viscous flow case, the redistributed mesh also represents the shock more clearly.

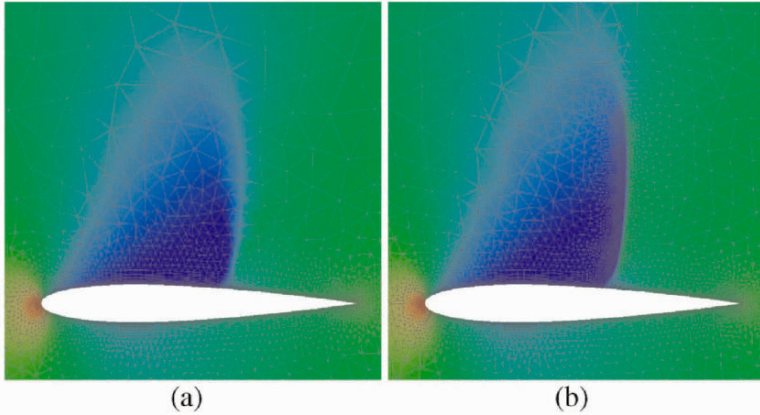


Figure 4. Hybrid meshes for the NACA0012 wing and C_p distribution (-1.0 to 1.0): (a) initial hybrid mesh; (b) redistributed hybrid mesh.

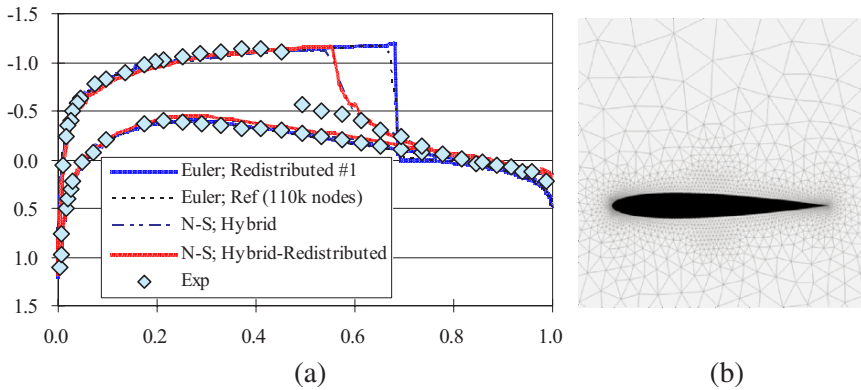


Figure 5. Comparison of C_p distribution for the NACA0012 wing ($M = 0.799$, $\alpha = 2.26$): (a) C_p distribution; (b) reference mesh (110k nodes).

4.2. Capsule Model

Figure 6 shows an initial tetrahedral mesh around a re-entry capsule model and Mach number distribution on a cross-section. The bow shock in front of the capsule becomes steady, but the flow solution is not fully converged. The shape of the outer boundary is a hemisphere so that the mesh can be used for flows at different angles of attack. Isosurfaces of a certain weight function value can be extracted as triangulated surfaces (Figure 7a), the medial axes of which are considered to represent the most important locations. An approach to obtain medial axes using a Delaunay trian-

gulation method from the triangulated surfaces can be considered. However, it is difficult to obtain medial axes automatically as smooth surfaces as discussed in the previous example. Although the isosurfaces shown in Figure 7a are smoothed using a Laplacian method, many holes and small features prevent extracting smooth medial axes.

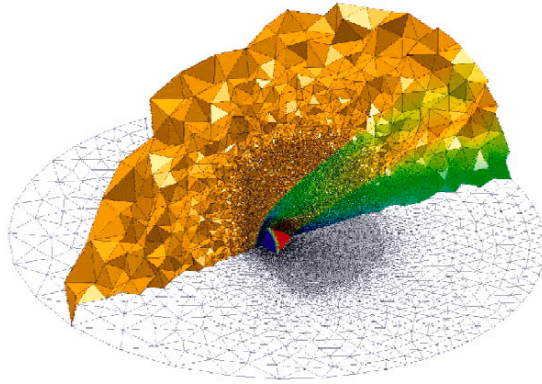


Figure 6. Initial mesh for a capsule model and Mach number distribution on a cross-section ($M = 1.0-4.0$).

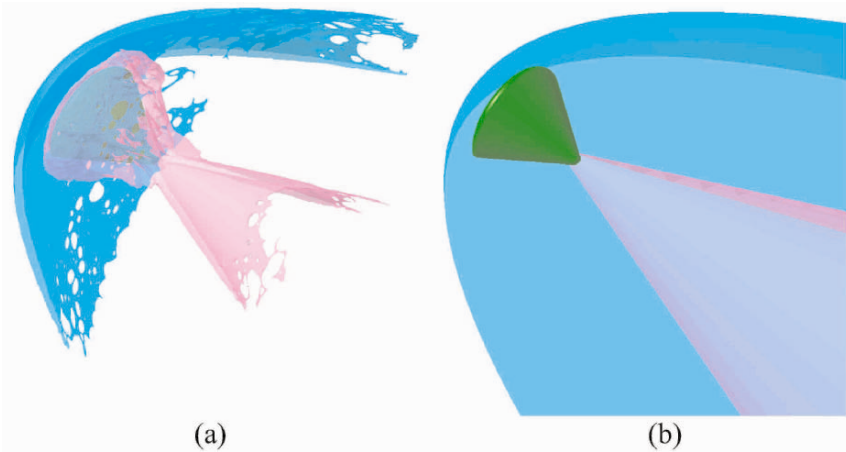


Figure 7. Flow features: (a) extracted isosurfaces at a weight function value of 0.05; (b) estimated features using a least square fitting method.

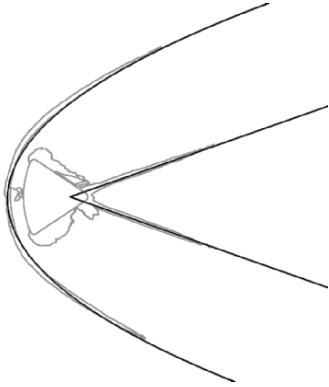


Figure 8. Flow features on the symmetry plane.

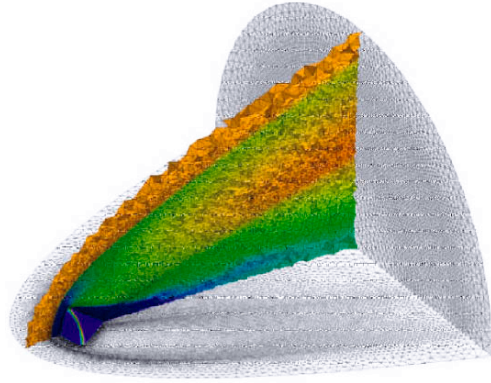


Figure 9. Redistributed mesh for a capsule model and Mach number distribution on a cross-section ($M = 1.0-4.0$).

The other approach using the least square fitting method is more appropriate in this case. After a user specifies one of the template functions, such as a cone, quadratic and quartic, and the z axis of the function, a corresponding medial axis is obtained as a mathematical function (Figure 7). The bow shock in front of the capsule is fitted to a quadratic, and the shock from the aft of it is fitted to a cone. Figure 8 shows the obtained surfaces (Figure 7b) and the isosurfaces for reference (Figure 7a) on the symmetry plane. The least square fitting method estimates the medial axes well. One of the disadvantages using unstructured meshes is that flow features diverge quickly. This approach enables us to estimate missing flow features.

Figure 9 shows a redistributed mesh, which has 0.74 million nodes. In this case, the entire mesh is regenerated because the shape of the outer boundary is changed to remove extra elements. The initial mesh shown in Figure 6 can be used for cases at different angles of attack, but it has 0.89 million nodes. In addition, the elements around the shocks in the far field are coarse. Figure 10 illustrates Mach number distribution on the symmetry planes of the initial and redistributed meshes. Both flows are fully converged. The initial mesh gives a carbuncle phenomenon on the bow shock, while the redistributed mesh gives better result. One of the solution feature surfaces shown in Figure 7b fits the bow shock well.

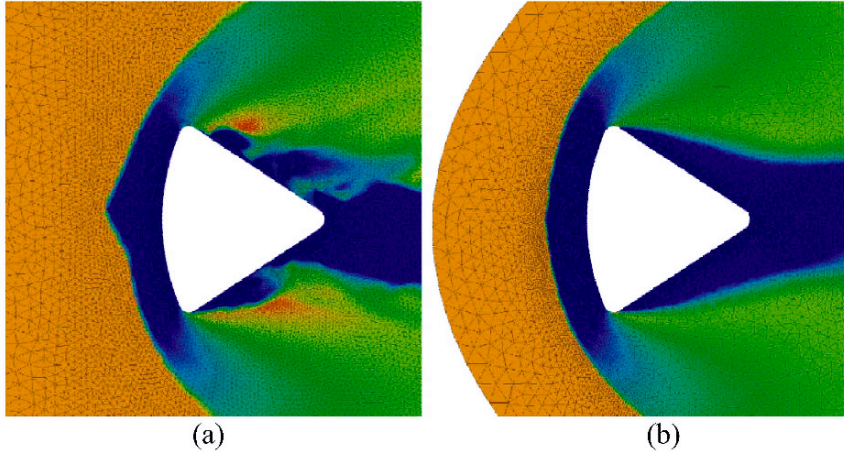


Figure 10. Mach number distribution for the capsule model on the symmetry planes ($M = 1.0-4.0$): (a) the initial mesh showing a carbuncle phenomenon; (b) the redistributed mesh.

The most notable advantage of the surface-based mesh redistribution method is that anisotropic nonsimplicial elements can be used around the feature surfaces to avoid creating skewed elements. Figure 11 shows a redistributed hybrid mesh, which has 0.62 million nodes, based on the same numerical result. Prismatic layers are placed around the bow shock. The quality of the mesh is excellent as shown in the dihedral angle distribution (Figure 11b).

5. Conclusion and Future Work

In this paper, we propose a solution-based mesh redistribution method for strong solution features. Solution features are indicated by a weight function and a shock sensor. The feature locations are estimated by medial axes of isosurfaces at a certain sensor value. To compute medial axes, two approaches are discussed. The discrete surface-based approach using a Delaunay triangulation method may not be suitable to estimate solution features as smooth surfaces. The mathematical-representation approach using least square fitting can represent solution features easily and can interpolate missing features due to truncation errors. The remeshing method with embedded surfaces enables anisotropic nonsimplicial elements to be placed around the features to avoid creating skewed elements.

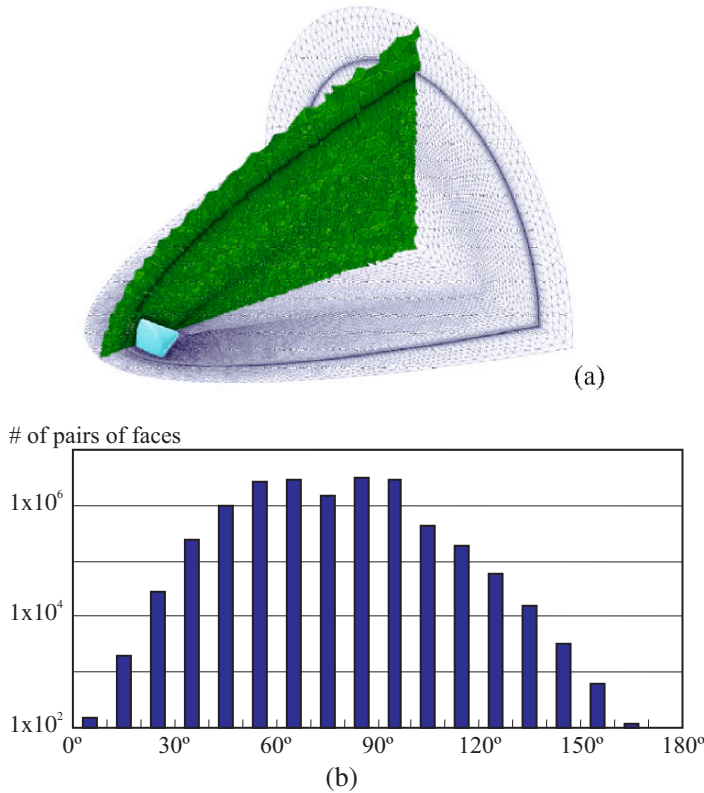


Figure 11. Redistributed mesh for a capsule model using anisotropic elements: (a) hybrid mesh; (b) dihedral angle distribution of the mesh.

The proposed approach, however, does not work well if a solution feature is difficult to be represented as a single surface, such as vortex breakdown. In future work, the solution-based mesh redistribution method will be combined with a mesh refinement method [3] to adapt a mesh to all the solution features efficiently [21].

Acknowledgments

This research is supported in part by the NASA Constellation University Institutes Project (CUIP) No. NCC3-994. Part of this material is based upon work supported by the National Science Foundation under the following NSF programs: Partnerships for Advanced Computational Infrastructure, Distributed Terascale Facility (DTF) and Terascale Extensions: Enhancements to the Extensible Terascale Facility.

References

1. Ito, Y., Shum, P. C., Shih, A. M., Soni, B. K. and Nakahashi, K., "Robust Generation of High-Quality Unstructured Meshes on Realistic Biomedical Geometry," *International Journal for Numerical Methods in Engineering*, Vol. 65, Issue 6, 2006, pp. 943-973.
2. Cavallo, P. A. and Grismer, M. J., "Further Extension and Validation of a Parallel Unstructured Mesh Adaptation Package," AIAA Paper 2005-0924, 43rd Aerospace Sciences Meeting and Exhibit, Reno, NV, 2005.
3. Senguttuvan, V., Chalasani, S., Luke, E. and Thompson, D., "Adaptive Mesh Refinement Using General Elements," AIAA Paper 2005-0927, 43rd AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2005.
4. Soni, B. K., Thornburg, H. J., Koomullil, R. P., Apte, M. and Madhavan, A., "PMAG: Parallel Multiblock Adaptive Grid System," Proceedings of the 6th International Conference on Numerical Grid Generation in Computational Field Simulation, London, UK, 1998, pp. 769-779.
5. Shephard, M. S., Flaherty, J. E., Jansen, K. E., Li, X., Luo, X., Chevaugon, N., Remacle, J.-F., Beall, M. W. and O'Bara, R. M., "Adaptive Mesh Generation for Curved Domains," *Applied Numerical Mathematics*, Vol. 52, Issue 2-3, 2005, pp. 251-271.
6. Suerich-Gulick, F., Lepage, C. and Habashi, W., "Anisotropic 3-D Mesh Adaptation for Turbulent Flows," AIAA Paper 2004-2533, 34th AIAA Fluid Dynamics Conference and Exhibit, Portland, OR, 2004.
7. Mavriplis, D. J., "Grid Resolution Study of a Drag Prediction Workshop Configuration Using the NSU3D Unstructured Mesh Solver," AIAA Paper 2005-4729, 23rd Applied Aerodynamics Conference, Toronto, Canada, 2005.
8. Ito, Y. and Nakahashi, K., "Direct Surface Triangulation Using Stereolithography Data," *AIAA Journal*, Vol. 40, No. 3, 2002, pp. 490-496.
9. Ito, Y., Shih, A. M. and Soni, B. K., "Reliable Isotropic Tetrahedral Mesh Generation Based on an Advancing Front Method," Proceedings of the 13th International Meshing Roundtable, Williamsburg, VA, 2004, pp. 95-105.
10. Ito, Y., Shih, A. M., Soni, B. K. and Nakahashi, K., "An Approach to Generate High Quality Unstructured Hybrid Meshes," AIAA Paper 2006-0530, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2006.
11. Soni, B. K., Koomullil, R., Thompson, D. S. and Thornburg, H., "Solution Adaptive Grid Strategies Based on Point Redistribution," *Computer Methods in Applied Mechanics and Engineering*, Vol. 189, Issue 4, 2000, pp. 1183-1204.
12. Lovely, D. and Haines, R., "Shock Detection from Computational Fluid Dynamics Results," AIAA Paper 1999-3285, 14th AIAA Computational Fluid Dynamics Conference, Norfolk, VA, 1999.
13. Marcum, D. L. and Gaither, K. P., "Solution Adaptive Unstructured Grid Generation Using Pseudo-Pattern Recognition Techniques," AIAA Paper 97-1860, 13th AIAA Computational Fluid Dynamics Conference, Snowmass Village, CO, 1997.
14. Sheehy, D. J., Armstrong, C.G. and Robinson, D. J., "Shape Description by Medial Surface Construction," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, Issue 1, 1996, pp. 62-72.
15. Koomullil, R. P., Thompson, D. S., and Soni, B. K., "Iced Airfoil Simulation Using Generalized Grids", *Journal of Applied Numerical Mathematics*, Vol. 46, Issues 3-4 , 2003, pp 319-330.
16. Koomullil, R. P., and Soni, B. K., "Flow Simulation Using Generalized Static and Dynamics Grids", *AIAA Journal*, Vol. 37, No. 12, 1999, pp 1551-1557.
17. Roe, P. L., "Approximate Riemann Solvers, Parameter Vector, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357-372.

18. Venkatakrishnan, V., "On the Accuracy of Limiters and Convergence to Steady State Solutions," AIAA Paper 93-0880, 31st AIAA Aerospace Sciences Meeting, Reno, NV, 1993.
19. Barth, T. J. and Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, 27th AIAA Aerospace Sciences Meeting, Reno, NV, 1989.
20. Visualization Toolkit (VTK), <http://www.vtk.org/>.
21. Shih, A., Ito, Y., Koomullil, R., Kasmai, T., Jankun-Kelly, M., Thompson, D., and Brewer, W., "Solution Adaptive Mesh Generation using Feature-Aligned Embedded Surface Meshes," 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2007, submitted.

Analysis of Hessian Recovery Methods for Generating Adaptive Meshes

Konstantin Lipnikov¹ and Yuri Vassilevski²

¹ Los Alamos National Laboratory
lipnikov@lanl.gov

² Institute of Numerical Mathematics
vasilevs@dodo.inm.ras.ru

Summary. We study adaptive meshes which are quasi-uniform in a metric generated by the Hessian of a P_1 finite element function. We consider three most efficient methods for recovering this Hessian, one variational method and two projection methods. We compare these methods for problems with anisotropic singularities to show that all Hessian recovery methods result in acceptable adaptive meshes although the variational method gives a smaller error.

1 Introduction

Generation of adaptive meshes is now the standard option in most software packages. Contrary to uniform meshes, adaptive meshes with the same number of elements result in much more accurate solutions of PDEs. In this article, we consider unstructured adaptive meshes which are frequently anisotropic.

One way to generate the adaptive mesh is to define a metric that reflects problem specifics. *A posteriori* error estimates or some features of a problem solution can be used to define this metric. Different error estimates for anisotropic meshes can be found in [8, 9] (see also references therein). In this article, we focus on the maximum norm of the interpolation error for P_1 finite element solutions. In this case, the metric is induced by the Hessian (matrix of second derivatives) of the exact solution. Since the exact solution is not available, the Hessian must be recovered from the finite element solution.

Among many methods for recovering the Hessian, we selected three most efficient methods, one variational method [4] and two projection methods [3, 12]. We compare these methods for problems with anisotropic singularities. This research topic was inspired by the statement made in article [7] that the L_2 -projection method was preferable for the Delaunay-type mesh generation algorithm considered there. In this article, we consider different mesh generation algorithm and show that all Hessian recovery methods result in acceptable adaptive meshes although the variational method gives a smaller error.

The Hessian recovery methods are illustrated with 2D examples. Their extension to 3D tetrahedral meshes is straightforward. These methods have linear complexity with respect to the number of mesh nodes.

The paper outline is as follows. In Section 2, we describe our metric-based mesh generation method. In Section 3, we consider the metric induced by the Hessian of a continuous piecewise linear mesh function and discuss different methods to recover this discrete Hessian. In Section 4, we analyze numerically these methods for problems with anisotropic singularities.

2 Metric-based Mesh Generation

Let Ω be a polygonal domain, and $M(x)$ be a symmetric positive definite 2×2 matrix for any $x = (x_1, x_2)$ in Ω . In metric given by M , the volume of a domain $D \subset \Omega$ and the length of a curve ℓ are defined as follows:

$$|D|_M = \int_D \sqrt{\det M(x)} \, dx, \quad |\ell|_M = \int_0^1 \sqrt{(M(\gamma(t))\gamma'(t), \gamma'(t))} \, dt,$$

where $\gamma(t) : [0, 1] \rightarrow \mathfrak{R}^2$ is a parametrization of ℓ .

Let Ω_h be a triangular mesh covering the domain Ω . In this article, we analyze meshes that are quasi-uniform in metric $M(x)$ (or M -quasi-uniform). These meshes have a number of useful approximation properties when the metric is connected with *a posteriori* error estimates or solution features. A short summary of these properties is given in the next section.

There are many methods for generating a M -quasi-uniform mesh (see, for example, [7]). In this article, we focus on iterative methods that use a sequence of local mesh modifications to generate a global mesh. The list of mesh modifications includes alternation of topology with node deletion/insertion and edge swapping, and node movement. To describe the method, we need the notion of a *mesh quality*. For a triangle Δ of Ω_h , we denote by $p(\Delta)$ the total length of its edges (perimeter) in the metric M . Then, the mesh quality is defined by

$$Q(\Omega_h) = \min_{\Delta \in \Omega_h} Q(\Delta), \quad (1)$$

where $0 < Q(\Delta) \leq 1$ is the quality of triangle Δ (see [4] for more details),

$$Q(\Delta) = 12\sqrt{3} \frac{|\Delta|_M}{p(\Delta)^2} F\left(\frac{p(\Delta)}{3h_*}\right). \quad (2)$$

Here $F(t) : \mathfrak{R}^+ \rightarrow [0, 1]$ is a continuous smooth function, $0 \leq F(t) \leq 1$, with the only maximum at point 1, $F(1) = 1$, and such that $F(0) = F(+\infty) = 0$. Parameter h_* is equal to the size of elements in a mesh consisting of N_* equilateral (in the metric M) triangles. Thus, h_* is a simple function of $|\Omega|_M$ and N_* . The last factor in (2) controls the size of the element, whereas the remaining factors control its shape.

Since the mesh quality is equal to the quality of the worst triangle, the local mesh modifications are applied to this triangle. We accept the first operation that minimizes the mesh quality. If none of the mesh modifications increases mesh quality, the triangle is temporary placed in a special list and the next worst triangle is processed. When the number of elements in the list exceeds some threshold, all of them are released back into the mesh.

This iterative method requires an initial mesh. The initial mesh may be arbitrary and very coarse. The parameter h_* provides simple control over the number of mesh elements. The bigger mesh quality $Q(\Omega_h)$, the closer the number of mesh elements to N_* which is the given number. In a computer program, we terminate the iterative method when $Q(\Omega_h)$ becomes bigger than 0.7.

3 Recovery of Metric from Hessian of Discrete Solution

The special choice of the metric $M(x)$ will allow us to generate meshes with desired properties. The most desirable property for applications is minimization of certain norm of the solution error. In this article, we consider the interpolation error, $u - \mathcal{I}_h u$, where u is a known function and \mathcal{I}_h is the piecewise linear interpolation operator.

A priori error bounds for the maximal norm on optimal meshes have been studied since the beginning of 90s [6]. Recall that the optimal mesh is defined as the mesh that minimizes the interpolation error among conformal meshes with a bounded number of elements. The main difficulty in analysis is possible anisotropy of the optimal mesh.

The lower upper bound on the error for general discrete spaces was obtained by V.Tikhomirov in 1960. In [1, 10], we analyzed a more narrow class of spaces of continuous piecewise linear functions on conformal triangulations and showed that the optimal meshes still provide the same error decay which is reciprocal to the number of elements N_* . In [11], our analysis has been extended to L_p -norm, where $p > 0$. For sufficiently smooth solutions ($u \in C^2(\bar{\Omega})$) and $0 < p \leq +\infty$, the asymptotical error is

$$\|u - \mathcal{I}_h u\|_{L_p(\Omega)} \sim N_*^{-1}.$$

Moreover, in [1, 10] we proved that meshes which are only quasi-uniform in the metric derived from the Hessian of u still result in the optimal interpolation error. Now, we describe this metric in more details.

Let $H(x)$ be the Hessian of u ,

$$H = \{H_{ij}\}_{i,j=1}^2, \quad H_{ij} = \frac{\partial^2 u}{\partial x_i \partial x_j}, \quad i, j = 1, 2.$$

We consider its spectral decomposition $H(x) = W^T(x)A(x)W(x)$ and generate the metric $M(x)$ as follows:

$$M(x) = W^T(x)|\Lambda(x)|W(x), \quad |\Lambda(x)| = \text{diag}\{|\lambda_1(x)|, |\lambda_2(x)|\}, \quad (3)$$

where $\lambda_i(x)$ are eigenvalues of $H(x)$. To emphasize distinctive nature of this metric, we shall write $|H(x)|$ instead of $M(x)$.

In a practical adaptive computation, the unknown solution Hessian $H(x)$ must be replaced by its discrete approximation H^h . In [1, 10], we have formulated sufficient conditions for approximation H^h under which the mesh quasi-uniform in the metric $|H^h|$ is also quasi-uniform in the metric $|H(x)|$. If the initial mesh, used to recover H^h , is far from the optimal one, the discrete Hessian may not approximate the continuous one. In this case, the problem is solved again on the generated mesh and the adaptation process is repeated. The final mesh will be referred to as the quasi-optimal mesh.

There are a few methods for recovering the discrete Hessian H^h from the piecewise linear function u^h defined at mesh nodes. In next sections, we analyze numerically one variational and two projection methods. It was reported in [7] that the projection method was more robust for the Delaunay-like mesh generation algorithm described there. Since we have different experience with the iterative mesh generation algorithm described in the previous section, we decided to analyze the effect of the Hessian recovery onto the interpolation error.

3.1 Variational Methods

Let superelement σ be a set of triangles sharing an interior mesh node a and $H^h(a)$ be the value of the continuous piecewise linear Hessian H^h at this mesh node. The weak definition of the discrete Hessian is as follows:

$$\int_{\sigma} H_{ij}^h(a) \varphi_a^h \, dx = - \int_{\sigma} \frac{\partial u^h}{\partial x_i} \frac{\partial \varphi_a^h}{\partial x_j} \, dx, \quad (4)$$

where φ_a^h is the piecewise linear (P_1) finite element function associated with the node a . Since φ_a^h vanishes on $\partial\sigma$, definition (4) is nothing else but the Green formula.

At a boundary mesh node a , the Hessian $H^h(a)$ is the weighted extrapolation from the neighboring interior nodes [5]:

$$H_{ij}^h(a) = \frac{\sum_b H^h(b) m_{ab}}{\sum_b m_{ab}}, \quad m_{ab} = \int_{\sigma} \varphi_a^h \varphi_b^h \, dx, \quad (5)$$

where summation goes over mesh points $b \in \partial\sigma$ that are not on $\partial\Omega$.

The approximation properties of the recovery method (4) have been studied in [1, 10]. The local convergence of the discrete Hessian towards the differential one has been established for a class of smooth solutions. Another variational method for the Hessian recovery has been introduced and analyzed in [2].

3.2 Projection Methods

Another group of methods for recovering the Hessian H^h uses the projection technique. The first method from this group reads:

$$H^h = \mathcal{I}_{ZZ} (\nabla (\mathcal{I}_{ZZ}(\nabla u^h))) \quad (6)$$

where \mathcal{I}_{ZZ} is the Zienkiewicz-Zhu (ZZ) projector on the P_1 finite element space [12]. The gradient of u^h is constant inside mesh triangles. To find the gradient of u^h at an interior mesh node a of the superelement σ defined above, we fit (in the least square sense) a linear function to the values of ∇u^h assigned to the centers of mass of triangles in σ . Repeating this procedure twice, we get the piecewise linear Hessian of u^h at the node a . Again, the Hessian is extrapolated to boundary nodes using (5).

In [7], the L_2 -projector \mathcal{I}_{L_2} is used instead of the ZZ-projector. However, both projectors do not remove high frequency errors introduced by small non-uniformities in the mesh. The superconvergence properties of the recovered gradient are restored by applying a smoothing operator S^h right after the projection operator:

$$H^h = S^h \mathcal{I}_{ZZ} (\nabla (S^h \mathcal{I}_{ZZ}(\nabla u^h))). \quad (7)$$

As shown in [3], two Jacobi conjugate gradient iterations for the equation $-\Delta v + v = 0$ with the initial guess corresponding to $\mathcal{I}_{ZZ}(\nabla u^h)$ are sufficient to dump high frequency errors in majority of problems. It is pertinent to note that the authors studied only isotropic meshes and the optimal number of Jacobi iteration was found experimentally. In the next section, we shall show that the same conclusion cannot be extended to anisotropic meshes and the maximal norm.

Note that for isotropic solutions, efficient implementation of the L_2 -projector can be done by lumping the mass matrix [3]. In other cases, solution of a linear system with a mass matrix is required to find $\mathcal{I}_{L_2}(\nabla u^h)$. Since the mass matrix may be quite stiff on an anisotropic mesh, we think that the \mathcal{I}_{ZZ} -projector is a more reasonable choice in (7) than the L_2 -projector suggested in [3].

All three Hessian recovery methods have linear complexity with respect to the number of mesh nodes. The variational method is less expensive. The complexity of the second projection method grows linearly with the number of smoothing iterations. The complexity of these methods is less than complexity of the mesh generation algorithm when the initial mesh is far from the optimal one. In the course of adaptive iterations the number of mesh modifications decreases and the relative complexity of the Hessian recovery methods grows. The actual numbers are essentially problem dependent.

4 Numerical Experiments

In this section, we analyze numerically three Hessian recovery methods given by (4), (6), and (7). We shall refer to the last two methods as the ZZ-projection and the BX-projection methods, respectively.

In the first example, we consider the problem of minimizing the interpolation error for the function [6]

$$u(x_1, x_2) = \frac{(x_1 - 0.5)^2 - (\sqrt{10}x_2 + 0.2)^2}{((x_1 - 0.5)^2 + (\sqrt{10}x_2 + 0.2)^2)^2}.$$

defined over the unit square $[0, 1]^2$. The function has slight anisotropic singularity at point $(0.5, -0.2/\sqrt{10})$ which is outside the computational domain but close to its boundary. The Hessian of u is the saddle point matrix, i.e. it has one positive and one negative eigenvalue.

The solution isolines and the adapted mesh are shown in Fig. 1. Table 1 shows the L_∞ and H^1 norms of the interpolation error. The maximal norm of the interpolation error is almost reciprocal to N_* . The energy norm is given only for illustration purposes. All the methods require 4 to 5 adaptive iterations to generate quasi-optimal meshes. *The variational method results in the most accurate interpolation solution.*

The gradient smoothing in the BX-projection method does not affect the energy norm of error but increases the maximum norm in 2-3 times. This factor becomes even bigger if we increase the number of smoothing iterations.

Table 1. Experiment 1: L_∞ and H^1 norms of the interpolation error for three Hessian recovery methods.

N_*	Variational method		ZZ-projection method		BX-projection method	
	L_∞	H^1	L_∞	H^1	L_∞	H^1
3000	0.0334	0.206	0.0410	0.211	0.1201	0.234
6000	0.0164	0.146	0.0234	0.147	0.0443	0.158
9000	0.0102	0.119	0.0138	0.119	0.0326	0.123

In the second experiment, we build the optimal mesh for the function proposed in [7]:

$$u(x_1, x_2) = x_2x_1^2 + x_2^3 + \tanh(10(\sin(5x_2) - 2x_1)).$$

The computational domain is the square $[-1, 1]^2$. The solution is anisotropic along the zigzag curve (see left picture in Fig. 2) and changes sharply in the direction normal to this curve.

Table 2 shows the L_∞ and H^1 norms of the interpolation error. As in the previous example, *the variational method results in the smallest interpolation error.* Similarly, the energy norm of error does not drop down as fast as the

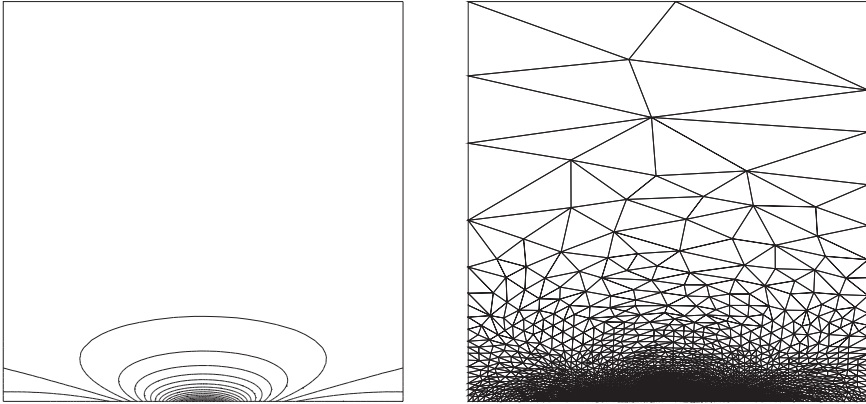


Fig. 1. Isolines of function u from the first example (left) and the adapted mesh after 5 iterations for the Hessian recovered with the variational method (right).

maximal norm since we minimize only the latter. Note that setting the number of smoothing iterations to 10 results in only 19% increase in the energy norm but destroys the anisotropic mesh structure and rockets up the maximal norm.

In Fig. 3, we show the behavior of the maximal norm of error during the course of adaptive iterations. Due to the discrete nature of the metric $|H^h|$, this behavior is frequently non-monotone. We point out the visible oscillations and slower convergence of the BX-projection method. The other two methods required 5 iterations for convergence. We explain this by a slight smearing of the recovered gradient by operator S^h .

Table 2. Experiment 2: L_∞ and H^1 norms of the interpolation error for three Hessian recovery methods.

N_*	Variational method		ZZ-projection method		BX-projection method	
	L_∞	H^1	L_∞	H^1	L_∞	H^1
3000	0.02081	0.920	0.02271	0.945	0.1464	0.906
6000	0.00846	0.686	0.01110	0.889	0.0427	0.716
9000	0.00561	0.650	0.00899	0.627	0.0186	0.635

It is pertinent to note that the L_∞ -norm of the error averaged over mesh triangles is close for all three methods. This indicates that the number of lower-quality triangles is much smaller than N_* . However, we should keep it mind that these triangles are usually located in crucial regions from the physical viewpoint.

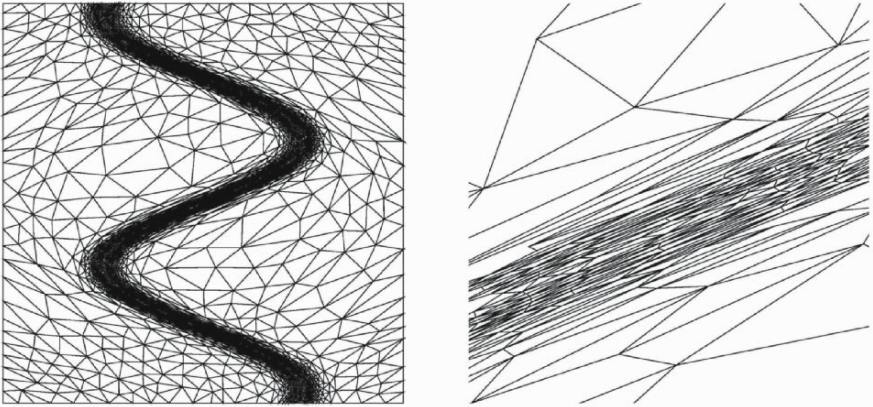


Fig. 2. Adapted mesh after 6 iterations for the Hessian recovered with the variational method (left) and its zoom around point $(0, 0)$ showing the anisotropic mesh structure.

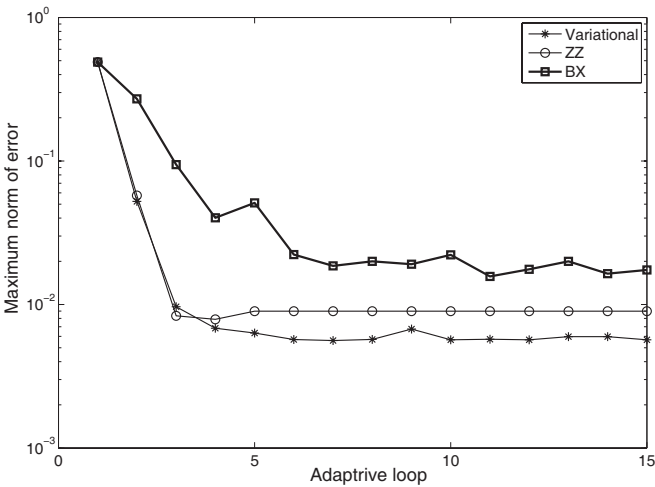


Fig. 3. The L_∞ -norm of the interpolation error during the course of 15 adaptive iterations with $N_* = 9000$.

In conclusion, we note that the choice of the recovery method must depend on the norm in which we want to minimize the error. For the maximal norm, the gradient smoothing is the bad idea. The variational method seems to exhibit the most robust behavior over a larger scale of norms.

Acknowledgements

The work of the second author was supported in part by the RFBR grant 04-07-90336 and the academic programs “Computational and informatic issues of the solution of large problems” and “Modern problems in theoretical mathematics”.

References

1. Agouzal A, Lipnikov K, Vassilevski Y (1999) Adaptive generation of quasi-optimal tetrahedral meshes, *East-West J. Numer. Math.* **7**, 223–244.
2. Agouzal A, Vassilevski Y (2002) On a discrete Hessian recovery for P_1 finite elements, *J. Numer. Math.* **10**, 1–12.
3. Bank RE, Xu J (2003) Asymptotically exact a posteriori error estimators, part II: General unstructured meshes, *SIAM J. Numer. Anal.* **41**, 2313–2332.
4. Buscaglia GC, Dari EA (1997) Anisotropic mesh optimization and its application in adaptivity, *Inter. J. Numer. Meth. Engrg.* **40**, 4119–4136.
5. Buscaglia GC, Agouzal A, Ramirez P, Dari EA (1998) On Hessian recovery and anisotropic adaptivity, In Proc. of ECCOMAS 98 *the 4th European Computational Fluid Dynamics Conference, 7-11 September 1998, Athens Greece*, John Wiley & Sons, 1998, Vol. 1, Part 1, pp. 403–407.
6. D’Azevedo E (1991) Optimal triangular mesh generation by coordinate transformation, *SIAM J. Sci. Stat. Comput.* **12**, 755–786.
7. Hecht F (2005) *A few snags in mesh adaptation loops*, In proceedings of the 14th International Meshing Roundtable, Springer-Verlag Berlin Heidelberg.
8. Kunert G (2001) A local problem error estimator for anisotropic tetrahedral finite element meshes, *SIAM J. Numer. Anal.* **39**, 668–689.
9. Picasso M (2003) An anisotropic error indicator based on Zienkiewicz-Zhu error estimator: Application to elliptic and parabolic problems, *SIAM J. Sci. Comput.* **24**, 1328–1355.
10. Vassilevski Y, Lipnikov K (1999) Adaptive algorithm for generation of quasi-optimal meshes, *Comp. Math. Math. Phys.* **39**, 1532–1551.
11. Vassilevski Y, Agouzal A (2005) An unified asymptotical analysis of interpolation errors for optimal meshes, *Doklady Mathematics* **72**, 879–882.
12. Zhu JZ, Zienkiewicz OC (1990) Superconvergence recovery technique and a posteriori error estimators, *Inter. J. Numer. Meth. Engrg.* **30**, 1321–1339.

Anisotropic Mesh Adaptation for Evolving Triangulated Surfaces

Xiangmin Jiao¹, Andrew Colombi², Xinlai Ni², and John C. Hart²

¹College of Computing, Georgia Institute of Technology, Atlanta, GA 30332

²Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801

Summary. Dynamic surfaces arise in many applications, such as free surfaces in multiphase flows and moving interfaces in fluid-solid interactions. In many applications, an explicit surface triangulation is used to track the dynamic surfaces, posing significant challenges in adapting their meshes, especially if large curvatures and sharp features may dynamically appear or vanish as the surfaces evolve. In this paper, we present an anisotropic mesh adaptation technique to meet these challenges. Our technique strives for optimal aspect ratios of the triangulation to reduce interpolation errors and to capture geometric features based on a novel extension of the quadric-based surface analysis. Our adaptation algorithm combines the operations of vertex redistribution, edge flipping, edge contraction, and edge splitting. Experimental results demonstrate the effectiveness of our anisotropic adaptation techniques for static and dynamic surfaces.

Key words: Mesh adaptation; anisotropic meshes; dynamic surfaces; feature preservation

1 Introduction

Many computational applications involve triangulation of complex surface geometry, and an increasing number of them involve dynamically changing surfaces, such as free surfaces in multiphase flows [39] and moving boundaries in fluid-solid interactions [26]. In these simulations, the geometry is not known *a priori* and is part of the solution of a numerical simulation. As a surface evolves, the surface may undergo severe expansion or contraction in different regions or along different directions, leading to large curvatures, sharp features, and even topological changes. It is therefore often necessary to adapt the meshes for these complex dynamic surfaces to maintain a valid surface representation with minimal geometric errors.

Mesh adaptation has been an active research subject in numerical simulations for nearly two decades [32, 35]. In recent years anisotropic generation

and adaptation of 2-D triangular and 3-D tetrahedral meshes have attracted significant attention to reduce or minimize errors [3, 7, 8, 11, 12, 13, 15, 19, 25, 28, 29, 37, 38]. In general, these methods strive to equi-distribute errors by adapting the mesh density based on a metric tensor field, typically estimated based on the Hessian of a solution field. Anisotropic meshes can also significantly enhance the accuracy of a surface representation [1, 6, 10, 20]. Very few methods have been developed on anisotropic meshing and remeshing of a surface mesh [1, 10]. The method of Alliez et al. [1] generates quad-dominant meshes, and the method of Cheng et al. [10] considers only smooth, implicit surfaces, which may not be suitable for triangulated surfaces with sharp features that occur in many simulations. In addition, moving surfaces introduce significant additional complexities and constraints to mesh adaptation. A robust dynamic triangulation algorithm was developed by Cheng et al. [9], which is specialized for skin surfaces and it is unclear how to generalize that algorithm to other surfaces. Therefore, anisotropic mesh adaptation for static or dynamic surfaces remains a significant challenge.

In this paper, we investigate the problem of adapting a dynamic surface mesh within a numerical simulation to reduce geometric errors. We propose an extension of the quadric-based surface analysis by Heckbert and Garland [20] and relate it to the interpolation error of a surface. Based on this analysis, we define a Riemannian metric tensor to adapt the surface mesh anisotropically using a combination of vertex redistribution, edge flipping, edge contraction, and edge splitting. These operations improve not only mesh quality but also the accuracy of the geometric representation. The interplay between different operations is potentially very complicated. To keep the algorithm simple, we optimize the mesh with vertex redistribution and edge flipping under geometric constraints, and use edge splitting and edge contraction to resolve pathological situations due to constraints. We compare the numerical solutions using anisotropic adaptation and isotropic adaptation, and demonstrate the significant advantages of our anisotropic adaptation. For simplicity this paper assumes the surface does not change topology during evolution.

The remainder of the paper is organized as follows. Section 2 presents some background information on anisotropic meshes and quadric-based surface analysis. Section 3 proposes a novel anisotropic transformation for surface meshes and applies it to mesh optimization using vertex redistribution and edge flipping. Section 4 describes the resolution of extreme angles and adaptation of mesh density using edge splitting and edge contraction. Section 5 presents numerical results and comparisons with isotropic adaptation for static and dynamic surfaces. Finally, Section 6 concludes the paper with a discussion.

2 Background

In this section, we present some background information on anisotropic meshes and quadric-based surface analysis.

2.1 Anisotropic Meshes

It is well known that a mesh with long and skinny triangles can interpolate a function with large curvature more accurately than a mesh with equilateral triangles [11, 13, 19, 36, 38]. A general formulation of anisotropic mesh generation or adaptation in 2-D or 3-D is to define a $d \times d$ *metric tensor* $\mathbf{M}(\mathbf{x})$ (or simply \mathbf{M}) at each point \mathbf{x} in \mathbb{R}^d , where d is 2 or 3, respectively. \mathbf{M} is referred to as the *Riemannian metric tensor* or *fundamental tensor* [24]. By definition, \mathbf{M} is symmetric positive-definite and has an eigen decomposition

$$\mathbf{M} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T, \quad (1)$$

where $\mathbf{\Lambda}$ is the diagonal matrix of the eigenvalues of \mathbf{M} , which are all real and positive, and \mathbf{E} is the matrix of the eigenvectors of \mathbf{M} . Geometrically, \mathbf{E} corresponds to a rotation matrix and $\mathbf{\Lambda}$ corresponds to scaling factors. Any point $\mathbf{y} = \mathbf{x} + \epsilon$ in an infinitesimal neighborhood of \mathbf{x} is mapped to

$$\tilde{\mathbf{y}} = \mathbf{y} + \sqrt{\mathbf{\Lambda}\mathbf{E}^T} \epsilon = \mathbf{y} + \sum_{i=1}^d \sqrt{\lambda_i} \mathbf{e}_i^T \epsilon \mathbf{e}_i, \quad (2)$$

which maps a circle (or sphere in 3-D) in the physical space into an ellipse (or ellipsoid) in a parametric space with the semiaxes proportional to $\sqrt{\lambda_i}$ for $i \in [1, d]$. A curve $\mathbf{r}(s) : [a, b] \rightarrow \mathbb{R}^2$ is mapped to a curve with length

$$l = \int_a^b \sqrt{\dot{\mathbf{r}}^T \mathbf{M} \dot{\mathbf{r}}} ds, \quad (3)$$

which can be approximated using the midpoint rule as

$$l \approx \sqrt{(\mathbf{r}_b - \mathbf{r}_a)^T \mathbf{M}_{(a+b)/2} (\mathbf{r}_b - \mathbf{r}_a)}. \quad (4)$$

In numerical computations, \mathbf{M} is typically chosen to minimize the interpolation error of a function f in a certain norm. Using Taylor series expansion one can show that the error in interpolating a function f with linear elements is approximately $\epsilon^T \mathbf{H} \epsilon$ as ϵ tends to zero, where \mathbf{H} denotes the Hessian of f . The Hessian is symmetric and therefore has an eigen decomposition $\mathbf{H} = \mathbf{R}\mathbf{D}\mathbf{R}^T$, where \mathbf{D} is a diagonal matrix of the eigenvalues of \mathbf{H} (i.e., the curvatures of f) and \mathbf{R} is composed of the eigenvectors of \mathbf{H} . The eigenvalues of \mathbf{H} may be negative, and the metric tensor \mathbf{M} can be obtained by setting $\mathbf{\Lambda} = |\mathbf{D}|$ and $\mathbf{E} = \mathbf{R}$ in (1), i.e., $\mathbf{M} = \mathbf{R}|\mathbf{D}|\mathbf{R}^T$, which minimizes the interpolation error by equi-distributing the error.

To generalize the above formulation to surface meshes, a 2-D metric tensor may be defined on a local or global parametric space [1, 10]. For solution-based adaptation the metric tensor may be assumed to be given as an input to the meshing algorithm, but in problems such as dynamic surfaces, which themselves are the solutions, the adaptation requires a careful analysis of the evolving surface.

2.2 Quadric-Based Surface Analysis

Surface analysis is a fundamental subject in differential and numerical geometry. Mathematical analysis has traditionally focused on smooth surfaces [24]. Its generalization to discrete surfaces has attracted significant attention in recent years, but the focus has been limited to the asymptotic behavior of discrete representations of smooth surfaces [20, 30]. If a surface discretization is relatively coarse or has singularities, then most asymptotic analyses break down. The quadric-based analysis proposed by Heckbert and Garland [20] seems to generalize well for their connections with approximation theory [31] and singularity analysis, as we will show shortly.

Given a triangulated surface mesh let each vertex v be the origin of a local coordinate frame, and m be the number of the faces incident on v . Let \mathbf{N} be a $m \times 3$ matrix whose i th row vector is the unit outward normal to the i th incident face of v , and \mathbf{W} be an $m \times m$ diagonal matrix with W_{ii} equal to the weight associated with the i th face. We typically use the the face area of the i th incident face of v for the weight W_{ii} . Let \mathbf{A} denote the 3×3 matrix $\mathbf{N}^T \mathbf{W} \mathbf{N}$, which we refer to as the *quadric metric tensor*. Suppose \mathbf{G} is a diagonal matrix containing the eigenvalues of \mathbf{A} . Let λ_i denote \mathbf{G}_{ii} with $\lambda_1 \geq \lambda_2 \geq \lambda_3$, and \mathbf{V} be the matrix of the eigenvectors of \mathbf{A} , so $\mathbf{A} = \mathbf{V} \mathbf{G} \mathbf{V}^T$. We refer to the vector space spanned by the eigenvectors of \mathbf{A} corresponding to the relatively large eigenvalues of \mathbf{A} as its *primary space* and the complementary space as its *null space*. As we will show later, for smooth surfaces the null space of \mathbf{A} is closely related to the Riemannian metric tensor for error minimization, and the threshold between the eigenvalues for the primary and null spaces can be chosen geometrically.

Suppose the triangles incident on v discretize a rectangular neighborhood with dimensions ϵ_1 and ϵ_2 along the maximum and minimum curvature directions, and κ_1 and κ_2 are the maximum and minimum curvatures, respectively. Heckbert and Garland [20] have shown that as ϵ tends to 0 the eigenvalues of \mathbf{A} are

$$\lambda_1 \approx 4\epsilon_1\epsilon_2 - \frac{1}{2}(\lambda_2 + \lambda_3) \quad (5)$$

$$\lambda_2 \approx \frac{4}{3}\epsilon_1^3\epsilon_2\kappa_1^2 \quad (6)$$

$$\lambda_3 \approx \frac{4}{3}\epsilon_1\epsilon_2^3\kappa_2^2 \quad (7)$$

For meshes approximating a smooth surface, λ_2/λ_1 and λ_3/λ_1 approach 0 in the rates of $O(\epsilon_1^2)$ and $O(\epsilon_2^2)$, respectively.

This asymptotic analysis, however, is not applicable near singularities. We generalize it to treat singularities as follows. Consider folding a smooth surface to form a ridge along the minimum curvature direction at a point, and let θ denote the dihedral angle between the two sides of the ridge. It is easy to show that after the folding, the three eigenvalues become

$$\tilde{\lambda}_1 \approx \lambda_1 \cos^2 \frac{\theta}{2} + \lambda_2 \sin^2 \frac{\theta}{2} \quad (8)$$

$$\tilde{\lambda}_2 \approx \lambda_1 \sin^2 \frac{\theta}{2} + \lambda_2 \cos^2 \frac{\theta}{2} \quad (9)$$

$$\tilde{\lambda}_3 \approx \lambda_3 \quad (10)$$

Since $\lambda_2/\lambda_1 = O(\epsilon_1^2)$ before folding, near singularities $\tilde{\lambda}_2/\tilde{\lambda}_1 \approx \tan^2(\theta/2)$ after folding. At sharp corners, the eigenvalues become more complicated, but in general all three eigenvalues are $O(\epsilon_1\epsilon_2)$. This singularity analysis provides a geometrically meaningful way to select the thresholds for the null space (χ_c for λ_2/λ_1 and χ_r for λ_3/λ_1). It is the foundation of the feature detection methods of Jiao [22], Jiao and Alexander [23], which detect ridges and corners by determining the null space along with some safeguards.

The asymptotic interpretation $\lambda_2/\lambda_1 \approx \epsilon_1^2\kappa_1^2/3$ and the singularity interpretation $\tilde{\lambda}_2/\tilde{\lambda}_1 \approx \tan^2(\theta/2)$ are consistent within a constant factor, in that $\tan^2(\theta/2)$ approaches $\epsilon_1^2\kappa_1^2$ as θ tends to zero. This dual asymptotic and singularity analysis is very useful, as it provides a new tool to handle the ambiguous cases that arise frequently in discrete surfaces. In particular, a relatively coarse mesh may behave in an ambiguous manner like neither singular nor smooth surfaces, but λ_2/λ_1 would be approximately $\tan^2(\theta/2)$ and \mathbf{e}_3 would be along the minimum curvature direction. This dual analysis is particularly useful for dynamic surfaces, in which some areas may have increasingly large curvatures before sharp features emerge, and the quadrics may be used to capture such a transition consistently without resorting to two drastically different treatments for smooth regions and sharp features.

The quadrics are also closely related to optimal triangles in an asymptotic sense. We define the aspect ratio ρ of a triangle to be that of its minimum containing ellipse, as advocated by Heckbert and Garland [20]. A dual definition, where the optimal triangle is defined as the triangle with maximum area contained in an ellipse with a given aspect ratio, can also be used [13]. In both cases, the optimal aspect ratio of a triangle is $\sqrt{\kappa_1/\kappa_2}$, where κ_1 and κ_2 are the eigenvalues of the Hessian of the surface with respect to its local parameterization. It has been shown that the quadric-based surface simplification produces optimal anisotropic triangles in an asymptotic sense for smooth surfaces [20], but there seems to be no published result on how to apply this analysis to mesh adaptation. In the following, we propose a quadric-based procedure for anisotropic mesh adaptation.

3 Anisotropic Mesh Optimization

It is challenging to define a suitable tensor field for surface meshes while taking into account the potential singularities. In this section we propose a tensor field and apply it to anisotropic vertex redistribution and edge flipping.

3.1 Anisotropic Transformation

For triangulations of smooth surfaces one can construct the Riemannian metric tensor for anisotropic transformation by computing the curvature tensors at each vertex to approximate the Hessian of a surface [1]. Although this approach is well founded for sufficiently fine discretization of smooth surfaces, the estimated curvatures are not meaningful near singularities and may be sensitive to perturbation for coarse meshes.

To avoid these numerical problems, we propose to construct the Riemannian tensor $\mathbf{M}_{2 \times 2}$ based on the quadrics as

$$\mathbf{M} = \mathbf{R} \begin{bmatrix} \lambda_2 & 0 \\ 0 & \lambda_3 \end{bmatrix} \mathbf{R}^T, \quad (11)$$

where λ_i are the eigenvalues of the quadric metric tensor \mathbf{A} , and \mathbf{R} is a 2×2 rotation matrix from the principal directions to the axes of the coordinate frame. From our previous analysis, for surface meshes conforming to the tensor field \mathbf{M} , $\rho = \sqrt{\lambda_2/\lambda_3} \approx \sqrt{|\kappa_1/\kappa_2|}$. To see this, observe that

$$\rho = \sqrt{\lambda_2/\lambda_3} \approx \frac{|\epsilon_1 \kappa_1|}{|\epsilon_2 \kappa_2|} = \frac{|\kappa_1/\kappa_2|}{\epsilon_2/\epsilon_1}.$$

By definition $\rho = \epsilon_2/\epsilon_1$, and therefore $\rho^2 \approx |\kappa_1/\kappa_2|$. Note that we can multiply \mathbf{M} by any factor without changing the aspect ratio. The computation of \mathbf{M} does not require estimating the curvatures explicitly, so it is simple and efficient. In addition, \mathbf{A} and in turn \mathbf{M} are computed in an integral form, so they are not sensitive to perturbation, and it is justified to construct a larger geometric support by summing up \mathbf{A} (or \mathbf{M}) at neighboring vertices.

It is important to note that the preceding analysis is asymptotic in nature, and may be invalid near singularities and degenerate cases. In particular, if $\kappa_1 \approx \kappa_2 \approx 0$, the aspect ratio is very sensitive to perturbation. If $\lambda_2 \gg \lambda_3$, which may occur near singularities (where $\lambda_3/\lambda_2 = O(\epsilon^2)$ along a ridge) or on a cylindrical patch (where $\lambda_3/\lambda_2 = 0$), the aspect ratio may become arbitrarily large. Too large aspect ratios may lead to too small time steps or larger condition numbers in a numerical simulation [36], and in turn severely decrease the efficiency and defeat the purpose of anisotropic adaptation. To resolve these issues, we impose an upper bound on the aspect ratio by imposing lower and upper bounds on the eigenvalues (the entries of the diagonal matrix in (11)) of \mathbf{M} , i.e.,

$$\widetilde{\mathbf{M}} = \mathbf{R} \begin{bmatrix} \min\{\max\{\lambda_2, \psi_l\}, \psi_u\} & 0 \\ 0 & \min\{\max\{\lambda_3, \psi_l\}, \psi_u\} \end{bmatrix} \mathbf{R}^T. \quad (12)$$

Based on the singularity analysis of the quadrics, ψ_l and ψ_u should be in the form of $\lambda_1 \tan^2(\theta/2)$ for some angle θ . Compared with the thresholds χ_r and χ_c for feature detection, in general $\psi_l/\lambda_1 \leq \chi_r \leq \psi_u/\lambda_1 \leq \chi_c$ (e.g., 8° , 20° , 30° , and 45° after converting ψ_l/λ_1 , χ_r , ψ_u/λ_1 , and χ_c into angles by the mapping $f(\chi) = 2 \arctan \sqrt{\chi}$). These angles are tunable parameters that can be selected based on their geometric meanings and experiments. With this modified tensor, the surface would be strongly anisotropic only near sharp features or large-curvature areas, where anisotropy is desired to avoid large interpolation errors.

3.2 Anisotropic Mesh Smoothing

Mesh smoothing is a popular method for mesh enhancement by redistributing vertices without changing mesh connectivity. It is also often the most difficult step in mesh adaptation. This problem is particularly challenging for smoothing surfaces because the vertex motion must preserve the geometry typically without a CAD model. A high-order surface approximation may be constructed [1, 6, 18], but large errors may still accumulate after repeated smoothing or adaptation of a dynamic surface. A simple solution was proposed by Jiao [22] through a weighted-residual formulation of local volume conservation. Here we use a simpler form of that method to redistribute vertices tangentially within the null space of \mathbf{A} , and focus on the issue of anisotropy in mesh smoothing.

A typical smoothing procedure moves a point toward a weighted average of its neighbor vertices. For anisotropic smoothing we must take into account the Riemannian tensors when computing the average. In the previous subsection we discussed the construction of the tensor at each vertex, which uses its own local coordinate frame. To compute a weighted average at a vertex p , the tensors at p and its adjacent vertices must be transformed into the same coordinate frame.

Given two adjacent vertices q and p , we compute the rotation matrix \mathbf{R} as follows. If the first eigenvectors at the two vertices are the same, then the rotation matrix is simply $\mathbf{T}_q^T \mathbf{T}_p$, where \mathbf{T}_p is a 3×2 matrix whose column vectors are the second and third eigenvectors of \mathbf{A} at p , and similarly for \mathbf{T}_q . If the normals differ, then $\mathbf{T}_q^T \mathbf{T}_p$ is no longer a rotation matrix. Let \mathbf{S} denote $\mathbf{T}_q^T \mathbf{T}_p$. We compute \mathbf{R} based on the projection of the third eigenvector of \mathbf{A} at p onto \mathbf{T}_q , i.e.,

$$\mathbf{R}_2 = \frac{\mathbf{S}_2}{\|\mathbf{S}_2\|}, \text{ and } \mathbf{R}_1 = \frac{\mathbf{S}_1 - \mathbf{R}_2 \mathbf{S}_1 \mathbf{R}_2}{\|\mathbf{S}_1 - \mathbf{R}_2 \mathbf{S}_1 \mathbf{R}_2\|},$$

where \mathbf{R}_i and \mathbf{S}_i denote the i th column vectors of \mathbf{R} and \mathbf{S} , respectively. By plugging \mathbf{R} corresponding to each adjacent vertex q into (12), we trans-

form the tensor at q to the local frame at p . For each edge pq , its midpoint \mathbf{x} is then transformed to $\sqrt{\mathbf{D}_{\frac{1}{2}} \mathbf{R}_{\frac{1}{2}}^T \mathbf{T}_p^T (\mathbf{x} - \mathbf{p})}$, where $\mathbf{R}_{\frac{1}{2}} \mathbf{D}_{\frac{1}{2}} \mathbf{R}_{\frac{1}{2}}^T = \mathbf{M}_{\frac{1}{2}} = \frac{1}{2}(\mathbf{M}_p + \mathbf{M}_q)$, and the length of the edge is estimated based on (4) as $\sqrt{(\mathbf{p} - \mathbf{q})^T \mathbf{T}_p \mathbf{M}_{\frac{1}{2}} \mathbf{T}_p^T (\mathbf{p} - \mathbf{q})}$. Using these transformations we obtain the vertex's smoothed location $\tilde{\mathbf{p}}$ within the tangent plane at p and then project $\tilde{\mathbf{p}}$ to the null space at p (which is the same as the tangent plane at smooth vertices but is the tangent line at ridges) when mapping back to \mathbb{R}^3 .

In our overall smoothing procedure, we first compute the metric tensors at all vertices and then compute the new position for all vertices using a Jacobi-style iteration, which is easy to parallelize in numerical simulations and also avoids recomputing the metric tensors after moving each vertex. Correction steps may be performed to add correction terms to preserve the shape [22]. Note that in principle any averaging scheme may be plugged into our smoothing procedure to compute the new position within the tangent space, but some schemes may exhibit instability when using a Jacobi-style iteration so may not be suitable in our setting.

3.3 Anisotropic Edge Flipping

Edge flipping is a commonly used operation in meshing algorithms such as Delaunay triangulation. For each edge uv with opposite vertices p and q , we flip uv if the Delaunay flipping criterion (i.e., $\angle upv + \angle uqv > \pi$) is satisfied in its parametric space. A key question is how to define this parametric space. Unlike vertex redistribution, where a vertex is at the center of the geometric support of the computation, it is unnatural to choose any vertex as reference for edge flipping. Because of the integral nature of quadric metric tensors, we obtain a reference frame by summing up the quadric metric tensors at the four vertices, and then obtain the reference frame and Riemannian tensor from its null space. The origin of the reference frame is positioned at the average of the four points. This approach avoids biasing toward any vertex. After obtaining the tensor one can compute the angles in the parametric space or use the modified Delaunay criterion [7]. Our algorithm repeatedly flips the edges using this modified Delaunay criterion until convergence.

If the metric tensor is a constant, then this procedure would converge to the Delaunay triangulation within the parametric space independent of the flipping sequence. For general surfaces this process may not converge to a Delaunay mesh and may even run into an infinite loop due to the potential inconsistencies caused by discretization errors. We use a greedy strategy to flip edges in decreasing order of maximum opposite angle, avoiding infinite loops by flipping edges only once. For meshes with sharp features we prohibit flipping any edge that is marked as a ridge edge (which are identified also using the quadric metric tensor [22]) because such a flipping introduces large errors.

4 Anisotropy-Aware Mesh Repair

Vertex redistribution and edge flipping optimize a mesh with a given number of vertices and under other geometric constraints (such as boundary and features). If a dynamic surface undergoes severe expansion or contraction, the number of vertices may need to be increased or decreased to maintain the overall density. Furthermore, there may be some small or large angles that can restrict time steps or lead to arbitrarily large errors in the normal directions [36]. These bad angles may not be resolved by mesh optimization due to geometric constraints. We address these issues using a process referred as *mesh repair*. Mesh repair does not attempt to optimize any quality measures but focuses on safeguarding and resolving pathological cases. Our overall adaptation strategy is to iterate between mesh optimization and mesh repair. Mesh repair is critical for generality and robustness but is also difficult to analyze. To keep it simple, we choose the two simplest operations: edge splitting and edge contraction. These repair operations must be anisotropy-aware so that they do not undo the effect of anisotropic mesh optimization.

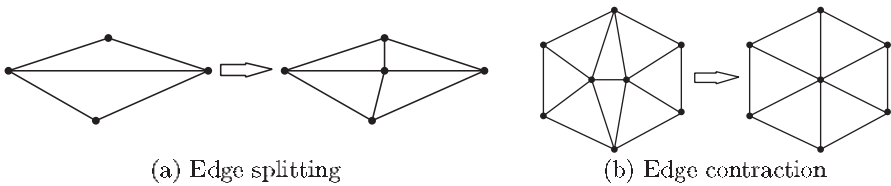


Fig. 1. Operations used in anisotropy-aware mesh repair

4.1 Edge Splitting

Edge splitting inserts a new vertex to an edge, as illustrated in Fig. 1(a). It helps to refine the area where the mesh may be too coarse, and to eliminate large angles with a long opposite edge. We choose the edges to split based on the following two criteria:

Absolute longness: the edge is the longest in its incident triangles and is longer than a given threshold L , or

Relative longness: the edge is longer than a desired edge length $l < L$, one of its opposite angles is close to π (greater than a threshold θ_l), and the shortest edge in its incident triangles is no shorter than s (where $s < l$).

The four parameters above must be chosen in a way consistent with edge contraction, which we consider in Sect. 4.3. In the second criterion the constraint on s is necessary to prevent over-refinement caused by splitting too small triangles. Note that this process leaves out some large angles, which will

be addressed by edge contraction. We split the edges in decreasing order of edge lengths. After splitting an edge the new vertex is first positioned at the edge center, and then projected along the normal direction to the point that minimizes a quadric error metric along the edge to preserve smoothness.

4.2 Edge Contraction

Edge contraction merges two adjacent vertices into a single vertex, and it helps to coarsen the mesh and to eliminate some very small angles (and also some large angles). Edge contraction is more difficult than edge splitting because it can potentially smear features and even cause mesh folding. We determine whether an edge is desirable to contract using the following criteria:

Absolute small angle: its opposite angle in an incident triangle is smaller than a threshold θ_s , and the longest edge of the triangle is shorter than the desired edge length l , or

Relative shortness: it is shorter than a small fraction r of the longest edge in its incident triangles, or

Absolute small triangle: the longest edge in its incident faces is shorter than a given threshold S , and it is the shortest edge in its incident triangles, or

Relative small triangle: the longest edge in its incident faces is shorter than the desired edge length l , and it is shorter than a fraction R of the longest edges in both physical and normalized space.

The first two criteria address poorly-shaped triangles with a very small angle or a short edge, and as a side product eliminate some triangles with very large angles. The last two criteria address well-shaped but too small triangles to decrease vertex density for mesh coarsening.

We contract edges in increasing order of edge lengths. To preserve features during contraction, if two vertices have different feature ranks (a smooth, ridge, and corner vertex has a feature rank of 0, 1, or 2, respectively) we place the merged vertex at the original vertex with the higher rank. If two vertices have the same rank, to preserve smoothness we obtain a weighted average of the original vertices and then project it along the normal direction to minimize the quadric error metric. To prevent mesh folding we reject contractions that would lead to topological changes or normal inversion of any face. We observe that such violations rarely occur as we contract shortest edges first.

4.3 Parameter Selection

Mesh repair requires a number of parameters as it deals with pathological cases by nature. The interplay among different parameters are quite complex. Overall, we have three different types of parameters: edge lengths (l , L , s , S), edge-length ratios (r , R), and angles (θ_l and θ_s).

For edge-length parameters, in general $s < S < l < L$, since S and L specify desirable lengths for edge contracting and splitting, and s/l is closely

related to the maximum aspect ratio allowed by anisotropic transformation (i.e., $\sqrt{\psi_u/\psi_l}$). For the edge-length ratio r and R , in general $r < R$ as they correspond to poorly-shaped and well-shaped triangles, respectively. The ratio r is related to the ratio of s/l in the relative-longness criterion, and in general $r \ll s/l \approx \sqrt{\psi_l/\psi_u}$. The threshold R is related to S , and we choose $R \approx S/l$.

For the angle thresholds, $\theta_s \ll \theta_l$, and it is desirable that $2\theta_s + \theta_l > 180^\circ$, so that large angles incident on a relatively short edge would be eliminated by edge contraction. The angle θ_s is also related to the thresholds ψ_l and ψ_u . The maximum of the minimum angle of a triangle contained in an ellipse with aspect ratio $\rho = \sqrt{\psi_u/\psi_l}$ is $2 \arctan \rho$, and therefore we choose $\theta_s \approx 2 \arctan \rho$.

Based on these consideration and extensive experiments, we choose the following default values for the parameters:

- $\psi_l = \lambda_1 \tan^2(4^\circ)$ and $\psi_u = \lambda_1 \tan^2(15^\circ)$;
- $r = 0.1$, $R = 0.5$;
- $s = l\sqrt{\psi_l/\psi_u}$, $S = Rl$, $L = 1.5l$;
- $\theta_s = 2 \arctan \sqrt{\psi_l/\psi_u} \approx 15^\circ$ and $\theta_l = 160^\circ$.

In general the desired edge length l may vary in space, but it typically suffices to have a uniform value as the desired average edge length. In the following we present some experimental results, all of which used the above default values.

5 Experimental Results

We present some preliminary results of anisotropic adaptation for static and dynamic surfaces to demonstrate its effectiveness, and compare it with isotropic remeshing for dynamic surfaces.

5.1 Remeshing Static Surfaces

When applied to a static surface our anisotropic adaptation algorithm essentially becomes a remeshing tool. We show two simple examples of static remeshing to demonstrate the effect of anisotropic adaptation. In the first example we remesh a small cube. As shown in Fig. 2, a layer of small triangles was formed around sharp features.

In the second example we remesh a surface mesh with corrupted sharp features, as shown in Fig. 3. We adapt the mesh while adding a normal motion to denoise the surface. The detail of this denoising procedure is beyond the scope of this paper, but its basic idea is to identify the noisy vertices similar to feature detection using the eigenvalues of the quadric metric tensor, and then add a normal motion in a volume-preserving fashion similar to the normal-diffusion approach of Ohtake et al. [33]. As obvious from the figures, the final mesh is anisotropic near high-curvature regions and both the mesh quality and surface geometry were improved substantially.

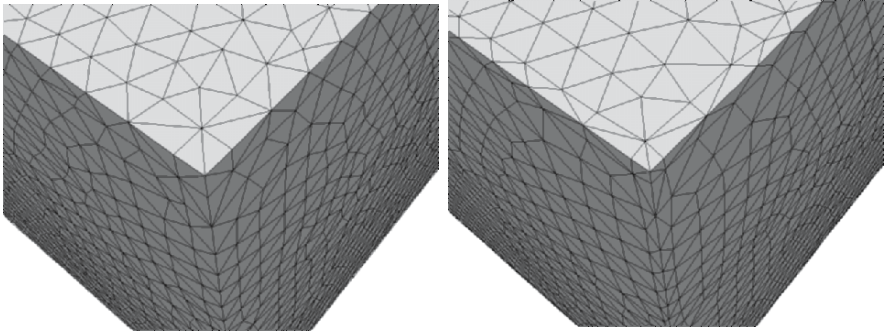


Fig. 2. Remesh cube with anisotropic mesh optimization

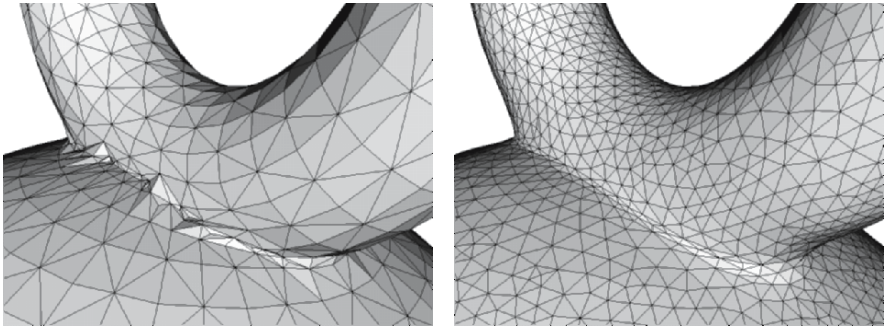


Fig. 3. Adapting a surface with corrupted features using anisotropic mesh adaptation

5.2 Adaptation of Dynamic Surfaces

As discussed earlier, the advantages for anisotropic surface adaptation are most prominent in adapting dynamically moving surfaces. We use a series of dynamic surfaces with an increasing level of difficulty to verify our assertion. In particular, we adapt a surface that is advected in two challenging flow fields for a time period T as detailed later. The larger T is the more severe the deformation becomes. We modulate time by the cosinusoidal function $\cos(\pi t/T)$ to make the flow periodic, so that in principle the shape at time $t = 0$ and $t = T$ should be identical. Such a test has been widely used to test dynamic surfaces or moving interfaces [4, 14, 16, 27]. In these tests, we propagate each vertex using the fourth-order Runge-Kutta integration scheme and then adapt the surface anisotropically. The time step was controlled using the approach of Jiao [21] to prevent mesh folding. We perform anisotropic mesh smoothing at every time step and invoke the full-fledged anisotropic adaptation every few iterations or when the time step becomes too small.

5.2.1 Modest Deformation

Dynamic surfaces pose significant challenges to mesh adaptation, and most traditional adaptation techniques lead to large errors or abrupt failure at a very early stage. In order to compare our method with existing adaptation techniques, we first consider the modest deformation of a sphere of radius 0.15 centered at $(0.5, 0.75, 0.5)$ in a vortex flow with a time period of $T = 2$. The spatial velocity of this deformation field is given by

$$u(x, y, z) = \sin^2(\pi x)(\sin(2\pi z) - \sin(2\pi y)), \quad (13)$$

$$v(x, y, z) = \sin^2(\pi y)(\sin(2\pi x) - \sin(2\pi z)), \quad (14)$$

$$w(x, y, z) = \sin^2(\pi z)(\sin(2\pi y) - \sin(2\pi x)). \quad (15)$$

This flow is a challenge as large curvatures develop at the maximum deformation. In this test we compare the results of using anisotropic mesh optimization with the isotropic remeshing, in particular our own implementation of the method of Alliez et al. [2]. In both cases we used a time step of 0.01. Figure 4 shows the meshes using anisotropic adaptation at times $t = 0, 1$, and 2 using a relative coarse initial mesh with 10,784 vertices and 21,564 triangles. Figure 5 shows the results using isotropic remeshing with uniform spacing using the same initial mesh. The anisotropic results are obviously far superior to the isotropic ones. Quantitatively, the volume loss for anisotropic adaptation was less than 0.1% compared to about 24% for isotropic remeshing. Note that the isotropic remeshing algorithm of Alliez et al. [2] can adapt vertex density based on curvatures, but we observed some numerical instability and worse results when adapting vertex density based on Gaussian or mean curvatures, probably because the curvatures are inherently sensitive to perturbation.

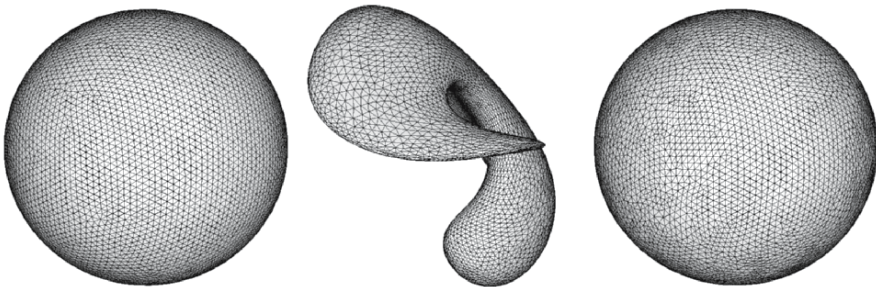


Fig. 4. Results of vortex flow using our anisotropic adaptation at $t = 0, 1$, and 2

In terms of efficiency, anisotropic adaptation took 9 minutes to complete the whole simulation on a PC with 3.2 GHz Pentium D processor. In comparison, the remeshing algorithm by Alliez et al. [2] is very expensive and takes several minutes even for a single remeshing step; therefore, we adopted

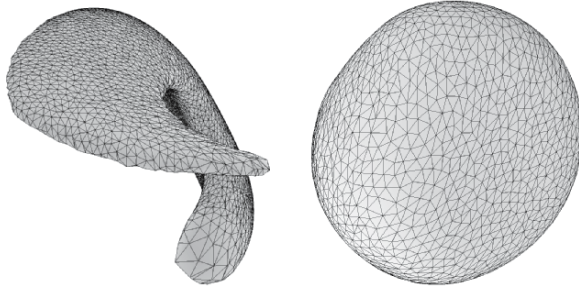


Fig. 5. Results using isotropic remeshing at times at $t = 1$, and 2 for the same flow and initial mesh as in Fig. 4

an approximation solver proposed by Ostromoukhov et al. [34] to make their algorithm more competitive. Even after this speed-up, isotropic remeshing took 30 seconds per remeshing step, which amounts to about 10 times slower than our technique. This relatively simple example shows the effectiveness and efficiency of anisotropic adaptation and demonstrates the challenges in remeshing dynamic surfaces.

5.2.2 Large Deformation

In the literature, another widely used test has a velocity field

$$u(x, y, z) = 2 \sin^2(\pi x) \sin(2\pi y) \sin(2\pi z), \quad (16)$$

$$v(x, y, z) = -\sin(2\pi x) \sin^2(\pi y) \sin(2\pi z), \quad (17)$$

$$w(x, y, z) = -\sin^2(2\pi x) \sin^2(2\pi y) \sin(\pi z), \quad (18)$$

which advects a sphere centered at $(0.35, 0.35, 0.35)$ for a time period of $T = 3$. This problem is sometimes referred to as the Enright test and has been solved using implicit surfaces, such as the level set method and its variations [16, 17] and hybrid front tracking methods [5, 14]. Note that if one simply propagates the vertices of a surface mesh independently, some triangles would become inverted very soon, so mesh adaptation is necessary.

We used an initial mesh with 23,238 vertices and 46,472 triangles with a time step of 0.015, so the whole computation took 200 iterations. Due to distortions introduced by the flow, anisotropic optimization alone cannot meet this challenge, so we used the full-fledged anisotropic mesh adaptation. We invoke anisotropic adaptation every 4 time steps. This flow is mildly unstable during the second half period, so a smoothing term similar to that in Fig. 3 was added to denoise the surface.

Figure 6 shows the surfaces with anisotropic mesh adaptation after 50, 100, 150, and 200 time steps. At the maximum deformation (i.e., $t = 1.5$) the surface area increased by a factor of 4.12, and the numbers of vertices and

triangles also roughly quadrupled (increased to 98,638 and 197,292, respectively). At time $t = 3$ the surface returned back to a nearly perfect sphere, and the errors in both the volume and surface area were less than 0.1%. Note that the popular level-set method lost 80% of volume for this test, while the much-improved particle level-set method of Enright et al. [16] lost 2.6% when using one million grid points. Because of the lower time complexity of our surface mesh-based scheme, the computation time of our method is expected to be orders of magnitude smaller than that of Enright et al. [16].

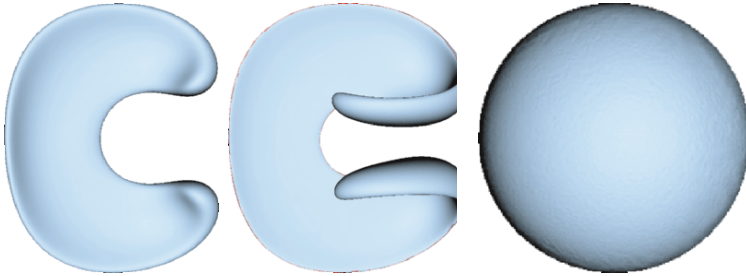


Fig. 6. Solution of Enright test with anisotropic mesh adaptation

5.2.3 Very Large Motion

The motion in the previous test is large, but it is still relatively simple because the surface remained smooth and the time period was relatively short. We test our method using a more challenging problem with the same flow as in Sect. 5.2.1 but a longer period of $T = 6$. Under this flow the sphere swirls for three cycles at the maximum deformation, forming cusps and extremely thin filaments, posing significant challenges to represent the surface accurately. To the best of our knowledge no solution to this problem has been reported previously in the literature, except for a result of $T = 4$ using a hybrid surface-marker and volume-of-fluid method [4]. Our simulation used a time step of 0.015 for 400 iterations using an initial mesh same as that in the previous test. Figure 7 shows the surface after 100, 200, and 400 time steps, respectively. At the maximum deformation the area increased by a factor of 5.6, and the numbers of vertices and triangles increased by a factor of 5.4. At time $t = 6$, the volume error was about 0.3%.

6 Conclusion

In this paper we present an effective approach for anisotropic adaptation of triangulated surfaces, with a focus on adapting dynamic surfaces that are the

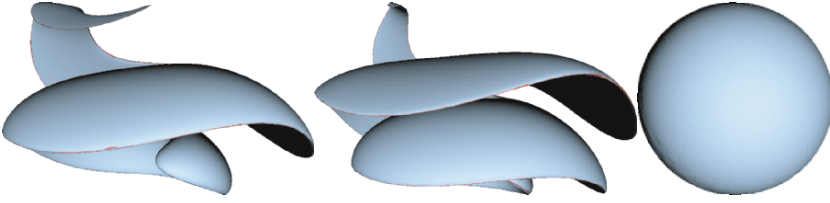


Fig. 7. Solution of vortex flow with anisotropic adaptation for period $T = 6$

solutions of numerical simulations. This setting poses significant challenges in accuracy and efficiency. We present an extension of the quadric-based surface analysis to treat singularities and in turn deliver a unified framework for resolving smooth surfaces, sharp features, and the ambiguities between them. We propose a simple and efficient transformation for anisotropic mesh adaptation with built-in safeguards for degeneracies, and use this transformation to optimize a mesh anisotropically. We also develop a mesh repair strategy to address pathological cases. The effectiveness of anisotropic adaptation was demonstrated using a number of examples, and orders of magnitude of improvements were achieved in accuracy and efficiency for dynamic surfaces compared to adapting the surface meshes isotropically or representing and propagating the surfaces using Eulerian methods. A number of research issues remain open for dynamic surfaces, including accurate resolution of topological changes of surface meshes and volume conservation in full-fledged mesh adaptation, which we are investigating and plan to report in the future.

Acknowledgement

This work was supported by the U.S. Department of Energy through the University of California under subcontract B523819 with the University of Illinois at Urbana-Champaign, and by NSF and DARPA under CARGO grant #0310446. We thank the anonymous reviewers for their helpful comments on improving the exposition of this work.

References

- [1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22:485–493, July 2003.
- [2] P. Alliez, E. Colin de Verdière, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In *Proc. Shape Modeling Intl.*, 2003.
- [3] R. Almeida, P. Feijoo, A. Galeao, C. Padra, and R. Silva. Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Comput. Methods Appl. Mech. Engrg.*, 182:379–400, 2000.

- [4] E. Aulisa, S. Manservigi, and R. Scardovelli. A surface marker algorithm coupled to an area-preserving marker redistribution method for three-dimensional interface tracking. *J. Comput. Phys.*, 197:555–584, 2004.
- [5] A. Bargteil, T. Goktekin, J. O’Brien, and J. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25, 2006.
- [6] H. Borouchaki and P. Frey. Adaptive triangular-quadrilateral mesh generation. *Int. J. Numer. Meth. Engr.*, 41:915–934, 1998.
- [7] F. J. Bossen and P. S. Heckbert. A pliant method for anisotropic mesh generation. In *Proc. 5th Int. Meshing Roundtable*, pages 63–74, Oct. 1996.
- [8] G. Buscaglia and E. Dari. Anisotropic mesh optimization and its application in adaptivity. *Int. J. Numer. Meth. Engr.*, 40:4119–4136, 1997.
- [9] H. Cheng, T. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. *Disc. Comput. Geom.*, 25:525–568, 2001.
- [10] S.-W. Cheng, T. K. Dey, E. A. Ramos, and R. Wenger. Anisotropic surface meshing. In *Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pages 202–211, 2006.
- [11] E. D’Azevedo and R. Simpson. On optimal interpolation triangle incidences. *SIAM J. Sci. Stat. Comput.*, 10:1063–1075, 1989.
- [12] E. D’Azevedo and R. Simpson. On optimal triangular meshes for minimizing the gradient error. *Numer. Math.*, 59:321–348, 1991.
- [13] V. Dolejsi. Anisotropic mesh adaptation for finite volume and finite element methods on triangular meshes. *Comput. Vis. Sci.*, 1:165–178, 1998.
- [14] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *J. Comput. Phys.*, 213:613–628, 2006.
- [15] Q. Du and D. Wang. Anisotropic centroidal Voronoi tessellations and their applications. *SIAM J. Sci. Comput.*, 26:737–761, 2005.
- [16] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183:83–116, 2002.
- [17] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. *Comput. Struc.*, 83:479–490, 2005.
- [18] P. Frey. About surface remeshing. In *Proc. 9th Int. Meshing Roundtable*, pages 123–136, Oct. 2000.
- [19] W. G. Habashi, J. Dompierre, Y. Bourgault, D. Ait-Ali-Yahia, M. Fortin, and M.-G. Vallet. Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. Part I: General principles. *Int. J. Numer. Meth. Fluids*, 32:725–744, 2000.
- [20] P. S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Comput. Geom.*, pages 49–65, 1999.
- [21] X. Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys.*, 2006. To appear.
- [22] X. Jiao. Volume and feature preservation in surface mesh optimization. In *Proc. 15th Int. Meshing Roundtable*, 2006.
- [23] X. Jiao and P. Alexander. Parallel feature-preserving mesh smoothing. In *Proc. Int. Conf. Comput. Sci. Appl.*, pages 1180–1189, 2005.
- [24] E. Kreyszig. *Introduction to Differential Geometry and Riemannian Geometry*, volume 16 of *Mathematical Expositions*. University of Toronto Press, 1968.
- [25] F. Labelle and J. R. Shewchuk. Anisotropic Voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proc. 19th Ann. ACM Sympos. Comput. Geom.*, pages 191–200. ACM, 2003.

- [26] M. Lesoinne and C. Farhat. Geometric conservation laws for flow problems with moving boundaries and deformable meshes and their impact on aeroelastic computations. *Comput. Methods Appl. Mech. Engrg.*, 134:71–90, 1996.
- [27] R. Leveque. High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal.*, 33:627–665, 1996.
- [28] X. Li, M. S. Shephard, and M. W. Beall. 3d anisotropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.*, 194:4915–4950, 2005.
- [29] S. Lo. 3-d anisotropic mesh refinement in compliance with a general metric specification. *Finite Elements Anal. Design*, 38:3–19, 2001.
- [30] M. Meyer, M. Desbrun, P. Schroder, and A. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–58, 2003.
- [31] E. Nadler. Piecewise linear best L_2 approximation on triangulations. In C. K. C. et al., editor, *Approximation Theory V*, pages 499–502. Academic Press, 1986.
- [32] J. Oden, T. Strouboulis, and P. Devloo. Adaptive finite element methods for high-speed compressible flows. *Int. J. Numer. Meth. Fluids*, 7:1211–1228, 1987.
- [33] Y. Ohtake, A. G. Belyaev, and H.-P. Seidel. Mesh smoothing by adaptive and anisotropic Gaussian filter. In *Vision, Modeling and Visualization*, pages 203–210, 2002.
- [34] V. Ostromoukhov, C. Donohue, and P.-M. Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.*, 23(3):488–495, 2004.
- [35] J. Peraire, M. Vahdati, K. Morgan, and O. Zienkiewicz. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.*, 72:449–466, 1987.
- [36] J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proc. 11th Int. Meshing Roundtable*, pages 115–126, 2002.
- [37] K. Shimada, A. Yamada, and T. Itoh. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *Proc. 6th Int. Meshing Roundtable*, pages 375–390, 1997.
- [38] R. B. Simpson. Anisotropic mesh transformations and optimal error control. *Applied Numer. Math.*, 14:183–198, 1994.
- [39] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comput. Phys.*, 169:708–759, 2001.

Multi-Dimensional Continuous Metric for Mesh Adaptation

Frédéric Alauzet¹, Adrien Loseille¹, Alain Dervieux², and Pascal Frey³

¹ INRIA-Gamma, Domaine de Voluceau, BP 105, 78153 Le Chesnay cedex, France

`Frederic.Alauzet@inria.fr` and `Adrien.Loseille@inria.fr`

² INRIA-Tropics, BP 93, 06902 Sophia-Antipolis, France

`Alain.Dervieux@inria.fr`

³ UMPC/Lab. J-L Lions, BP 187, 75252 Paris Cedex 05, France

`frey@ann.jussieu.fr`

Summary. Mesh adaptation is considered here as the research of an optimum that minimizes the \mathcal{P}_1 interpolation error of a function u of \mathbb{R}^n given a number of vertices. A continuous modeling is described by considering classes of equivalence between meshes which are analytically represented by a metric tensor field. Continuous metrics are exhibited for \mathbf{L}^p error model and mesh order of convergence are analyzed. Numerical examples are provided in two and three dimensions.

Key words: Continuous metric, metric tensor, mesh adaptation, anisotropy, interpolation error, order of convergence

1 Introduction

Nowadays, in the context of numerical simulations based on finite elements or finite volumes methods, unstructured mesh adaptation has largely proved its efficiency for improving the accuracy of the numerical solution as well as for capturing the behavior of physical phenomena. In principle, this technique allows (i) to substantially reduce the number of degrees of freedom, thus impacting favorably the cpu time and (ii) to automatically capture the anisotropy of the physical phenomena. However, such efficiency is usually observed from the practical point of view only, as a thorough theoretical analysis is really tedious to carry out on unstructured meshes. Indeed, there is no simple Hilbert space structure for the type of non-isotopological meshes that are required for a variational study. To overcome this difficulty, we represent meshes with continuous functions describing them. To this end, the concept of continuous metric, introduced by Dervieux et al. [19, 7], is used to replace the notion of a mesh in a variational analysis. A continuous metric is simply a continuous function associating a metric tensor to each vertex of the domain.

Over the last few years, numerous papers have been published concerning mesh adaptation in numerical simulations. It points out that isotropic mesh adaptation has been already well addressed in two and three dimensions, see for instance [12, 2, 18, 22, 24, 25, 21, 27]. Regarding anisotropic mesh adaptation, numerous works using (or implicitly using) the concept of metric in order to equally distribute the interpolation error have been published in two dimensions [1, 3, 4, 10, 9, 17]. However, only a few of them have dealt with the three-dimensional case [13, 16, 20, 28, 23].

Recently, some papers have proposed a new approach to define an optimal metric in two dimensions for minimizing the interpolation error in norm \mathbf{L}^p in order to generate anisotropic adapted meshes [26, 6, 5]. Formally speaking, let u be an analytic solution defined on a bounded domain Ω and let N denotes the desired number of vertices for the mesh, the aims is to create the “best” mesh \mathcal{H} , i.e., the best continuous metric \mathcal{M} , to minimize the interpolation error $\|u - \Pi_h u\|_p$ in \mathbf{L}^p norm, $\Pi_h u$ being the linear interpolate of u on \mathcal{H} . To this end, a model of the interpolation error $e_{\mathcal{M}}$ is required. Once $e_{\mathcal{M}}$ has been properly defined, a calculus of variation is performed on the domain Ω . Mathematically, we need to solve the following minimization problem:

$$\text{find } \mathcal{E}(\mathcal{M}) \text{ such that } \min_{\mathcal{M}} \int_{\Omega} |e_{\mathcal{M}}(\mathbf{x})|^p \, d\mathbf{x}. \quad (1)$$

The aim of this paper is to extend the results presented in two dimensions in [6] in any dimension. Firstly, we indicate how we get our local error model $e_{\mathcal{M}}$ based on a bound of the interpolation error. We also demonstrate that the optimal directions of the desired metric coincide with the directions of the Hessian of the solution. Secondly, to extend the result in dimension n , we propose a definition of anisotropic quotients. A coordinate transformation is performed with these quotients to simplify the resolution of Problem (1) by a calculus of variation. Finally, we analyze the order of convergence of the error with respect to the obtained optimal metric.

Analytical examples in two and three dimensions are given to illustrate the impact of the continuous metric throughout a mesh adaptation process.

2 Metric Notions

The notion of length in a metric space is closely related to the notion of metric and subsequently to the definition of the scalar product in the vector space. When this metric is continuously defined over the whole domain, it is called a *continuous metric*. In the following, the natural scalar dot product of \mathbb{R}^n is denoted by $\langle \cdot, \cdot \rangle$.

Metric Definition

A *metric tensor* (or, more simply, a *metric*) \mathcal{M} in \mathbb{R}^n is a $n \times n$ symmetric strictly definite positive matrix; hence \mathcal{M} is always diagonalizable.

From this definition, it follows up that the *scalar product* of two vectors in \mathbb{R}^n is defined with respect to a metric \mathcal{M} as:

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{M}} = \langle \mathbf{u}, \mathcal{M}\mathbf{v} \rangle = {}^t\mathbf{u}\mathcal{M}\mathbf{v} \in \mathbb{R}.$$

Under this notion, the Euclidean *norm* of a vector in \mathbb{R}^n is easily defined as:

$$\|\mathbf{u}\|_{\mathcal{M}} = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{M}}} = \sqrt{{}^t\mathbf{u}\mathcal{M}\mathbf{u}},$$

that actually measures the length of vector \mathbf{u} with respect to metric \mathcal{M} .

A metric \mathcal{M} could be geometrically represented by its associated unit ball, an ellipsoid, defined as:

$$\mathcal{E}ll_{\mathcal{M}} = \{b \mid \sqrt{{}^t\vec{ab}\mathcal{M}\vec{ab}} = 1\}$$

where a denotes the center of the ellipsoid. The main axes are given by the eigenvectors of matrix \mathcal{M} and the radius along each axis is given by the inverse of the square root of the associated eigenvalues.

In the following sections, the metric unit ball is essentially used to define neighborhoods. We denote by $\mathcal{B}_{\mathcal{M}}(a)$ the unit ball of a in metric \mathcal{M} , also defined as:

$$\mathcal{B}_{\mathcal{M}}(a) = \{b \in \mathbb{R}^n \mid d_{\mathcal{M}}(a, b) \leq 1\}.$$

More details on these notions can be found in [14].

Continuous Metric

Let $\Omega \subset \mathbb{R}^n$ be the computational bounded domain. Defining a continuous metric on Ω is equivalent to define an Euclidean space supplied with a Riemannian metric $(\Omega, \mathcal{M}(\cdot))$, where $\mathcal{M}(\cdot)$ is the continuous metric. In this case, the distance between two points a and b is given by the integral:

$$d_{\mathcal{M}}(a, b) = \int_0^1 \sqrt{{}^t\vec{ab}\mathcal{M}(\gamma(t))\vec{ab}} dt, \quad (2)$$

where $\gamma(t) = a + t\mathbf{ab}$ is a normal parametrization of the arc ab .

A mesh is called a *unit mesh* with respect to the continuous metric $\mathcal{M}(\cdot)$ if all its edges have a length strictly equal (or very close) to one in the continuous metric $\mathcal{M}(\cdot)$, as given by Relation (2) and if all its elements are (almost) regular.

This notion of a continuous metric is a trick that allows us to forsake the mesh in the analysis. Indeed, a mesh generator governed by a metric tensor field makes use of the distance criterion specified by the continuous metric. Therefore, two different *unit meshes* with respect to this metric can be considered as strictly equivalent. This metric defines a class of equivalence between meshes. In this respect, the mesh becomes an unknown of the problem with respect to a continuous metric.

Notations

In the following, a continuous metric \mathcal{M} defines for any point a of the domain $\Omega \in \mathbb{R}^n$ a matrix $\mathcal{M}(a)$. For sake of simplicity we omit a in our notations, we denote by \mathcal{M} instead of $\mathcal{M}(a)$ the continuous metric. This continuous metric is always diagonalized. We define the following functions as unknowns of our problem:

- let $\mathcal{R}_{\mathcal{M}}$ be the function associating to each point a of the domain $\Omega \in \mathbb{R}^n$ the coordinate transformation matrix of $\mathcal{M}(a)$. $\mathcal{R}_{\mathcal{M}}$ is composed with the eigenvectors $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$ of \mathcal{M} .
- let be $\lambda_1, \lambda_2, \dots, \lambda_n$ the n functions associating to each point a of the domain the eigenvalues of $\mathcal{M}(a)$. Or similarly, we could denote by h_1, h_2, \dots, h_n the local size functions defined by $h_i = \sqrt{1/\lambda_i}$ for $i = 1, \dots, n$.

For a unit mesh, an edge parallel to \mathbf{v}_i should be, according to (2), of length equal to $h_i = \sqrt{1/\lambda_i}$, where h_i is the local mesh size in direction \mathbf{v}_i .

If the distance in a metric space is defined by a metric, the latter also indicates the vertex density (or distribution) over the domain directly from the local sizes $(h_i)_{i=1..n}$. Indeed, let d be the mesh density, defined as $d = \prod_{i=1}^n h_i^{-1}$, then the number of vertices $\mathcal{C}(\mathcal{M})$ of the mesh, or *the mesh complexity*, is given by:

$$C(\mathcal{M}) = \int_{\mathbf{x} \in \Omega} d(\mathbf{x}) \, d\mathbf{x} = \int_{\mathbf{x} \in \Omega} \prod_{i=1}^n \frac{1}{h_i}(\mathbf{x}) \, d\mathbf{x}.$$

Let u be a C^2 continuous function, we denote its Hessian by $H_u = \left(\frac{\partial u}{\partial x_i \partial x_j} \right)_{i,j}$.

It can be decomposed as follows:

$$H_u = \mathcal{R}_u \Lambda \mathcal{R}_u^{-1} = \mathcal{R}_u \text{diag} \left(\frac{\partial^2 u}{\partial \alpha_i^2} \right) \mathcal{R}_u^{-1},$$

where \mathcal{R}_u is formed by the eigenvectors of H_u denoted $(\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n)$. If the diagonal matrix Λ has all non-zero terms, then the matrix $|H_u|$ is a metric tensor defined as:

$$|H_u| = \mathcal{R}_u |\Lambda| \mathcal{R}_u^{-1} = \mathcal{R}_u \text{diag} \left(\left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| \right) \mathcal{R}_u^{-1}. \tag{3}$$

3 Local Error Modeling

In this section, the local error $e_{\mathcal{M}}(a)$ in the neighborhood of a vertex a is designed. This error is evaluated in the neighborhood $\mathcal{B}_{\mathcal{M}}(a)$ of a defined by the continuous metric \mathcal{M} . We first express this error in terms of the discrete error e_K , the interpolation error in \mathbf{L}^∞ norm on an element K of mesh \mathcal{H} considered as a representative of the continuous metric \mathcal{M} . Next, we demonstrate

that this error is maximal when its main directions $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$ are aligned with the main directions of the Hessian $(\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n)$. Then, we obtain the following local error model:

$$e_{\mathcal{M}}(a) = \sum_{i=1}^n h_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|.$$

3.1 Local Error Definition

Let a be a point of domain Ω and let $\mathcal{B}_{\mathcal{M}}(a) = \{\mathbf{x} \in \Omega \mid d_{\mathcal{M}}(\mathbf{x}, a) \leq 1\}$ denotes its unit ball. The aim is to control the local error $e_{\mathcal{M}}$ in the vicinity of a defined as:

$$e_{\mathcal{M}}(a) = \max_{\mathbf{x} \in \mathcal{B}_{\mathcal{M}}(a)} |u(\mathbf{x}) - \Pi_h u(\mathbf{x})|.$$

Practically, we need a discrete support to be able to compute this error. To follow up on this idea, we consider a mesh \mathcal{H} as a member of the class of equivalence defined by \mathcal{M} and a as a vertex of this mesh. As the mesh is represented by \mathcal{M} , the unit ball $\mathcal{B}_{\mathcal{M}}(a)$ is represented on \mathcal{H} by $\mathcal{B}_h(a)$, its ball in the mesh, i.e., the set of all mesh vertices connected to the vertex a . Indeed, \mathcal{H} is a unit mesh for \mathcal{M} , then the lengths of all edges ab , $b \in \mathcal{B}_h(a)$ are equal to one with respect to the metric. Consequently, we propose the following model for the local error on the mesh \mathcal{H} :

$$e_{\mathcal{M}}(a) = \max_{K \in \mathcal{B}_h(a)} \|u - \Pi_h u\|_{\infty, K} = \max_{K \in \mathcal{B}_h(a)} e_K. \quad (4)$$

Therefore, we need to compute the interpolation error on an element K in \mathbf{L}^∞ norm to model the local error. This will be explained in the next section.

3.2 The Discrete Case

In this section, a geometric error estimation of the interpolation error in \mathbf{L}^∞ norm is presented. This error estimate will be used hereafter to construct our error model. Here, we will focus exclusively on the three-dimensional case.

We consider a tetrahedral unstructured mesh with the following notations:

- $K = [a, b, c, d]$ is a tetrahedron with a diameter not necessary supposed small
- $u : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a (regular) function representing the solution of our problem
- $\Pi_h u$ is the linear interpolate of u on the element K defined by the parameterization $\Pi_h u = (1 - \lambda - \mu - \nu)u(a) + \lambda u(b) + \mu u(c) + \nu u(d)$, with $0 \leq \lambda + \mu + \nu \leq 1$.

The aim is to bound the error $e = u - \Pi_h u$ on K . To this end, we rely on a Taylor expansion with an integral rest of the function e at a vertex of K (for instance a) with respect to any interior point x in K :

$$(u - \Pi_h u)(a) = (u - \Pi_h u)(x) + \langle \overrightarrow{x\bar{a}}, \nabla(u - \Pi_h u)(x) \rangle + \int_0^1 (1-t) \langle \overrightarrow{a\bar{x}}, H_u(x + t\overrightarrow{x\bar{a}}) \overrightarrow{a\bar{x}} \rangle dt,$$

where $\nabla u(x)$ and $H_u(x)$ denote the gradient and the Hessian of the variable u at point x , respectively. Actually, as we assume that the maximal error is achieved at the location x (closer to a than to b, c or d), then $\nabla(u - \Pi_h u)(x) = 0$ and as u and $\Pi_h u$ coincide at the vertex a by definition, we get:

$$|e(x)| = \left| \int_0^1 (1-t) \langle \overrightarrow{a\bar{x}}, H_u(x + t\overrightarrow{x\bar{a}}) \overrightarrow{a\bar{x}} \rangle dt \right|.$$

Let a' representing the point corresponding to the intersection of the line ax with the face opposite to a . It exists a real number λ such that $\overrightarrow{a\bar{x}} = \lambda \overrightarrow{aa'}$. As a is closer to x than to any other vertex of K , $\lambda \leq 3/4$, it yields:

$$|e(x)| = \left| \int_0^1 (1-t) \lambda^2 \langle \overrightarrow{aa'}, H_u(a + t\overrightarrow{x\bar{a}}) \overrightarrow{aa'} \rangle dt \right|, \leq \frac{9}{16} \max_{y \in aa'} |\langle \overrightarrow{aa'}, H_u(y) \overrightarrow{aa'} \rangle| \left| \int_0^1 (1-t) dt \right|,$$

and then:

$$|e(x)| \leq \frac{9}{32} \max_{y \in K} |\langle \overrightarrow{aa'}, H_u(y) \overrightarrow{aa'} \rangle|.$$

At this point, we can introduce the \mathbf{L}^∞ norm of the interpolation error and consider the symmetric definite positive matrix $|H_u|$ (cf. Section 2). The following bound is obtained:

$$\|u - \Pi_h u\|_{\infty, K} \leq \frac{9}{32} \max_{y \in K} \langle \overrightarrow{aa'}, |H_u(y)| \overrightarrow{aa'} \rangle.$$

Notice that the previous relationship is not very useful in practice as the bound depends on the extremum x that is not known *a priori*. However, it can be reformulated in a more practical manner as follows:

$$e_K = \|u - \Pi_h u\|_{\infty, K} \leq \frac{9}{32} \max_{y \in K} \max_{\mathbf{v} \subset K} \langle \mathbf{v}, |H_u(y)| \mathbf{v} \rangle, \tag{5}$$

where $\mathbf{v} \subset K$ means that \mathbf{v} is a vector inside the element K , i.e., there exist two points $m, n \in K$ such that $\mathbf{v} = \overrightarrow{m\bar{n}}$.

This expression provides a bound in the case where the maximum error is achieved inside the element K . If the maximum error is obtained on the element face, then we obtain:

$$e_K = \|u - \Pi_h u\|_{\infty, K} \leq \frac{2}{9} \max_{y \in [a, b, c]} \max_{\mathbf{v} \subset [a, b, c]} \langle \mathbf{v}, |H_u(y)| \mathbf{v} \rangle.$$

Similarly, for a maximum value obtained along an element edge, we have:

$$e_K = \|u - \Pi_h u\|_{\infty, K} \leq \frac{1}{8} \max_{y \in ab} \langle \vec{ab}, |H_u(y)| \vec{ab} \rangle.$$

In conclusion, Relation (5) provides a convenient bound of the interpolation error on an element K .

Remark 1. This proof can be extended to any dimension n using a recurrence relation where the constant is given by:

$$c_n = \lambda^2 \left| \int_0^1 (1-t) dt \right| = \frac{1}{2} \left(\frac{n}{n+1} \right)^2.$$

3.3 Optimal Directions

From the previous section, the local error $e_{\mathcal{M}}$ in the neighborhood of a is defined by Relation (4) and for any simplex K of \mathbb{R}^n we have:

$$e_K = c_n \max_{y \in K} \max_{\mathbf{v} \subset K} \langle \mathbf{v}, |H_u(y)| \mathbf{v} \rangle \leq \max_{y \in K} \max_{\mathbf{v} \subset K} \langle \mathbf{v}, |H_u(y)| \mathbf{v} \rangle. \quad (6)$$

We note that right-hand side of Relation (6) is a second order estimate since $\|\mathbf{v}\|_2$ is smaller than the diameter of element K . In the proposed modeling, we neglect third order terms, which allows us to replace $\max_{y \in K} f(y)$ by $f(a)$ in the neighborhood $\mathcal{B}_{\mathcal{M}}(a)$. The right-hand side of Relation (6) is used to model the interpolation error e_K . In consequence, Equation (4) is written in a more simple form as:

$$e_{\mathcal{M}}(a) = \max_{K \in \mathcal{B}_h(a)} \max_{\mathbf{v} \subset K} \langle \mathbf{v}, |H_u(a)| \mathbf{v} \rangle.$$

The set of eigen-vectors $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)$ of \mathcal{M} forms a basis of \mathbb{R}^n . Hence, any vector \mathbf{v} inside $\mathcal{B}_h(a) \subset \mathcal{B}_{\mathcal{M}}(a)$ can be written in this basis such as $\mathbf{v} = \sum_{i=1}^n \nu_i \mathbf{v}_i$ which could be also written $\mathbf{v} = \sum_{i=1}^n \mu_i h_i \mathbf{v}_i$ with $\|\boldsymbol{\mu}\|_2 = \sum_{i=1}^n \mu_i^2 \leq 1$. Finally, from Expression (3), we have to compute the expression:

$$e_{\mathcal{M}}(a) = \max_{\|\boldsymbol{\mu}\|_2 \leq 1} \sum_{j=1}^n \left(\sum_{i=1}^n \mu_i h_i \langle \mathbf{v}_i, \mathbf{u}_j \rangle \right)^2 \left| \frac{\partial^2 u}{\partial \alpha_j^2} \right|. \quad (7)$$

Problem (7) could be written in a matrix form as follows:

$$e_{\mathcal{M}}(a) = \max_{\|X\|_2 \leq 1} (HPX)^T |A| HPX, \quad (8)$$

where

- X is the unknown vector of the μ_i
- H and $|A|$ are the diagonal matrices $\text{diag}(h_i)$ and $\text{diag}\left(\left|\frac{\partial^2 u}{\partial \alpha_i^2}\right|\right)$, respectively
- P is the transformation matrix from the eigen-basis of \mathcal{M} to that of $|H_u|$, defined by $P = (P_{ij})_{ij} = (\langle \mathbf{u}_i | \mathbf{v}_j \rangle)_{ij}$.

We consider the one to one variable substitution $Y = PX$. As the orthogonal matrix P preserves the norm, Problem (8) is equivalent to the following:

$$e_{\mathcal{M}}(a) = \max_{\|Y\|_2 \leq 1} (HY)^T |A| HY = \max_{\|Y\|_2 \leq 1} \sum_{i=1}^n h_i^2 y_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|, \quad (9)$$

here, the vector Y has been written as (y_1, y_2, \dots, y_n) . It yields:

$$\max_{\|Y\|_2 \leq 1} (HY)^T |A| HY \leq \max_{j=1..n} \left(h_j^2 \left| \frac{\partial^2 u}{\partial \alpha_j^2} \right| \right) \sum_{i=1}^n y_i^2.$$

Therefore, the constraint is active, i.e., the maximum is reached for $\|Y\| = 1$, and we get the following equality:

$$\max_{\|Y\|_2 \leq 1} (HY)^T |A| HY = \max_{i=1..n} \left(h_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| \right).$$

To find the optimal directions, we differentiate the Lagrangian \mathcal{L} associated with Problem (9). Let λ be the Lagrange multiplier associated with the constraint $\|Y\|^2 = 1$, which is equivalent to the initial constraint. This constraint replaces the initial one as it is differentiable. The optimality condition $\nabla \mathcal{L}(Y, \lambda) = 0$ is equivalent to the system:

$$\begin{cases} \vdots \\ 2y_i \left(h_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| - \lambda \right) = 0 & \text{for } i = 1, \dots, n. \\ \vdots \end{cases}$$

The n solution vectors $Y_i, i = 1, \dots, n$ are such that:

$$y_i = 1 \text{ and } y_j = 0 \text{ for } j \neq i \quad \text{and} \quad \lambda = h_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|.$$

Going back to the initial problem, the n solution vectors X_i are finally:

$$X_i = P^{-1}Y_i = \mathbf{u}_i.$$

The previous demonstration indicates that the optimal directions of the metric are aligned with the main directions of the Hessian. Replacing the value of X_i back in Relation (7), the local error model now becomes:

$$e_{\mathcal{M}}(a) = \sum_{i=1}^n h_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|. \quad (10)$$

In the remainder of paper, we consider that the optimal direction are those of H_u . However, we still have to compute the optimal sizes.

4 Calculus of Variation

From the previous section, we know that the interpolation error in the neighborhood of the vertex a could be modelled by Relation (10). Now, we are looking for a function \mathcal{M} that minimizes, for a given number N of vertices, the \mathbf{L}^p norm of this error. To this end, we have to solve the following problem:

$$\min_{\mathcal{M}} \mathcal{E}(\mathcal{M}) = \min_{\mathcal{M}} \int_{\Omega} (e_{\mathcal{M}}(\mathbf{x}))^p \, d\mathbf{x} = \min_{h_i} \int_{\Omega} \left(\sum_{i=1}^n h_i^2(\mathbf{x}) \left| \frac{\partial^2 u}{\partial \alpha_i^2}(\mathbf{x}) \right| \right)^p \, d\mathbf{x}, \quad (11)$$

under the constraint:

$$\mathcal{C}(\mathcal{M}) = \int_{\Omega} \prod_{i=1}^n h_i^{-1}(\mathbf{x}) \, d\mathbf{x} = \int_{\Omega} d(\mathbf{x}) \, d\mathbf{x} = N. \quad (12)$$

In the following, a formal resolution scheme is proposed. Then we check that the resulting solution is indeed a metric and is also the sole minimum.

4.1 Anisotropic Quotients in Dimension n

We notice that if the local sizes h_i are used as unknowns then Constraint (12) is non linear, however the constraint is linear with respect to the density $d = \prod_{i=1}^n h_i^{-1}$. Nevertheless, if d is introduced as an unknown, then one of the h_i will be removed, thus removing a certain symmetry. Moreover, anisotropic ratios r_i , which relate all the sizes i.e., $r_i = f(h_1, h_2, \dots, h_n)$, may be naturally extracted from the metric. Instead of keeping h_i as variables, we shall use what we call anisotropic quotients. Let us propose a definition of anisotropic quotients in dimension n .

The idea is to define a notion of an anisotropic quotient very similar to the notion of an anisotropic ratio in two dimensions and that can be easily extended to any dimension. Indeed, none clear definition of the anisotropic ratio is known by the authors in three dimensions. Then, this definition will be used to perform a substitution of variables to obtain as unknown the first $n - 1$ anisotropic quotients r_i and the density d .

We define the i^{th} anisotropic quotient in dimension n as:

$$r_i = \left(\frac{h_i^n}{\prod_{k=1}^n h_k} \right)^{1/n}.$$

The advantage of this definition is to provide a simple geometric interpretation in terms of the ratio of hyper-volumes. In two dimensions, this quotient is simply the square root of the anisotropic ratio.

From the previous definition, the following variable substitution is considered to solve Problem (11)-(12):

$$(h_i)_{i=1..n} \rightarrow ((r_i)_{i=1..n-1}, d).$$

Conversely, the local sizes are calculated by:

$$h_i = d^{-\frac{1}{n}} r_i \text{ for } i = 1, \dots, n-1 \text{ and } h_n = d^{-\frac{1}{n}} \left(\prod_{i=1}^{n-1} r_i \right). \quad (13)$$

The intermediate pivot variable P is defined as $P = \left(\prod_{i=1}^{n-1} r_i \right)^{-1}$. Finally, after the substitution (13) we face the following problem to solve:

$$\min_{((r_i)_i, d)} \int_{\mathbf{x} \in \Omega} d^{-\frac{2p}{n}}(\mathbf{x}) \left(\sum_{i=1}^{n-1} r_i^2(\mathbf{x}) \left| \frac{\partial^2 u}{\partial \alpha_i^2}(\mathbf{x}) \right| + P^{-2}(\mathbf{x}) \left| \frac{\partial^2 u}{\partial \alpha_n^2}(\mathbf{x}) \right| \right)^p \mathbf{d}\mathbf{x}, \quad (14)$$

under the linear constraint:

$$\int_{\mathbf{x} \in \Omega} d(\mathbf{x}) \mathbf{d}\mathbf{x} = N. \quad (15)$$

4.2 Formal Resolution

In this section, we assume that the considered functions are smooth enough. The Euler-Lagrange optimality necessary condition reads: for a critical point \mathcal{M} , the variation of the cost function \mathcal{E} is proportional to the variation of the constraint \mathcal{C} . Let $\delta\mathcal{E}(\mathcal{M}; \delta\mathcal{M})$ be the variation of the functional \mathcal{E} in \mathcal{M} in the direction $\delta\mathcal{M}$ defined as:

$$\delta\mathcal{E}(\mathcal{M}; \delta\mathcal{M}) = \lim_{\varepsilon \rightarrow 0} \frac{\mathcal{E}(\mathcal{M} + \varepsilon \delta\mathcal{M}) - \mathcal{E}(\mathcal{M})}{\varepsilon}.$$

As the number of vertices is constant, the variation of the constraint is zero:

$$\delta\mathcal{C}(\mathcal{M}; \delta\mathcal{M}) = \lim_{\varepsilon \rightarrow 0} \frac{1}{\varepsilon} \left(\int (d + \varepsilon d) - \int d \right) = \int \delta d = 0.$$

Then, we deduce by means of the Euler-Lagrange optimality necessary condition that it exists a Lagrange multiplier λ such that:

$$\forall \delta\mathcal{M}, \quad \delta\mathcal{E}(\mathcal{M}; \delta\mathcal{M}) = \lambda \delta\mathcal{C}(\mathcal{M}; \delta\mathcal{M}) = 0,$$

or equivalently:

$$\forall \delta r_i \text{ for } i = 1, \dots, n-1 \text{ and } \forall \delta d \text{ such that } \int \delta d = 0 \text{ then} \\ \sum_{i=1}^{n-1} \delta\mathcal{E}(\mathcal{M}; \delta r_i) + \delta\mathcal{E}(\mathcal{M}; \delta d) = 0, \quad (16)$$

where δr_i and δd are functions representing the different components of the variation $\delta \mathcal{M}$, i.e., $\delta \mathcal{M} = (\delta r_1, \delta r_2, \dots, \delta r_{n-1}, \delta d)$.

Problem (14)-(15) is solved in two distinct steps that are based on Equation (16):

- step 1: evaluation of the anisotropic quotients r_i for $i = 1, \dots, n - 1$
- step 2: evaluation of the density d , which is a normalization, to obtain the number of desired vertices N .

Step 1. If Equality (16) is developed by choosing $\delta d = 0$, it comes:

$$\int_{\Omega} 2^p d^{-\frac{2p}{n}} (*)^{p-1} \left(\sum_{i=1}^{n-1} \left(r_i \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| - P^{-2} r_i^{-1} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| \right) \delta r_i \right) = 0, \quad (17)$$

where the term $(*)$ is equal to:

$$\sum_{i=1}^{n-1} r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| + P^{-2} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|.$$

If the integrand of Equation (17) is zero then the equation is trivially satisfied and its solution is retained. It implies the $n - 1$ following relations:

$$r_i \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| - P^{-2} r_i^{-1} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| = 0 \text{ for } i = 1, \dots, n - 1.$$

We deduce the $n - 1$ first anisotropic quotients:

$$r_i = \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{\frac{1}{2}} \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2}} P^{-1} \text{ for } i = 1, \dots, n - 1. \quad (18)$$

Using the $n - 1$ anisotropic quotients given by (18) and the fact that P is defined by the product of the inverse of the r_i for $i = 1, \dots, n - 1$, we have:

$$P = P^{-(n-1)} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{\frac{n-1}{2}} \prod_{i=1}^{n-1} \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2}},$$

then,

$$P = \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{\frac{1}{2}} \prod_{i=1}^n \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2n}}. \quad (19)$$

From the previous relation, r_i for $i = 1, \dots, n - 1$ are exhibited independently of P :

$$r_i = \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{\frac{1}{2}} \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2}} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{-\frac{1}{2}} \prod_{i=1}^n \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right|^{\frac{1}{2n}},$$

and finally,

$$r_i = \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2}} \prod_{i=1}^n \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{1}{2n}}.$$

The anisotropic quotients may be also expressed in function of matrix $|H_u|$:

$$r_i = \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-\frac{1}{2}} (\det |H_u|)^{\frac{1}{2n}} \text{ for } i = 1, \dots, n - 1.$$

Step 2. The evaluation of the density d is also deduced from Equality (17) by legally choosing $\delta r_i = 0$ for $i = 1, \dots, n - 1$, it results:

$$\int_{\Omega} d^{-\frac{2p+n}{n}} \left(\sum_{i=1}^{n-1} r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| + P^{-2} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| \right)^p \delta d = 0. \tag{20}$$

A solution of the previous equality with δd verifying the constraint $\int \delta d = 0$ is obtained when the integrand that is multiplied by δd is constant. Therefore, we have:

$$d^{-\frac{2p+n}{n}} \left(\sum_{i=1}^{n-1} r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| + P^{-2} \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| \right)^p = Cst,$$

which, from Relations (18) and (19) equivalent to:

$$n^p d^{-\frac{2p+n}{n}} \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{p}{n}} = Cst.$$

Finally, as the mesh must contain N vertices which are given by the integral of the density on Ω , the density reads:

$$d = N \left(\int_{\Omega} \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{p}{2p+n}} \right)^{-1} \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{p}{2p+n}}.$$

The density may also be written in function of the determinant of matrix $|H_u|$:

$$d = N \left(\int_{\Omega} (\det |H_u|)^{\frac{p}{2p+n}} \right)^{-1} (\det |H_u|)^{\frac{p}{2p+n}}.$$

Final solution. The solution of Problem (14)-(15) has been exhibited above. Then, the converse variables substitution given by Relation (13) is applied to solve Problem (11)-(12). For $i = 1, \dots, n$, we have:

$$h_i = \frac{1}{N^{1/n}} \left(\int_{\Omega} (\det |H_u|)^{\frac{p}{2p+n}} \right)^{1/n} (\det |H_u|)^{\frac{1}{2(2p+n)}} \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{-1/2},$$

or equivalently,

$$\lambda_i = \frac{1}{h_i^2} = N^{2/n} \left(\int_{\Omega} (\det |H_u|)^{\frac{p}{2p+n}} \right)^{-2/n} (\det |H_u|)^{\frac{-1}{2p+n}} \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|.$$

4.3 Uniqueness and Order of Convergence

In this section, we will demonstrate by a direct proof that the obtained metric really minimize the error model. Let \mathcal{M} be a metric define by $n - 1$ anisotropic quotients r_i for $i = 1, \dots, n - 1$ and a density d which is written under the general form $d = N(\int_{\Omega} f)^{-1} f$. We recall that r_i for $i = 1, \dots, n - 1$ and f are strictly positive functions. From Relation (14), the error committed with this metric is given by:

$$\mathcal{E}(\mathcal{M}) = N^{-\frac{2p}{n}} \left(\int_{\Omega} f \right)^{-\frac{2p}{n}} \int_{\Omega} f^{-\frac{2p}{n}} \left(\sum_{i=1}^{n-1} r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| + \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| \prod_{i=1}^{n-1} r_i^{-2} \right)^p.$$

The error committed with the optimal metric \mathcal{M}_{opt} is obtained with Relation (11):

$$\mathcal{E}(\mathcal{M}_{opt}) = n^p N^{-\frac{2p}{n}} \left(\int_{\Omega} \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{p}{2p+n}} \right)^{\frac{2p+n}{n}}. \tag{21}$$

In order to prove that $\mathcal{E}(\mathcal{M}_{opt}) \leq \mathcal{E}(\mathcal{M})$, we use the generalized arithmetic-geometric inequality which comes from the concavity of \ln :

$$\ln \left(\frac{1}{n} \sum_{i=1}^n r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| \right) \geq \frac{1}{n} \sum_{i=1}^n \ln \left(r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| \right) = \ln \left(\prod_{i=1}^n \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{1}{n}} \right),$$

as we have $\prod_{i=1}^n r_i = 1$. By substituting the value of r_n provided by Relation (13), it comes:

$$\sum_{i=1}^{n-1} r_i^2 \left| \frac{\partial^2 u}{\partial \alpha_i^2} \right| + \left| \frac{\partial^2 u}{\partial \alpha_n^2} \right| \prod_{i=1}^{n-1} r_i^{-2} \geq n \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{1}{n}}.$$

Finally, if we denote

$$g = \left| \prod_{i=1}^n \frac{\partial^2 u}{\partial \alpha_i^2} \right|^{\frac{p}{2p+n}},$$

we get

$$\begin{cases} \mathcal{E}(\mathcal{M}_{opt})^{\frac{n}{2p+n}} = n^{\frac{pn}{2p+n}} N^{-\frac{2p}{2p+n}} \int_{\Omega} g, \\ \mathcal{E}(\mathcal{M})^{\frac{n}{2p+n}} \geq n^{\frac{pn}{2p+n}} N^{-\frac{2p}{2p+n}} \left(\int_{\Omega} f \right)^{\frac{2p}{2p+n}} \left(\int_{\Omega} f^{-\frac{2p}{n}} g^{\frac{2p+n}{n}} \right)^{\frac{n}{2p+n}}. \end{cases}$$

By utilizing the Holder inequality, we obtain

$$\left(\int_{\Omega} f^{\frac{2p}{2p+n}} \left(\frac{g}{f^{\frac{2p}{2p+n}}} \right) \right) \leq \left(\int_{\Omega} f \right)^{\frac{2p}{2p+n}} \left(\int_{\Omega} f^{-\frac{2p}{n}} g^{\frac{2p+n}{n}} \right)^{\frac{n}{2p+n}}, \tag{22}$$

as we have

$$\begin{cases} 1 + \frac{n}{2p} \geq 1, \\ 1 + \frac{2p}{n} \geq 1, \\ \frac{1}{1 + \frac{n}{2p}} + \frac{1}{1 + \frac{2p}{n}} = 1. \end{cases}$$

Therefore, Relation (22) implies that $\mathcal{E}(\mathcal{M}_{opt}) \leq \mathcal{E}(\mathcal{M})$, for all metric \mathcal{M} . As optimization Problem (14)-(15) is strictly convex, the solution is unique.

Order of Convergence

Let first introduce the definition of order of convergence in dimension n .

Definition. A sequence of n -d metrics $(\mathcal{M}_N)_N$ such that $\mathcal{C}(\mathcal{M}_N) = N$ vertices (cf. Relation (12)) gives a k^{th} order of convergence for a given norm $\|e_{\mathcal{M}_N}(\mathbf{x})\|_{\mathbf{L}^p}$ if we have:

$$\|e_{\mathcal{M}_N}(\mathbf{x})\|_{\mathbf{L}^p} \leq Cst N^{-k/n}. \tag{23}$$

The p^{th} square root of Relation (21) is considered to estimate the order of convergence:

$$\mathcal{E}^{1/p}(\mathcal{M}_{opt}) = nN^{-\frac{2}{n}} \left(\int_{\Omega} |\det(H_u)|^{\frac{p}{2p+n}} \right)^{\frac{2p+n}{pn}} \leq \frac{Cst}{N^{2/n}}.$$

Therefore, according to Relation (23), the previous inequality expresses a second-order of convergence of the metric sequence obtained by the present adaptation strategy for regular solutions whatever the chosen dimension and the chosen \mathbf{L}^p norm.

5 Practical Continuous Metric in Three Dimensions

The metrics corresponding to the usual cases of the \mathbf{L}^1 , \mathbf{L}^2 and \mathbf{L}^∞ norms in three dimensions are presented. The \mathbf{L}^∞ is found by passing to the limit. Notice that the obtained expression is exactly the same as the one based on geometric error estimate that has been commonly used in the literature [13].

The results are sum up with the following equation:

$$\mathcal{M}_{\mathbf{L}^p} = D_{\mathbf{L}^p} \mathcal{R}_u^{-1} \begin{pmatrix} \|\lambda_1\| \\ \|\lambda_2\| \\ \|\lambda_3\| \end{pmatrix} \mathcal{R}_u,$$

where $D_{\mathbf{L}^p}$ and λ_i for $i = 1, 2, 3$ are given in the following table:

Norm	$D_{\mathbf{L}^p}$	$\ \lambda_i\ $	Convergence order
\mathbf{L}^p	$N^{\frac{2}{3}} \left(\int_{\Omega} \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{\frac{p}{2p+3}} \right)^{-\frac{2}{3}}$	$\left \frac{\partial^2 u}{\partial \alpha_i^2} \right \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{-\frac{1}{2p+3}}$	2
\mathbf{L}^1	$N^{\frac{2}{3}} \left(\int_{\Omega} \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{\frac{1}{5}} \right)^{-\frac{2}{3}}$	$\left \frac{\partial^2 u}{\partial \alpha_i^2} \right \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{-\frac{1}{5}}$	2
\mathbf{L}^2	$N^{\frac{2}{3}} \left(\int_{\Omega} \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{\frac{2}{7}} \right)^{-\frac{2}{3}}$	$\left \frac{\partial^2 u}{\partial \alpha_i^2} \right \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{-\frac{1}{7}}$	2
\mathbf{L}^∞	$N^{\frac{2}{3}} \left(\int_{\Omega} \left \prod_{i=1}^3 \frac{\partial^2 u}{\partial \alpha_i^2} \right ^{\frac{1}{2}} \right)^{-\frac{2}{3}}$	$\left \frac{\partial^2 u}{\partial \alpha_i^2} \right $	2

Notice that the metric could also be written:

$$\mathcal{M}_{\mathbf{L}^p} = D_{\mathbf{L}^p} (\det |H_u|)^{\frac{-1}{2p+3}} \mathcal{R}_u^{-1} |A| \mathcal{R}_u. \quad (24)$$

Relation (24) gives a quantitative physical interpretation of the impact of the norm on the metric construction. Indeed, the metric prescribes a fine size for a large eigenvalue of the Hessian. For the continuous metric, the eigenvalues are multiplied by the term $(\det |H_u|)^{\frac{-1}{2p+3}}$ which is small for large eigenvalues. Consequently, reducing p of the \mathbf{L}^p norm implies to less refine large Hessian regions, for instance if a discontinuity occurs in the solution, it becomes less “attractive” for lower p . This will be illustrated in the next section on numerical examples.

6 Analytical Examples

In this section, we propose to analyze the impact of the continuous metric throughout a mesh adaptation process on two-dimensional and three-dimensional analytical examples. The mesh adaptation process is a single loop algorithm. We first apply the analytical function on the mesh, then the Hessian of the solution is computed by means of a double \mathbf{L}^2 projection algorithm. Finally, the continuous metric is constructed and a unit mesh with respect to this metric is generated. The process is repeated until convergence of the mesh. Notice that here the gradation of the mesh (the size variation between two neighboring elements) is not controlled.

As regards the mesh adaptation algorithm, the two-dimensional and three-dimensional methods are based on local mesh modifications. Each approach consists in modifying iteratively the initial mesh so as to complete a unit mesh

with respect to the metric \mathcal{M} . The ingredients to comply with these requirements typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modification operators involved are: edge flipping, edge collapsing, edge splitting, node repositioning and degree relaxation.

The surface mesh modification algorithm, also used for the two-dimensional case, is pretty straightforward, edge lengths are computed with respect to the metric \mathcal{M} and edges too small are collapsed while edges too long are splitted into unit length segments. Edge flips and node repositioning operations are performed to improve the overall size and shape mesh quality [11].

Similarly, in the volume mesh modification algorithm edge lengths are computed with respect to the metric \mathcal{M} and edges too small are collapsed while edges too long are splitted using a vertex insertion procedure based on an anisotropic generalization of the Delaunay kernel [8]. Generalized edge swaps and node repositioning operations are performed to improve the overall size and shape mesh quality [15].

For each function, we compare meshes obtained for minimizing \mathbf{L}^1 , \mathbf{L}^2 and \mathbf{L}^∞ norms of the model error and we analyze the order of convergence obtained with each norm.

A Smooth Two-Dimensional Function

The first function f_1 is a smooth function involving small and large variation amplitudes, Figure 2 (top right). The function is defined as follow:

$$f_1(x, y) = \begin{cases} 0.1 \sin(50xy) & \text{if } xy \leq \frac{-\pi}{50} \\ \sin(50xy) & \text{if } \frac{-\pi}{50} < xy \leq \frac{2\pi}{50} \\ 0.1 \sin(50xy) & \text{if } \frac{2\pi}{50} < xy \end{cases} \quad (25)$$

As we notice in Section 5, the metric defined with \mathbf{L}^1 norm better captures the small amplitudes than metrics constructed with norms with larger p . This is exemplified in Figure 2 where meshes are obtained for \mathbf{L}^1 , \mathbf{L}^2 and \mathbf{L}^∞ norms with a number of vertices targeted to 3,300 are represented. Indeed, the small amplitude waves regions (in each corner) are more captured when using a \mathbf{L}^p norm with a lower p whereas the large amplitude region is clearly more refined with the \mathbf{L}^∞ norm.

In each case, the adapted mesh fits well the metric. The mesh adaptation algorithm statistics indicate that almost 91 % of the edges have a unit length, i.e., a length between $1/\sqrt{2}$ and $\sqrt{2}$. We can compute the efficiency index of the resulting adapted meshes, i.e., a scalar value representing the adequacy between the metric specification and the actual element size, with the following formula:

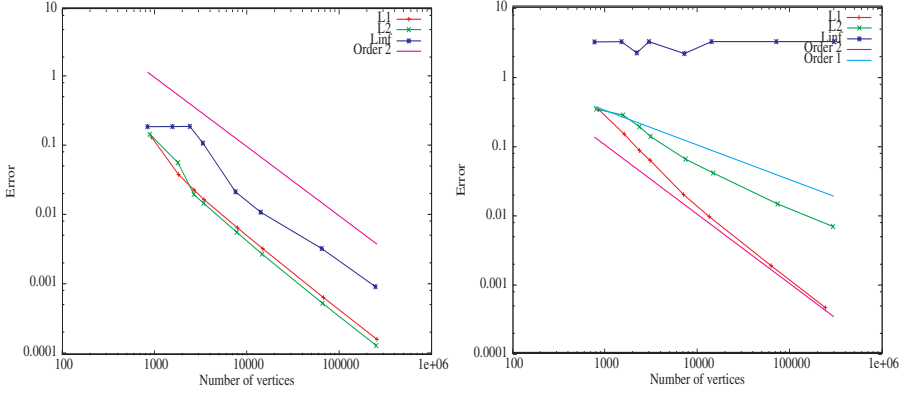


Fig. 1. Convergence order for functions f_1 (left) and f_2 (right) regarding the used \mathbf{L}^p norm

$$\tau_{\mathcal{H}} = \exp \left(\frac{1}{n_e} \sum_{i=1}^{n_e} (Q_l(e_i) - 1) \right),$$

where n_e is the number of edges of the mesh and $Q_l(e_i)$ is the length quality of the edge e_i in the metric given by:

$$Q_l(e_i) = \begin{cases} l_{\mathcal{M}}(e_i) & \text{if } l_{\mathcal{M}}(e_i) \leq 1 \\ (l_{\mathcal{M}}(e_i))^{-1} & \text{else} \end{cases}$$

with $l_{\mathcal{M}}(e_i)$ the edge length in the metric \mathcal{M} . Here, an efficiency index close to 0.86 is obtained in each case.

Figure 3 shows the final mesh obtained for minimizing \mathbf{L}^1 norm of the error with a number of vertices targeted to 15,000. The anisotropic mesh refinement along each wave is clearly emphasized in this figure.

For this function, a convergence of order 2 is reached for all norms as predicted by theory. This convergence is illustrated in Figure 1 (left).

A Steep-Gradient Two-Dimensional Function

The second analytic function f_2 is a function with small amplitude waves going through a sinusoidal steep-gradient step (similar to a Dirac layer) in the middle of the domain, Figure 4 (top right). The function is defined as follow:

$$f_2(x, y) = 0.1 \sin(50x) + \arctan \left(\frac{0.1}{\sin(5y) - 2x} \right) \quad (26)$$

This function is proposed to point out the attractive effect of a steep-gradient region on the mesh adaptation process. As illustrated in Figure 4,

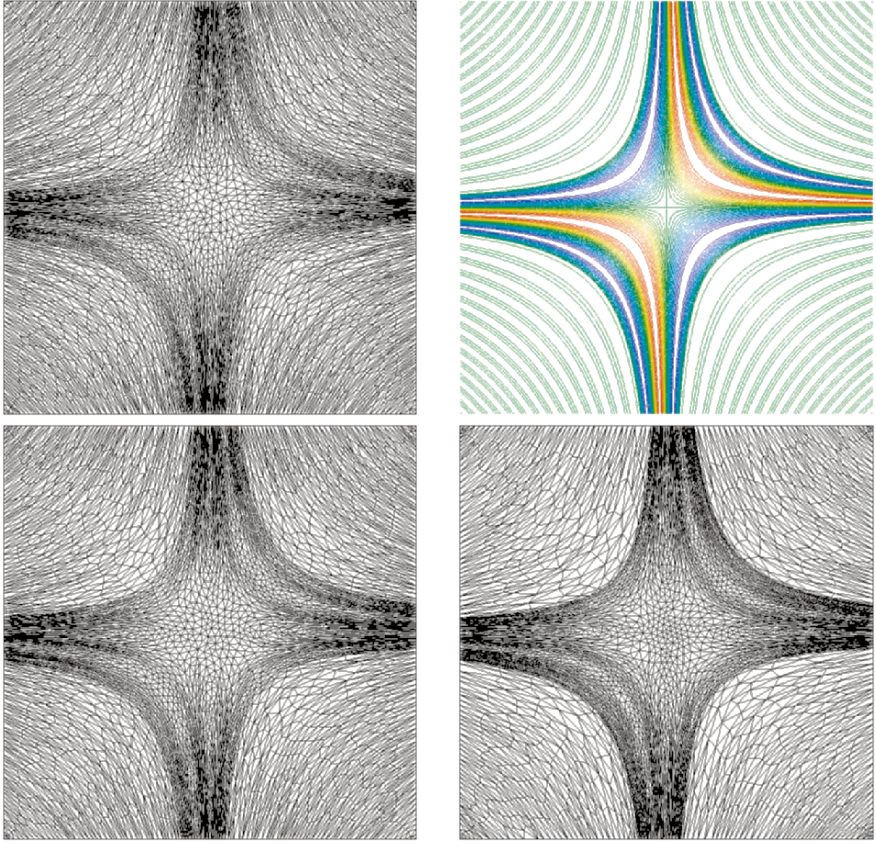


Fig. 2. Final adapted meshes minimizing \mathbf{L}^1 (top left), \mathbf{L}^2 (bottom left) and \mathbf{L}^∞ (bottom right) norms with a target number of vertices equal to 3,300 for the analytical function f_1 . Top right, f_1 iso-values are represented

the mesh obtained ($N = 15,000$) with \mathbf{L}^∞ norm only refines the steep-gradient region, whereas meshes with \mathbf{L}^1 and \mathbf{L}^2 norms were able to capture the small amplitude waves. Notice that even with 300,000 vertices \mathbf{L}^∞ norm continues to impose only vertex insertion in the steep-gradient region. This example puts in evidence that metrics defined by a \mathbf{L}^p norm with a low p value are more appropriate to capture weak phenomena in simulations involving large amplitude phenomena such as shocks in CFD.

For this harder case, it is more difficult for the adaptive mesh generator to respect the metric field, even more without any mesh gradation, as most of the vertices are inserted in the steep-gradient region. The adapted mesh for the \mathbf{L}^1 norm fits well the metric, about 93 % of the edges having a length between $1/\sqrt{2}$ and $\sqrt{2}$ and the efficiency index being 0.866. Nevertheless, only 51 % of the edges have a length between $1/\sqrt{2}$ and $\sqrt{2}$ for the \mathbf{L}^∞ case which

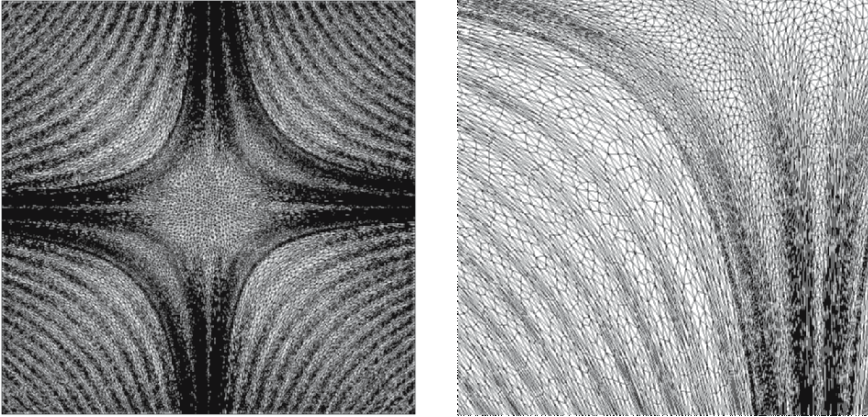


Fig. 3. Final adapted mesh minimizing \mathbf{L}^1 norm with a target number of vertices equal to 15,000 for the analytical function f_1

leads to an efficiency index equal to 0.73 (notice that 80 % of the edges have a length between 1/2 and 2, mainly because of the lack of gradation control).

An order of convergence equal to 2 is only reached for \mathbf{L}^1 norm, whereas orders of 1 and 0 are obtained for \mathbf{L}^2 and \mathbf{L}^∞ norms, respectively. Here, we don't want to drive any conclusion, as a complete analysis of the discontinuous cases must be assessed as in [6] where authors explained that theoretical orders of convergence are in some cases only asymptotically reached.

A Smooth Three-Dimensional Function

Function f_{3d} is a smooth function involving small and large variation amplitudes. f_{3d} is a three-dimensional extension of function f_1 in a spherical domain. The function is defined as follows:

$$f_{3d}(x, y, z) = \begin{cases} 0.1 \sin(50 \mathbf{x}) & \text{if } \mathbf{x} \leq \frac{-\pi}{50} \\ \sin(50 \mathbf{x}) & \text{if } \frac{-\pi}{50} < \mathbf{x} \leq \frac{2\pi}{50} \\ 0.1 \sin(50 \mathbf{x}) & \text{if } \frac{2\pi}{50} < \mathbf{x} \end{cases} \quad (27)$$

where $\mathbf{x} = (x - 0.4)(y - 0.4)(z - 0.4)$.

As mentioned previously in two dimensions, the metric defined with \mathbf{L}^1 norm better captures the small amplitudes than metrics constructed with norms with larger p . This is shown in Figures. 5 and 6 where anisotropic adapted surface and volume meshes are depicted for \mathbf{L}^1 , \mathbf{L}^2 and \mathbf{L}^∞ norms with a number of vertices targeted to 275,000. The small variations of the function are clearly better captured with the \mathbf{L}^1 norm than the \mathbf{L}^∞ norm.

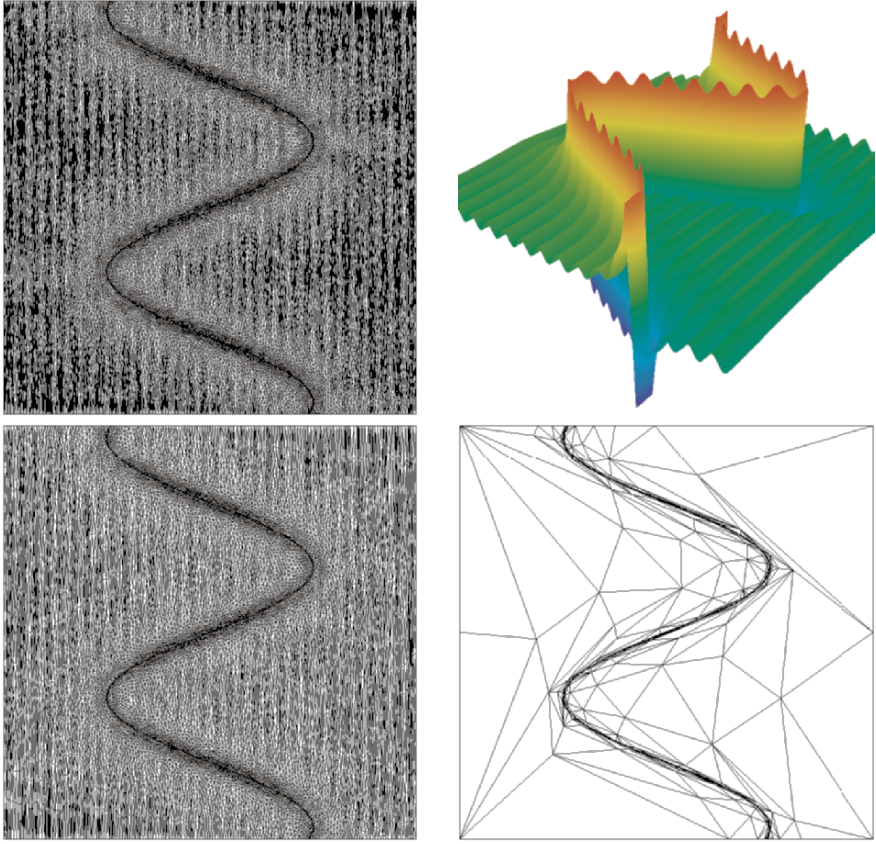


Fig. 4. Final adapted meshes minimizing \mathbf{L}^1 (top left), \mathbf{L}^2 (bottom left) and \mathbf{L}^∞ (bottom right) norms with a target number of vertices equal to 15,000 for the analytical function f_2 . Top right, the surface described by f_2 is represented

In each case, the adapted tetrahedral mesh respects well the metric. Between 80 % and 85% of the edges have a unit length and an efficiency index between 0.825 and 0.84 is obtained.

Order of convergence for this analytical function is presented in Figure 5 (top right). A convergence of order equal to 2 is reached for \mathbf{L}^1 and \mathbf{L}^2 norms as expected by theory. As for as \mathbf{L}^∞ norm is concerned, second order convergence is not obtained regarding the global \mathbf{L}^∞ norm over all the domain. Nevertheless, if the local \mathbf{L}^∞ norm is integrated over the domain then an almost second order of convergence is observed. Indeed, this norm aims at equally distributing the interpolation error over the domain.

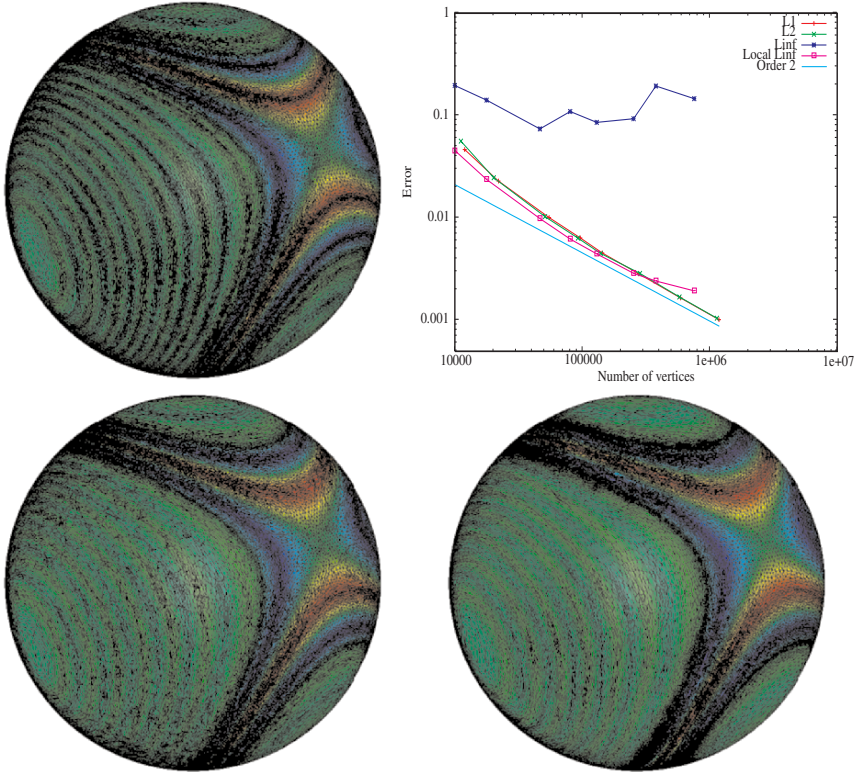


Fig. 5. Final anisotropic adapted surface meshes minimizing \mathbf{L}^1 (top left), \mathbf{L}^2 (bottom left) and \mathbf{L}^∞ (bottom right) norms with a target number of vertices equal to 275,000 for the analytical function f_{3d} . Top right, convergence order for function f_{3d} regarding the used \mathbf{L}^p norm

7 Conclusion

In this paper, a continuous approach has been proposed to derive metrics in order to minimize the interpolation error in \mathbf{L}^p norm. This approach is based on classes of equivalence between meshes represented by continuous metrics. We have demonstrated that there exists a unique optimal metric. A theoretical analysis gives the conditions for second-order convergence.

These theoretical results have been exemplified on analytical functions in two and three dimensions where second order of convergence have been observed.

In future work, we intend to apply this continuous setting to discontinuous solutions and to realistic three-dimensional numerical simulations.

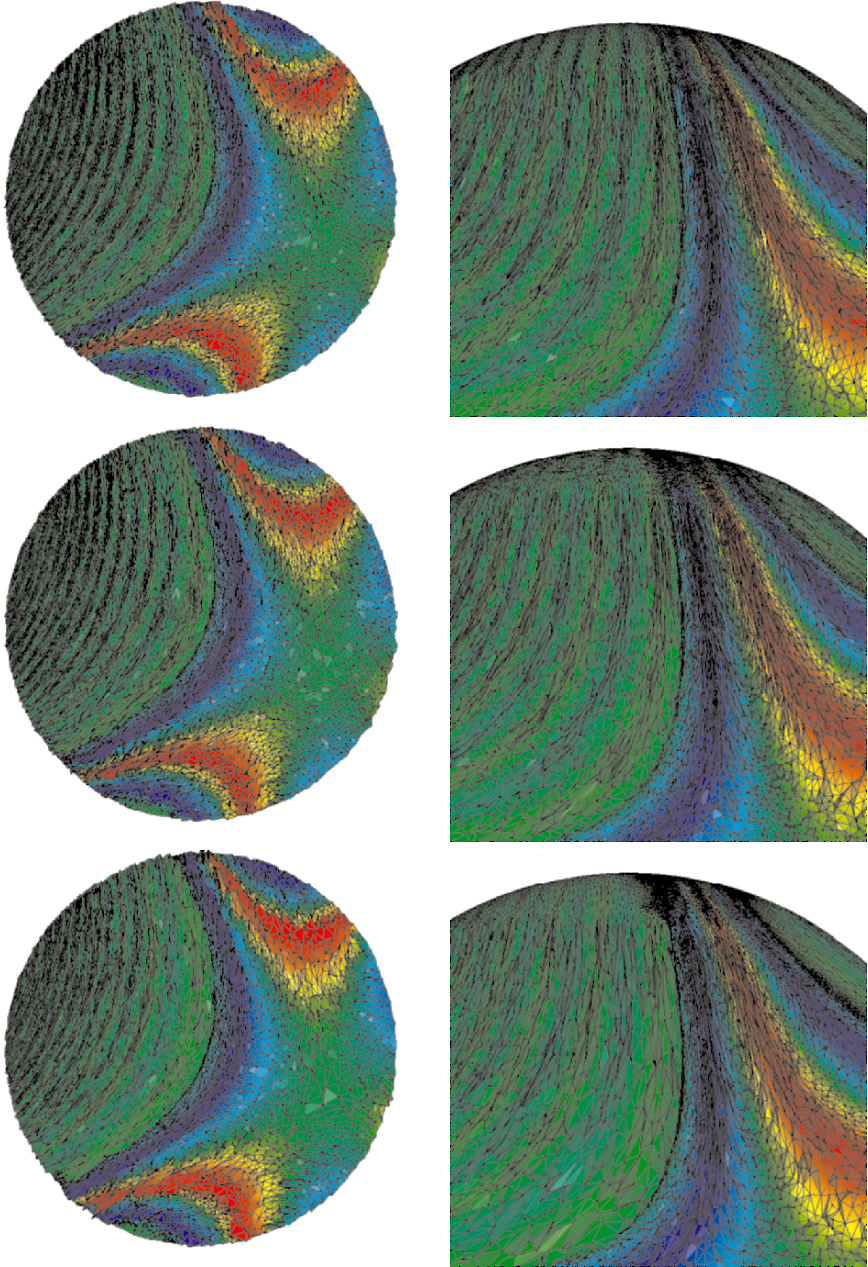


Fig. 6. Final anisotropic adapted volume meshes in a cut plane minimizing L^1 (top), L^2 (middle) and L^∞ (bottom) norms with a target number of vertices equal to 275,000 for the analytical function f_{3d}

References

1. ALMEIDA, R., FEIJÓO, R., GALEAO, A., PADRA, C., AND SILVA, R. Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Comput. Methods Appl. Mesh. Engrg.* 182 (2000), 379–400.
2. BISWAS, R., AND STRAWN, R. A new procedure for dynamic adaption of three-dimensional unstructured grids. *Applied Numerical Mathematics* 13 (1994), 437–452.
3. BUSCAGLIA, G., AND DARI, E. Anisotropic mesh optimization and its application in adaptivity. *Int. J. Numer. Meth. Engrg* 40 (1997), 4119–4136.
4. CASTRO-DIAZ, M., HECHT, F., MOHAMMADI, B., AND PIRONNEAU, O. Anisotropic unstructured mesh adaptation for flow simulations. *Int. J. Numer. Meth. Fluids* 25 (1997), 475–491.
5. CHEN, L., SUN, P., AND XU, J. Optimal anisotropic simplicial meshes for minimizing interpolation errors in L^p -norms. *Math. Comp.* 0 (2006). to appear.
6. COURTY, F., LESERVOISIER, D., GEORGE, P.-L., AND DERVIEUX, A. Continuous metrics and mesh adaptation. *Applied Numerical Mathematics* 56 (2006), 117–145.
7. DERVIEUX, A., LESERVOISIER, D., GEORGE, P.-L., AND COUDIERE, Y. About theoretical and practical impact of mesh adaptations on approximation of functions and of solution of pde. *Int. J. Numer. Meth. Fluids* 43 (2003), 507–516.
8. DOBRZYNSKI, C. *Adaptation de maillage anisotrope 3D et application à l'aérothermique des bâtiments*. PhD thesis, Université Pierre et Marie Curie, Paris VI, Paris, France, 2005. (in French).
9. DOMPIERRE, J., VALLET, M., FORTIN, M., BOURGAULT, Y., AND HABASHI, W. Anisotropic mesh adaptation: towards a solver and user independent cfd. *AIAA paper 97-0861* (1997).
10. FORMAGGIA, L., MICHELETTI, S., AND PEROTTO, S. Anisotropic mesh adaptation in computational fluid dynamics: Application to the advection-diffusion-reaction and the Stokes problem. *Applied Numerical Mathematics* 51 (2004), 511–533.
11. FREY, P. About surface remeshing. In *Proc. of 9th Int. Meshing Roundtable* (New Orleans, LO, USA, 2000), pp. 123–136.
12. FREY, P., AND ALAUZET, F. Anisotropic mesh adaptation for transient flows simulations. In *Proc. of 12th Int. Meshing Roundtable* (Santa Fe, New Mexico, USA, 2003), pp. 335–348.
13. FREY, P., AND ALAUZET, F. Anisotropic mesh adaptation for CFD computations. *Comput. Methods Appl. Mech. Engrg.* 194, 48–49 (2005), 5068–5082.
14. FREY, P., AND GEORGE, P.-L. *Mesh generation. Application to finite elements*. Hermès Science, Paris, Oxford, 2000.
15. GEORGE, P.-L., AND BOROUCAKI, H. Back to edge flips in 3 dimensions. In *Proc. of 12th Int. Meshing Roundtable* (Santa Fe, NM, USA, 2003), pp. 393–402.
16. GRUAU, C., AND COUPEZ, T. 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. *Comput. Methods Appl. Mech. Engrg.* 194, 48–49 (2005), 4951–4976.
17. HUANG, W. Metric tensors for anisotropic mesh generation. *J. Comput. Phys.* 204 (2005), 633–665.
18. KALLINDERIS, Y., AND VIJAYAN, P. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA Journal* 31, 8 (1993), 1440–1447.

19. LESERVOISIER, D., GEORGE, P.-L., AND DERVIEUX, A. Métrique continue et optimisation de maillage. RR-4172, INRIA, Apr. 2001. (in French).
20. LI, X., SHEPHARD, M., AND BEALL, M. 3D anisotropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.* 194, 48-49 (2005), 4915–4950.
21. LÖHNER, R., AND BAUM, J. Adaptive h-refinement on 3D unstructured grids for transient problems. *Int. J. Numer. Meth. Fluids* 14 (1992), 1407–1419.
22. MAVRIPLIS, D. Adaptive meshing techniques for viscous flow calculations on mixed-element unstructured meshes. *AIAA paper 97-0857* (1997).
23. PAIN, C., HUMBLEBY, A., DE OLIVEIRA, C., AND GODDARD, A. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Comput. Methods Appl. Mech. Engrg.* 190 (2001), 3771–3796.
24. PERAIRE, J., PEIRO, J., AND MORGAN, K. Adaptive remeshing for three-dimensional compressible flow computations. *J. Comput. Phys.* 103 (1992), 269–285.
25. RAUSCH, R., BATINA, J., AND YANG, H. Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations. *AIAA Journal* 30 (1992), 1243–1251.
26. SCHALL, E., LESERVOISIER, D., DERVIEUX, A., AND KOOBUS, B. Mesh adaptation as a tool for certified computational aerodynamics. *Int. J. Numer. Meth. Fluids* 45 (2004), 179–196.
27. SPEARES, W., AND BERZINS, M. A 3d unstructured mesh adaptation algorithm for time-dependent shock-dominated problems. *Int. J. Numer. Meth. Fluids* 25 (1997), 81–104.
28. TAM, A., AIT-ALI-YAHIA, D., ROBICHAUD, M., MOORE, M., KOZEL, V., AND HABASHI, W. Anisotropic mesh adaptation for 3d flows on structured and unstructured grids. *Comput. Methods Appl. Mech. Engrg.* 189 (2000), 1205–1230.

How Efficient are Delaunay Refined Meshes? An Empirical Study

Bruce Simpson

School of Computer Science; University of Waterloo, Waterloo, Ontario, Canada,
N2L 3G1
rbsimpson@uwaterloo.ca

Summary. Given a data function, $f(x, y)$, defined for (x, y) in a domain, D and an error measure for approximating f on D , we can call a piecewise linear function, $f^{pl}(x, y)$, acceptable if its error measure is less than or equal to a given error tolerance. Adaptive Delaunay Refinement (ADR) is one approach to generating a mesh for D that can be used to create an acceptable $f^{pl}(x, y)$. A measure of the efficiency of methods for generating a mesh, M , for piecewise approximation is the size of M . In this paper, we present empirical evidence that ADR generated meshes can be twice as large as necessary for producing acceptable interpolants for harmonic functions. The error measure used in this study is the maximum of the triangle average L2 errors in M . This observation is based on demonstrating a comparison mesh generating using maximal efficiency mesh theory as reviewed in the paper. There are two different approaches to point placement commonly used in ADR, edge based refinement and circumcenter based refinement. Our study indicates that there is no significant difference in the efficiency of the meshes generated by these two approaches.

1 Introduction

The meshing context of this paper is piecewise linear function approximation on a planar domain D . I.e. Given a function $f(x, y)$ defined for $(x, y) \in D$, create an unstructured triangular mesh, M , for D and the coefficients of a piecewise linear approximation, $f^{(pl)}(x, y) \approx f(x, y)$. Unstructured meshes pose an efficiency-computational cost trade-off. Local computations with unstructured meshes tend to be more complex than with structured grids. However, a given accuracy in $f^{(pl)}(x, y)$ can usually be achieved by an unstructured mesh with significantly fewer vertices than needed by structured grids. So, for global computations, unstructured meshes can be more efficient by virtue of being smaller. Iterative adaptive h -refinement is a long standing mesh generation

techniques that aims to provide this efficiency (since 1970s.) The combination of this technique with Delaunay meshing has been used almost as long (since 1980s). We will use the abbreviation ADR for adaptive Delaunay h -refinement in the sequel. Clearly, there is a limit, for a given f and target error, on how small the meshes that meet this target error can be. A mesh that meets the error target with a minimal number of vertices is a maximal efficiency mesh, Simpson [26]. In this paper, we address questions of how efficient are the meshes generated by adaptive Delaunay refinement, relative to maximal efficiency meshes.

In §2, we review the development of adaptive Delaunay refinement methods and explain the details of the versions that are used in our computations. In particular, we distinguish two choices for the insertion vertex used for refining a triangle, T , the midpoint of a longest edge of T , which we will denote $LEBis(T)$, and the circumcenter of T , which we will denote by $CC(T)$. In §3, we present in detail a demonstration of ADR for piecewise linear interpolation of a specific harmonic function. This demonstration leads to a discussion of a class of simple meshes consisting of isosceles right angled triangles. This class is closed under adaptive h -refinement or ADR for either $LEBis(T)$ or $CC(T)$ type insertion and any of these methods produce the same result when applied to an initial mesh in the class.

The demonstration of §3 includes evidence that ADR generated meshes can be roughly twice as large as maximum efficiency meshes, for isotropic data. This evidence is based on a comparison to the size of a highly efficient mesh created by a computation specific to the data function. The computation of the comparison mesh is detailed in §4 and an overview of the theory supporting this computation is given in §5. In §6, a second example is presented in less detail that confirms the features of the example discussed in §3. The results of this study are highly consistent with a similar study by E. F. D’Azevedo, [10], which demonstrates that adaptive meshes created by the program PLTMG [3], discussed below, are about twice as large as specially constructed comparison meshes.

2 Iterative Refinement Methods

Iterative refinement refers to a hierarchy of mesh generation methods, as shown in Figure 1. We will use this hierarchy to discuss related previous work and the methods used in the computations of this paper. Basic iterative h -refinement (BR) requires a size measure for triangles, $size(T)$, an input a mesh M_0 for D , and a maximum size tolerance, $sizeTol$, which may be variable across D . The method then attempts to create a mesh, M for D such that $size(T) \leq sizeTol$ for every $T \in M$. Delaunay refinement methods, (DR), are basic h -refinement methods in which M_0 and M are constrained Delaunay triangulations. Adaptive refinement methods, (AR) are basic h -refinement methods related to specific applications, such as piecewise linear

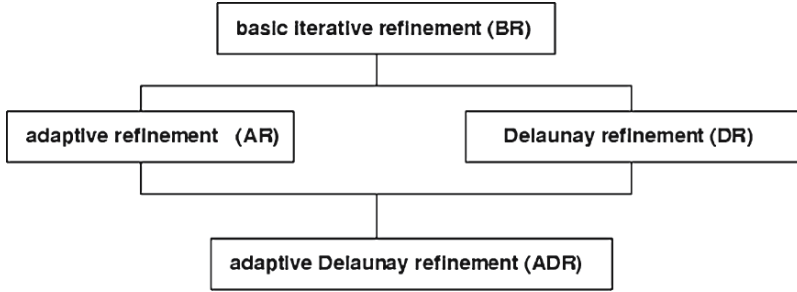


Fig. 1. A hierarchy of iterative refinement methods

approximation. These methods are aware of a data function through an error measure for triangles. They require an error tolerance, $errTol$ and the method attempts to create a mesh such that the error in every triangle does not exceed $errTol$. Adaptive Delaunay refinement (ADR) methods have characteristics both of adaptive h -refinement and Delaunay refinement methods. This requires a reconciliation of the the refinement criterion on $size(T)$ in the case of Delaunay refinement with the triangle error measure in the case of adaptive h -refinement.

2.1 Basic and Adaptive Refinement Methods

To describe the basic attributes of iterative refinement methods, we use the following pseudo code algorithmic description of basic refinement.

```

1: basic iterative refinement( $M$ )
2: initialize  $S$  by the triangles of  $M$ 
3: while  $S$  is not empty do
4:   select  $T$  using  $S$ 
5:   if Refine( $T$ ) then
6:      $V_{insert} = \text{SelectNewVertex}(T, M)$ 
7:      $(S_{add}, S_{rem}) = \text{Insert}(V_{insert}, M)$ 
8:      $S \leftarrow S + S_{add} - S_{rem}$ 
9:   else
10:     $S \leftarrow S - T$ 
11:   end if
12: end while
  
```

(1)

The algorithm computes a sequence of meshes, $M_n, n = 0, 1, 2, \dots$ on the domain D . M_{n+1} is obtained from M_n by selecting a triangle T in M_n such that a method predicate, $\text{Refine}(T)$, has value True and inserting a related vertex into M_n . The input parameter is mesh M , which is identified with M_0 of the above mentioned sequence. This description uses several abstract procedures that are incompletely described by their names. Basic refinement methods can

be illustrated by simple model examples of these procedures, which have been associated with early refinement methods. We will describe the methods that we review, and those used in the computations of this paper, by specifying these processes in the child methods of the method hierarchy of Figure 1. The description uses a dynamic set, S , of triangles of M which, at any time contains the triangles of M that may trigger refinement. Perhaps a simpler explanation is that at all times $T \in M - S$ implies $\text{Refine}(T) == \text{False}$. $\text{Insert}(V, M)$ takes M_n as input argument for parameter M , and returns M_{n+1} as output argument. The procedure returns two sets of triangles; S_{add} are the triangles in M_{n+1} but not in M_n , and S_{rem} are in M_n but not M_{n+1} .

A simple instance of this basic refinement algorithm uses simple longest edge bisection of T . For this, SelectNewVertex of step 6: of (1) returns the midpoint of a longest edge of T for V_{insert} and Insert of step 7: splits T into two new triangles T_A and T_B using a new edge from V_{insert} to the vertex opposite the split edge of T . If the edge being split is internal to the mesh, then the same split is performed on the neighbouring triangle, T_{neigh} , producing two more triangles T_C and T_D in the refined mesh. The implication for step 7: of this method is that Insert returns $S_{add} = \{T_A, T_B, T_C, T_D\}$ and $S_{rem} = \{T, T_{neigh}\}$. In a 1984 paper, [18], Rivara presented a class of basic h -refinement methods using simple longest edge bisection.

It is common to implement the selection of step 4: of (1) so that larger triangles are selected before smaller ones. The need for quick access to the largest triangle in S complicates the data structure for this dynamic set, Shewchuck, [25]. The longest edge propagation path of Rivara, [19, 20], is a heuristic for finding a local maximum edge length in the mesh. For any triangle T_0 in M , the longest edge propagation path of T_0 , $\text{Lepp}(T_0)$, is the sequence $\{T_j\}_{j=0}^N$, where T_j is the neighbor triangle on a longest edge of T_{j-1} , and longest edge (T_j) > longest edge (T_{j-1}), for $j = 1, \dots, N$. This condition determines $N \geq 0$. Consequently either $\text{Lepp}(T_0)$ terminates with T_N that has a longest edge that is a constrained edge of M , (e.g. a mesh boundary edge), or it terminates with a pair of neighbouring triangles, (T_N, \bar{T}) such that their common edge is a longest edge of both. For the computations of this paper, these ideas affect step 4: of (1); i.e. a triangle T_0 is initially selected from S and then T is set to T_N of $\text{Lepp}(T_0)$. It is common then that T is not itself in S . This is a technicality of the combined use of Lepp and longest edge bisection discussed in §2.1 immediately following ; see Rivara, [19], for a discussion of it.

6: $V_{insert} = \text{SelectNewVertex}(T, M)$

The general pattern of this procedure is that it computes a candidate vertex V_{cand} for V_{insert} that will improve the configuration of triangles in the mesh near V_{insert} relative to the refinement criteria. If V_{cand} is so close to a boundary edge, e , that its insertion would result in an obtuse triangle adjacent to e , then V_{cand} is said to encroach on e . Inserting V_{cand} in this case would violate the

assumptions we have placed on the meshes of the refinement sequence. So if V_{cand} does not encroach on any boundary edge, then it can be returned as V_{insert} by `SelectNewVertex`(T, M), otherwise the midpoint of an encroached edge is returned. Two standard choices for V_{cand} are the circumcenter of T , $CC(T)$, or the midpoint of a longest edge of T , which is referred to as longest edge bisection, $LEBis(T)$. We will compare the efficiency of meshes generated by these two possible choices in §3.

Adaptive Refinement

The basic refinement methods that have just been discussed contain the geometric features of adaptive h -refinement methods for triangular meshes which starting appearing in the literature in the late 1970s¹. The context of these refinement methods was piecewise linear function approximation, $f^{(pl)}(x, y)$, typically by the finite element method, for a data function, $f(x, y)$, typically defined implicitly by a partial differential equation. The implication for our refinement method hierarchy is that the `Refine`(T) predicate uses an error estimate of some measure of the error, $err(x, y) = f(x, y) - f^{(pl)}(x, y)$. The literature on error estimation is large and continues to grow; we do not review it here. Bank and Weiser published an early paper on error estimation for this purpose, [5]. This research was done at an early stage of the sustained development by Bank and collaborators, of the pde solving software PLTMG which incorporates adaptive h -refinement, [3].

2.2 Delaunay and Adaptive Delaunay Refinement Methods

As mentioned above, for Delaunay refinement methods, all the M_n are constrained Delaunay meshes; but in addition, we require that no boundary edge is the longest edge of an obtuse triangle. For Delaunay refinement, step 7: of (1) is the familiar Delaunay vertex insertion into the mesh, M_n . Algorithmically, the update can be accomplished by a simple insertion of V_{insert} followed by a series of edge swaps, or equivalently, as the Delaunay kernel operation of George and Borouchaki, [13], page 55, which is expressed in this reference by $M_{n+1} = M_n - Cav(V_{insert})^2 + Ball(V_{insert})^3$. To relate this operation to the basic refinement algorithm, we identify S_{rem} of step 7: of (/refalgorithm) with $Cav(V_{insert})$ and S_{add} with $Ball(V_{insert})$.

What has been the motivation for developing ADR methods from basic h -refinement methods? The initial uses of iterative Delaunay refinement seem to have been motivated by the use of the circumcentre of T for (V_{insert}). Frey,

¹Perhaps the earliest reference to adaptive refinement based on LEBis is Sewell, 1979, [23]. See also Bank and Sherman, 1979, [4]; the edge based refinement of this reference is not exactly LEBis

²the cavity of V in M_n , $Cav(V)$, is the set of triangles with V in their circumcircle

³the ball of V in M_{n+1} , $Ball(V)$, is the set of triangles in M_{n+1} incident on V

1987, [12], promoted it on the basis that $CC(T)$ was a new vertex that was equidistant from the vertices of T , and a longer distance from any other nodes of M , and other authors have concurred, e.g. Peraire et al, 1987, [16].

Another motivation came from proofs that the Delaunay incidences minimized the interpolation error for $f(x, y) = x^2 + y^2$ in a number of error measures, over a given set of vertices, e.g. D’Azevedo and Simpson, 1989, [11] and Rippa, 1992 [17], The implication was that for isotropic errors, a maximal efficiency mesh would be a Delaunay mesh. Of itself, this is not a very convincing motivation because most data, and errors, are anisotropic. The extension of these ideas to anisotropic errors was also recognized in the applications literature e.g. Mavriplis, 1991 [14], and the mesh theory literature. Anisotropic errors are minimized by meshes that are Delaunay in appropriate stretched coordinate systems (see §5).

There are also motivations from the benefits of using Delaunay meshes for the discretization of PDEs by either the FEM or FVM that we do not discuss here, e.g. Shewchuck, 2002 [24], and Sukumar, 2003, [29].

A characteristic of Delaunay meshes is that they contain the most equi-angular, and hence most equi-lateral, triangles of any vertex connectivity of a triangular mesh. This is a shape benefit that is not tied to the choice of $CC(T)$ for insertion as was noted by Baker, 1994, [2]. Rivara and Palma, 1997, [21] and [19] reported combining *Lepp* and the choice of $LEBis(T)$ for V_{insert} with Delaunay insertion. Borouchaki and George presented in the same year, [7], an edge based Delaunay refinement scheme that uses many features similar to those that we have discussed above.

Delaunay refinement research was simultaneously being carried forward by the momentum of research in computational geometry. This research extended the **Refine** predicate to include a requirement that the minimum angle in the triangles exceed a specified angle tolerance, $angTol$. I.e.

$$\mathbf{Refine}(T) \equiv \mathbf{True} \iff size(T) \geq sizeTol \text{ or } min\ angle(T) \leq angTol \quad (2)$$

Chew, 1993, [8], Ruppert, 1995, [22], and Shewchuck, 1996, [25], developed Delaunay refinement algorithms based on $V_{cand} = CC(T)$, and suitable encroachment rules, that were proven to terminate with M_N satisfying $min\ angle(M_N) \geq angTol$ for $angTol$ values up to about 30° . Methods proven to terminate satisfying this angle constraint are referred to as quality methods for mesh generation. Based on this research, Shewchuck produced the widely used quality Delaunay refinement code, Triangle, [25]. Using $size(T) = R_{cc}(T)$, $V_{cand} = CC(T)$, and a largest T first ordering for the selection in step 4, Baker, [2], gave an alternative proof that Delaunay refinement with the given encroachment rule, terminates satisfying $min\ angle(M_N) \geq 30^\circ$.

3 ADR

In this section, we provide a detailed study of ADR using the data function

$$compexp(x, y) = (1 + e^{2\pi x} \cos(2\pi y)) / (2e^{2\pi}) \tag{3}$$

as the working example. Because this function is the real part of complex valued function $(1 + e^{2\pi z}) / (2e^{2\pi})$, we will refer to it as the complex exponential function. $f^{(pl)}(x, y)$ is the piecewise linear interpolant of f and the error measure to be used is

$$|e|_M = \max_{T \in M} |err|_{2,A}(T) \tag{4}$$

where $|err|_{2,A}(T)$ is a computable estimate of the average L_2 error over triangle T ,

$$||err||_{2,A}(T) = \left(\int_T (f - f^{(pl)})^2 dA / A(T) \right)^{\frac{1}{2}} \tag{5}$$

$A(T)$ is the area of T . $|err|_{2,A}(T)$ is computed by estimating $||err||_{2,A}(T)$ using a 7 point order 4 quadrature rule for integration over T which may be found in Strang and Fix, [28], page 184. The criteria used for **Refine**(T) are those of (2) with $|err|_{2,A}(T)$ in place of $size(T)$ and similarly for $errTol$. We set $angTol = 20^\circ$; however, the angle criterion plays very little role in this study.

We introduce several notations:

$M(D, f, errTol)$ – for a mesh generated by ADR on domain D , for data function f using error tolerance $errTol$

$N_V(M)$ – for the number of vertices in mesh M .

$|err|_{2,A}(T, f)$ – for $|err|_{2,A}(T)$ if we wish to be explicit about dependence on f .

3.1 A Demonstration

For this computation, D is US , the unit square and the initial mesh, M_0 is the two triangle mesh on US . $LEBis$ is used in **SelectNewVertex**. $M(US, compexp, 10^{-3})$ is shown in Figure 2(A); the triangles of this mesh are shaded with a 10 level gray scale based on $\log_{10}(|err|_{2,A}(T))$ to show the distribution of errors. A summary histogram of this error data is also shown in Figure 2(B).

It can be seen in Figure 2(A) that only one triangle shape is present in the refined mesh; all the triangles are isosceles, right angled. We will abbreviate ‘isosceles, right angled triangle’ to IRAT. For an IRAT, the midpoint of the longest edge coincides with the circumcircle center, i.e. $LEBis(T) = CC(T)$. In Lemma 1 below, we use this to infer that, for this M_0 , the ADR methods that use either choice of V_{insert} produce the same mesh as an adaptive h -refinement method that uses simple longest edge bisection refinement. The regions of constant triangle shape and size form a series of patches with curved outlines in the mesh. The edge lengths step down by a constant factor

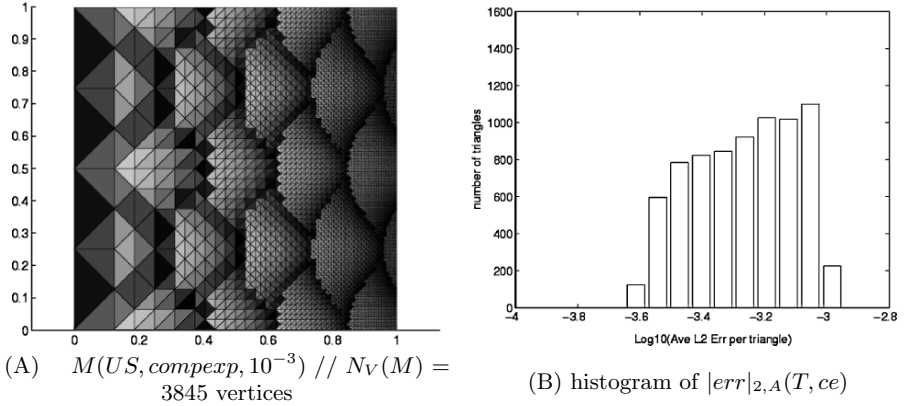


Fig. 2. ADR mesh for complex exponential, (3), with $errTol = 10^{-3}$, : 3845 vertices

of $1/\sqrt{2}$ on moving from one patch to its neighbour on the right. This pattern can be conveniently summarized by the histogram of the distribution of the $log_2(\text{longest edge})$ shown in Figure 3(A). It shows a discrete spectrum of sizes at the negative half integers. This pattern persists if we decrease $errTol$. Figure 3(B) shows the same histogram for $errTol = 1.0^{-4}$; the corresponding mesh has 37520 vertices, i.e. about 10 times the number in the mesh of Figure 3(A).

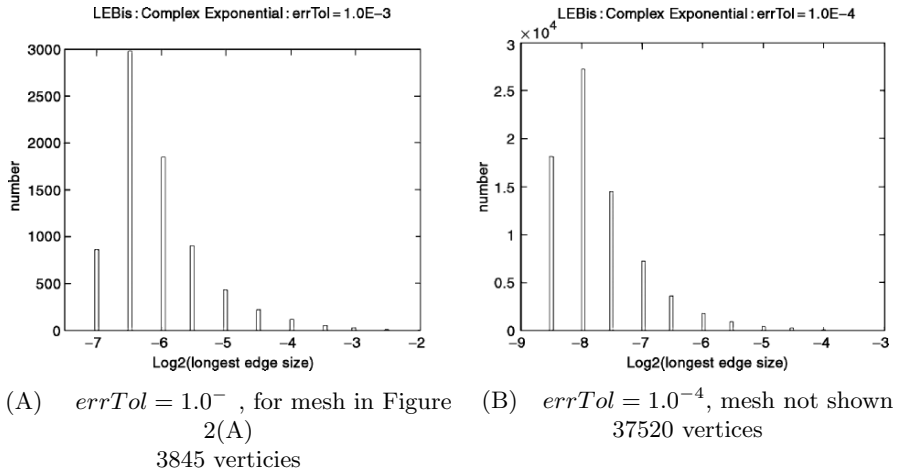


Fig. 3. Histograms of $Log_2(size(T))$ for $M(US, compe xp, errTol)$

The grey scale visualization of the distributed triangle interpolation errors of Figure 2(A) shows that on each patch of constant triangle shape and size, the errors are larger for the triangles nearer to the right side of the patch. At

the boundary of two patches, the neighbouring triangle of the larger size are shaded dark and those of the smaller size are light indicating a discrete step in error size across this boundary. The error step is of relative size $\approx \frac{1}{2}$ since the triangle edge size change is $\frac{1}{\sqrt{(2)}}$. The gradual darkening of the error grey scale across the patches is reflected in the relatively block shaped histogram of the distribution of $\log_{10}(|e|_T)$ shown in Figure 2(B).

We now show that the features of this example have some generality.

3.2 Isosceles, Right Angled Triangle Meshes

We will refer to a mesh in which each triangle is an IRAT as an IRATM. Evidently, an IRATM is a Delaunay mesh that meets the non-obtuse boundary triangle criteria for the Delaunay refinement procedure `DelRefine(M)` of §2. The next lemma shows that for a class of common Delaunay refinement methods, if M_0 is an IRATM, then all M_n of the refinement sequence are IRATMs.

Lemma 1. *Let M be an IRATM, and let T be a triangle selected using by either `Lepp` or largest triangle first ordering. Let V_{insert} be either `LEBis(T)` or `CC(T)`, and let M' be the result of Delaunay insertion of V_{insert} into M . Then M' is an IRATM.*

Proof If the longest edge of T is a boundary edge, then the lemma is clearly true. Assume that the longest edge of T is not on the boundary, and let \bar{T} be its neighbour on this edge. Then \bar{T} cannot be smaller than T . But since T has been selected by either `Lepp` or largest first ordering, \bar{T} cannot be larger than T . So we conclude that \bar{T} is the same size as T and the two triangles form a square. `SelectNewVertex` will choose the centre of the square for V_{cand} . V_{cand} does not encroach on any boundary edge, nor does it lie in the circumcircle of any mesh triangle other than T and \bar{T} . Hence $V_{insert} = V_{cand}$ and `DelaunayInsert` breaks $T \cup \bar{T}$ into 4 IRATs in M' and all other triangles in M' are unchanged from M .

Corollary 1. *Let M_0 be an IRATM, then the same mesh is obtained by the following three refinement methods applied to M_0 :*

- i) adaptive h -refinement with simple longest edge bisection*
- ii), iii) ADR with either `CC(T)` or `LEBis(T)` for V_{insert}*

provided that either `Lepp` or largest triangle first ordering is used in the selection of T for refinement and the empty diametral circle encroachment rule (,or no encroachment rule,) is used.

This observation muddies the water for establishing a general merit of combining Delaunay insertion with adaptive h -refinement to get ADR, or some merits of using one of `LEBis(T)` or `CC(T)` in preference to the other.

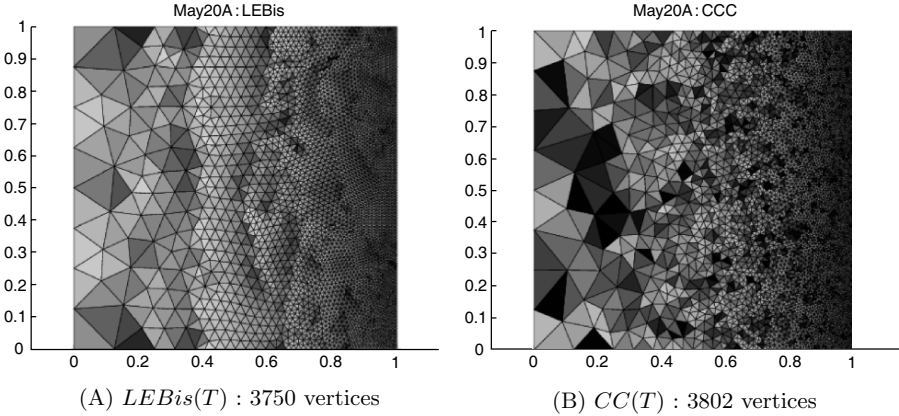


Fig. 4. ADR with strong encroachment test: $errTol = 10^{-3}$

Any such theoretical statements of merit would have to be contingent on the choice of initial mesh.

Before you try this with your favourite ADR code, note that, while mathematically correct, the proof of the lemma depends heavily on exact arithmetic. For IRATs, the circumcircle of T is the diametral circle of its longest edge. So empty circle testing and encroachment testing are basically the same. The binary outcome of the exact arithmetic encroachment test is, however, lost in floating point computation. If inexact arithmetic is used, there are in effect three outcomes:

1. the vertex is definitely not inside the diametral circle
2. it is definitely inside the circle
3. the test is not definitive.

In such arithmetic, the property that ADR maps an IRATM into a bigger IRATM will hold if we define encroachment to occur only if the vertex is definitely inside the diametral circle. We will refer to this as the weak encroachment test. For the meshes created in §3.1, we used this criterion. If, however, we change the criterion to be that encroachment takes place unless the vertex is definitely outside the diametral, circle, then we get the strong encroachment test. Two things ensue from using the strong test in `SelectNewVertex`. We do not get a sequence of IRATM meshes, and there is a difference between using the $LEBis(T)$ choice of V_{cand} and the $CC(T)$ choice. In Figure 4(A) and (B), we show the two meshes generated by the strong test and the $LEBis(T)$ choice of V_{cand} in (A) and the $CC(T)$ choice in (B). While the meshes are evidently not IRATMs, and clearly differ, the sizes are not significantly different, i.e. the variation between the meshes in Figures 2(A), 4(A), and 4(B) is less than 12%.

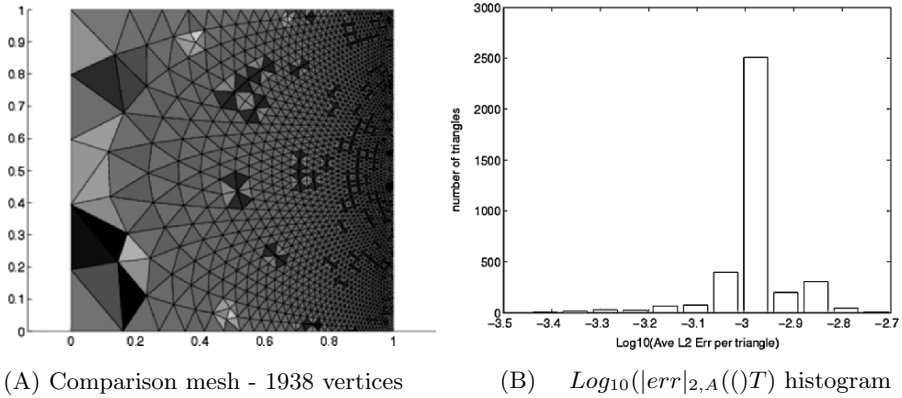


Fig. 5. Comparison mesh for complex exponential; (3)

3.3 A Comparison Mesh

In §3.1, we noted that the light shading of the error gray scale in the triangles on the right sides of the uniform mesh patches of Figure 2(A) indicate some inefficiencies in this mesh. These triangles are smaller than necessary, which also shows in the block structure of the error histogram in Figure 2(B). We can get some insight into the extent of this inefficiency by creating a comparison mesh, M_{comp} using techniques discussed in the next section that are based on some theory of mesh efficiency which is reviewed in §5. For $errTol = 10^{-3}$, M_{comp} is shown in Figure 5(A) and $NV(M_{comp}) = 1938$. A histogram of $\log_{10}(|err|_{2,A}(T))$ is shown in Figure 5(B). This histogram shows that the mesh is strongly equidistributing; i.e. that $|err|_{2,A}(T) \approx errTol$ for most of the triangles in M_{comp} . This histogram also shows that M_{comp} is not, strictly speaking, a feasible mesh; i.e. it contains some triangles that do not meet the error tolerance. Meshes that are fully equidistributing to a specified error tolerance are theoretically possible. But constructing them is as difficult as constructing M_{MaxEff} . Here we propose M_{comp} as an indication that a M_{MaxEff} has about 1900 to 2000 vertices. This is evidence for our conclusion that the ADR generated meshes, $M(US, ce, 10^{-3})$ that we have shown are about twice as big as necessary.

3.4 A Modified Domain: The Hollow Square

Perhaps efficiency factor of about 2 that we observed for M_{comp} compared to meshes of Figures 2 or 4 is due to taking D as the unit square and /or M_0 as an IRATM. In this section, we look at an alternative $D = HolloSq$ which is the hollow square created by removing the square from $(.2,.2)$ to $(.8,.8)$ from the unit square as shown in Figure 6. M_0 is the 8 triangle mesh on HS .

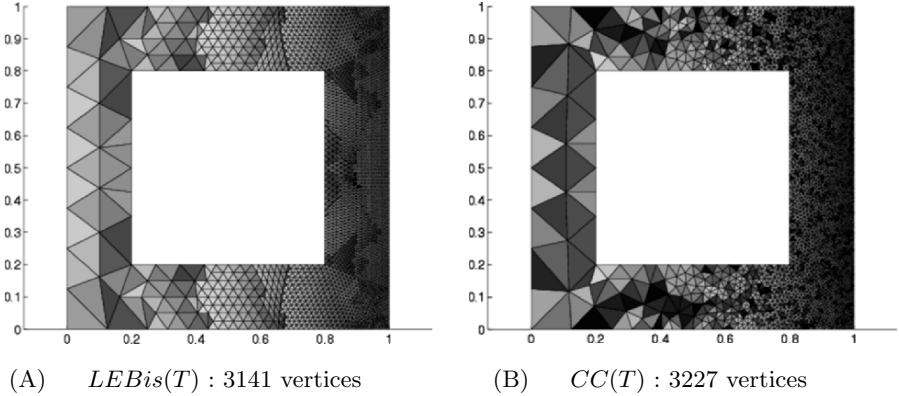


Fig. 6. $M_{LEBis}(HolloSq, compe xp, 10^{-3})$ and $M_{CC}(HolloSq, compe xp, 10^{-3})$

Figure 6 (A) shows $M_{LEBis}(HolloSq, compe xp, 10^{-3})$ with 3141 vertices, created using $LEBis(T)$ and Figure 6 (B) shows $M_{CC}(HolloSq, compe xp, 10^{-3})$ with 3227 vertices, created using $CC(T)$. $M_{LEBis}(HolloSq, ce, 10^{-3})$ clearly shows the pattern of patches of regular submeshes that characterizes $M(US, compe xp, 10^{-3})$ of Figure 2(A) and Figure 4(A). It seems likely that this is related to the shape stability of simple longest edge bisection studied by Adler, [1]. By comparison, $M_{CC}(HolloSq, compe xp, 10^{-3})$ in Figure 6(B) shows no such patterns; it, and Figure 4(B), show relatively continuous transitions of triangle edge lengths with significantly more irregularity. These characteristics of the two vertex insertion methods were reported by Baker, [2].

Despite the difference in appearance of the the two meshes in Figure 6, they are essentially the same size i.e. neither $LEBis(T)$ nor $CC(T)$ appear to provide an efficiency advantage over the other. This effect persists for a range of $errTol$. Figure 7 shows that the sizes of $M_{LEBis}(HolloSq, compe xp, errTol)$ and $M_{CC}(HolloSq, compe xp, errTol)$ are essentially the same for $10^{-4.2} \leq errTol \leq 10^{-2.4}$. A least squares fit to this data produces the linear relationship $N_V(M) \approx 3.15/errTol$.

4 Creating a Comparison Mesh

We discuss the following steps for creating a comparison mesh like that of Figure 5(A).

- 1) Introduce an appropriate new coordinate system by a transformation

$$(u, v) = G(x, y) \tag{6}$$

which maps D in the (x, y) plane 1-1 onto a domain \bar{D} in the (u, v) plane. Let the inverse transformation be $(x, y) = g(u, v)$.

2) Define data function

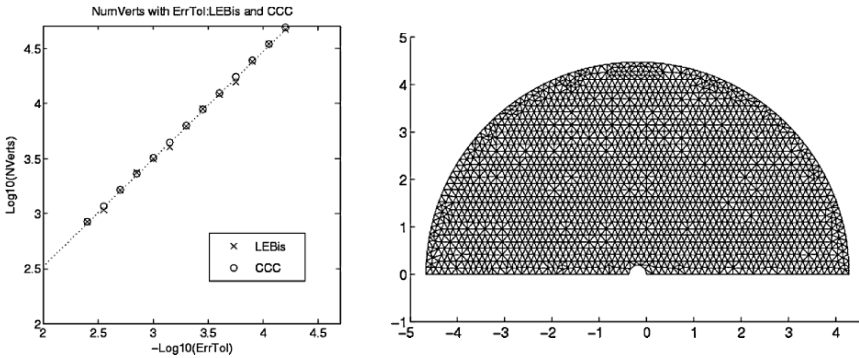
$$F(u, v) = f(g(u, v)) \text{ for } (u, v) \in \bar{D} \tag{7}$$

for $(u, v) \in \bar{D}$ to be the function corresponding to f invariant under (6). Create an appropriate mesh, $\bar{M}_{comp} = M(\bar{D}, F, errTol)$

3) Using $(x, y) = g(u, v)$, map \bar{M}_{comp} to M_{comp} on D .

Key to getting a suitable comparison mesh are the interpretations of ‘appropriate’ as it appears in steps 1) and 2) of (7). These explanations are based on some theory of maximal efficiency meshes which is, unfortunately, incomplete and technically complex, even for two dimensions. We give an overview of this theory in §5. The choices of appropriate $G(x, y)$ and \bar{M}_{comp} are quite problem specific and customized; so these techniques for creating M_{comp} are not practical general mesh generation techniques.

Note that in looking for efficient meshes for plinear interpolation, we are not concerned with whether they are Delaunay meshes or not. While we do create a Delaunay mesh for \bar{M}_{comp} in step 2, the inverse mapping of step 3 does not ensure that M_{comp} is Delaunay, nor do we care.



(A) $N_V(M(HS, ce, errTol)) = 3.15/errTol$ (B) $\bar{M}_{Comp}, N_V(\bar{M}) = 1938$ vertices for either *LEBIs* or *CC*

Fig. 7.

For the complex exponential, D’Azevedo, [9], showed that an appropriate transformation is⁴

$$\begin{aligned} u &= G_1(x, y) = \sqrt{20}/e^\pi (e^{\pi x} \cos(\pi y) - 1) \\ v &= G_2(x, y) = \sqrt{20}/e^\pi e^{\pi x} \sin(\pi y) \end{aligned} \tag{8}$$

⁴Actually (8) is the transformation computed by D’Azevedo rotated through 90°.

The image of the unit square in the (x, y) plane under this transformation is shown in Figure 7(B). It is a semi-ring in the upper half of the (u, v) plane, centred on $(-c, 0)$, for $c = \sqrt{20}/e^\pi = .1933\dots$ and having inner radius c , and outer radius $2\sqrt{5} = 4.472\dots$

Figure 7(B) also shows the version of \overline{M}_{comp} used as per step 2 of (7) to create M_{comp} , shown in Figure 5(A), as per step 3. To create \overline{M}_{comp} , we use a small amount of ADR applied to an initial Delaunay mesh, \overline{M}_0 . \overline{M}_0 is constructed from a regular grid of vertices controlled by a grid spacing parameter, h . The boundary vertices form a uniformly spaced partition of $\partial\overline{D}$ of spacing $\approx h$, and the internal vertices lie on a grid $(j h, k \sqrt{5}h)$ for integers j and k corresponding to grid points inside \overline{D} at a distance greater than h from the boundary. ADR is then used to reduce the size of the triangles of \overline{M}_0 to meet an error criterion, \overline{errTol} for F , while retaining much of the desired shape in the resulting triangles. We pick h large enough that \overline{M}_0 requires at least every internal triangle to be refined at least once.

This example can provide some simple intuitive insight into the term ‘appropriate’ for G of step 1 and \overline{M}_{comp} of step 2 in this case. We know that the mesh spacing in M should be small on the $x = 1$ boundary of the unit square (for accuracy) and should be large on the $x = 0$ boundary (for efficiency). If such a mesh is to be the transform of an essentially uniform mesh on \overline{D} , then the image of the $x = 1$ boundary must be much longer than the image of the $x = 0$ boundary. G accomplishes this by mapping the $x = 1$ boundary to the large outer circle, and the $x = 0$ boundary onto the small inner circle, of the boundary of \overline{D} .

As described in §5, the ideal comparison mesh would be a regular mesh of essentially uniform triangles of appropriate shape. Exact uniformity is not possible because of the geometry of \overline{D} and not useful because of the approximate relation between $|err|_{2,A}(\overline{T}, F)$ and $|err|_{2,A}(T, f)$ when T and \overline{T} are images under (8). Step 2 at (7) has two continuous control parameters, h and $errTol$, that can be tuned to control the size of M_{comp} and the spectrum of $|err|_{2,A}(T)$. However, the variation of these outcomes with the control parameters is only approximately continuous; there are discrete jumps in behaviour due to the ‘stiffness’ of essentially uniform meshes. For this reason, we have presented the mesh of Figures 5(A) and 7(B) as a comparison mesh even though it is not feasible, rather than a feasible comparison mesh with a spectrum that peaks significantly below the error tolerance of 10^{-3} .

5 An Overview of Some Theory of Maximal Efficiency Meshes

In the introduction, we describe a goal of adaptive h -refinement as producing meshes that locate the vertices efficiently for the control of the piecewise linear approximation purpose of the mesh. We also mention that there is a limit to how efficiently this can be achieved. The mathematical formulation of this

limit is a maximal efficiency mesh, M_{maxEff} , which minimizes the number of vertices over the set of meshes for which $\max_{T \in M} (\|err\|_{2,A}(T)) \leq errTol$. We can view ADR as a method for generating meshes, $M(D, f, errTol)$ that are feasible meshes for the constrained optimization problem of determining M_{maxEff} . In this section, we give an overview of some of the theory pertaining to M_{maxEff} . We start with the simple case of quadratic f , then discuss what can be said for more general functions, and then look at harmonic functions in particular.

5.1 The Quadratic Data Function Model

Much of our understanding of max efficiency meshes is guided by analysis of $f(x, y)$ as a quadratic polynomial, which has produced some rigorously correct results and helpful insights. Three key components of the study of optimal meshes for quadratic f are

- a) an explicit formula for $\|err\|_{2,A}(T, f)$ that shows its dependence on the size and shape of T
- b) an affine transformation to a new coordinate system that reduces the formula of a) to one of two canonical cases.
- c) a characterization of the maximal efficiency triangle shapes for the two canonical cases.

The transformation of b) is then used to reduce optimal meshing problems to one of two canonical optimal meshing problems. The characterization of c) provides some indication of the nature of solutions to these problems, including some special case, non-typical solutions.

Component a): Error in Linear Approximation of Quadratic Data

Let $f(x, y) = \frac{1}{2}H_{1,1}x^2 + H_{1,2}xy + \frac{1}{2}H_{2,2}y^2$ where H is the constant Hessian of f . In developing a formula for $\|err\|_{2,A}(T, f)$ for this quadratic f , Nadler [15], introduces quantities, d_j , associated with the j th edge of T which can be expressed in native coordinates, (x, y) , by the quadratic form

$$d_{j+1} = \frac{1}{2}(P_{j+1} - P_j, H(P_{j+1} - P_j)) \text{ for } P_j = (x_j, y_j) \tag{9}$$

The error is then given by

$$\|err\|_{2,A}(T, f) = \frac{1}{180}((d_1 + d_2 + d_3)^2 + d_1^2 + d_2^2 + d_3^2). \tag{10}$$

See also Berzins, [6]. Note from (10) that d_j has units of error; so they are invariant under affine changes of coordinates.

Component b): Canonical Forms

We now develop an affine transformation from native coordinates, (x, y) to coordinates, (u, v) in which the d_j have a canonical form. These new coordinates are commonly called stretched coordinates. Because H is symmetric, there is an orthogonal matrix, M such that $M^t H M$ is diagonal, D ; the diagonal entries of D , D_k $k = 1, 2$ being the eigenvalues of H . Consequently, we can rewrite (9) as

$$d_{j+1} = \frac{1}{2}(M (P_{j+1} - P_j), D M (P_{j+1} - P_j))$$

$M P_j$ are rotated coordinates of P_j which we will denote by (p_j, q_j) . Assuming that $|D_1| \leq D_2$, we can then write

$$d_{j+1} = \frac{1}{2}D_2 (a_2(p_{j+1} - p_j)^2 + (q_{j+1} - q_j)^2) \tag{11}$$

where $a_2 = D_1/D_2$, so $|a_2| \leq 1$. For simplicity, we assume $a_2 \neq 0$, avoiding the degenerate case.

The transformation to stretched coordinates takes the form

$$\begin{pmatrix} u \\ v \end{pmatrix} = \sqrt{D_2} \begin{pmatrix} \sqrt{|a_2|} & 0 \\ 0 & 1 \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} G_1(x, y) \\ G_2(x, y) \end{pmatrix}. \tag{12}$$

It is useful to introduce the anisotropy ratio, a , which carries the sign of a_2 , as $a = \text{sign}(a_2) \sqrt{|a_2|}$ Under this transformation, we have the following forms for d_j

$$\begin{aligned} \text{if } a > 0 \quad d_{j+1} &= \frac{1}{2}((u_{j+1} - u_j)^2 + (v_{j+1} - v_j)^2) \quad \text{the definite case} \\ \text{if } a < 0 \quad d_{j+1} &= \frac{1}{2}(-(u_{j+1} - u_j)^2 + (v_{j+1} - v_j)^2) \quad \text{the indefinite case} \end{aligned} \tag{13}$$

Stretched coordinates are useful here because the maximal efficiency meshing problem is invariant under (12) in the sense that a solution in one coordinate system transforms to a solution in the other. Let us note the details involved in this assertion. If we continue to use $F(u, v) = f(g(u, v))$ as introduced in §3.3, we can regard functions (f, D) and (F, \bar{D}) as one abstract quadratic function, invariant under (12). However, by virtue of (13), we can see that from the point of view of error behaviour, we can assume

$$\begin{aligned} F(u, v) &= (u^2 + v^2)/2 \quad \text{if } a > 0 \\ &= (-u^2 + v^2)/2 \quad \text{if } a < 0 \end{aligned} \tag{14}$$

which we will denote by $F(u, v) = (\pm u^2 + v^2)/2$.

We can extend the mapping between D and \bar{D} to be a mapping of an arbitrary mesh M on D to mesh \bar{M} on \bar{D} in which each triangle $T \in M$ is mapped exactly to a triangle $\bar{T} \in \bar{M}$. In this sense, we could say that M and \bar{M} are invariant under (12). Consequently, the piecewise linear function

$F^{(pl)}(u, v)$ can be defined either as the piecewise linear interpolant of F on \overline{M} , or the transform of $f^{(pl)}(x, y)$; they are equivalent. Using this invariance of $f^{(pl)}(x, y)$ and the invariance of the d_k terms of Nadler’s formula, we can see that $\|err\|_{2,A}(T, f)$ is invariant, i.e. can be calculated in either coordinate system.

Consequently, we have the following equivalences

$$\begin{aligned} M(D, f, errTol) &\iff M(\overline{D}, F, errTol) \\ M_{maxEff} &\iff \overline{M}_{maxEff} \end{aligned} \tag{15}$$

meaning that if meshes M and \overline{M} are images under (12), then M meets the error tolerance, $errTol$ if, and only if, \overline{M} does, and M is a maximal efficiency mesh if, and only if, \overline{M} is.

Component c): Maximal Efficiency Triangles

A triangle is a maximal efficiency triangle, T_{maxEff} , if it maximizes $A(T)$ over the set of triangles such that $\|err\|_{2,A}(T) \leq errTol$. The maximal efficiency triangle for a general quadratic f is the transform of its canonical case. Both D’Azevedo, [9], and Nadler [15], identified the shapes of maximal efficiency triangles for the canonical cases. For the definite case, the maximal efficiency triangles are equilateral with any orientation. For the indefinite case, there are several isosceles shapes for T_{maxEff} ; however, their optimality is orientation dependent, which complicates the construction of \overline{M}_{maxEff} for the indefinite case. One of these shapes is the isosceles triangle with horizontal base and height to base ratio $\sqrt{5}/2$.

If, per chance, domain \overline{D} could be tiled with maximal efficiency triangles for $errTol$, then the resulting mesh would be \overline{M}_{maxEff} for \overline{D} . Then, in turn, the mesh obtained by transforming \overline{M}_{maxEff} to the domain D would be M_{maxEff} for $(D, f, errTol)$.

5.2 More General Data Functions

If $f(x, y)$ is not quadratic, then, assuming that f is smooth, it can be approximated near P by a quadratic based on the Hessian, $H_f(P)$. It is common in meshing algorithms to use the quadratic f theory locally, i.e. for triangles that are small enough to qualify as being ‘near P ’. However, this is of limited value in the global view of meshing needed for the theory of maximal efficiency meshes. So we turn to some classical differential geometry for the tools we need.

If the line segment from P_j to P_{j+1} is considered to be infinitely short, i.e. by setting $P_{j+1} = P_j + dP$ for differential dP , then, in native coordinates, (9) becomes the differential edge length formula

$$ds^2 = (dx, H_f(P) dx) \tag{16}$$

In the case that $H_f(P)$ is positive definite, (16) is the Riemannian geometry formula for differential arc length for metric tensor H_f . For the case in which $H_f(P)$ is indefinite, (16) is the formula for differential arc length in a Minkowski, or hyperbolic, geometry. Classical differential geometry has established that for particular classes of metric tensors, it is possible to define stretched coordinates globally for the domain by transformations $(u, v) = G(x, y)$. The differential edge length formulae in the (u, v) coordinates are either Euclidean, i.e. $ds^2 = du^2 + dv^2$, or canonical hyperbolic, i.e. $ds^2 = -du^2 + dv^2$, Sokolnikoff, [27], Chapter 2. These are the same two canonical cases as we identified at (13).

In [9], D’Azevedo determined sufficient conditions on H_f for a global transformation to stretched coordinates to exist, and describes a procedure for constructing $G(x, y)$. These conditions are met, in particular, by harmonic functions and the procedure applied to the complex exponential data function, (3) results in essentially the transformation (8).

Since these transformations to stretched coordinates are not generally affine, triangles in stretched coordinates are mapped onto ‘curved triangles’ i.e. three sided patches connecting the images of the vertices with curved sides. In particular, $\|err\|_{2,A}(T, f)$ is not invariant. Consequently, the problems of determining M_{maxEff} for D, f and $errTol$ and \bar{M}_{maxEff} for \bar{D}, F and $errTol$ no longer enjoy the equivalence that we noted at (15) for quadratic data functions. It is not clear whether such equivalences would hold in some asymptotic sense as $errTol \rightarrow 0$. We have observed in calculations that the ratio $|err|_{2,A}(\bar{T}, F)/|err|_{2,A}(T, f) \approx .75$ as T becomes small. This suggests that $|err|_{2,A}(\bar{T}, F)$ may not be even a consistent approximation to $|err|_{2,A}(T, f)$ for small T . These observations have implications for the construction of comparison meshes that we present in §4. We introduce a separate error tolerance for tuning the comparison mesh and we report the error statistics for M_{comp} in native coordinates, (x, y) .

5.3 Application to Harmonic Functions

For harmonic data functions $f_{yy}(x, y) = -f_{xx}(x, y)$ so H_f has the special form

$$H_f = \begin{pmatrix} r & s \\ s & -r \end{pmatrix} \tag{17}$$

for $r = f_{xx}(x, y)$, $s = f_{xy}(x, y)$. The eigenvalues of (17) are $D_2 = \sqrt{r^2 + s^2}$, $D_1 = -D_2$; hence, non-degenerate $H_f(x, y)$ is always indefinite. The anisotropy ratio, a , of (11) is exactly -1 , so, in this sense, the error metric is isotropic. However, because H_f is always indefinite, the error will show some directional dependence and maximal efficiency triangles have orientation restrictions and are not equilateral as discussed in §5.1.

The D’Azevedo global transformation to stretched coordinates can be computed for harmonic functions, and, in particular, for the complex exponential,

(3) as per (8) which was used to construct the image domain, \overline{D} in the (u, v) plane shown in Figure 7(B). Using ADR, $\overline{M} = M(\overline{D}, (-u^2 + v^2)/2, \overline{errTol})$ was created as described in §4. The initial mesh for \overline{M} had a regular interior grid which, if refined once, would be composed of maximal efficiency triangles for the indefinite case in stretched coordinates.

6 The Double Pole: A Second Example Data Function

If we look at the mesh characteristics and efficiency of ADR applied to another harmonic data function, we find that our observations of §3 for the complex exponential are confirmed. In this section, we demonstrate this using the ‘double pole’ data function

$$dp(x, y) = Re(1/(z-c)^2) = ((x-c_1)^2 + (y-c_2)^2) / ((x-c_1)^2 + (y-c_2)^2)^2 \quad (18)$$

on the unit square, for pole at $c = (1.1, .5)$.

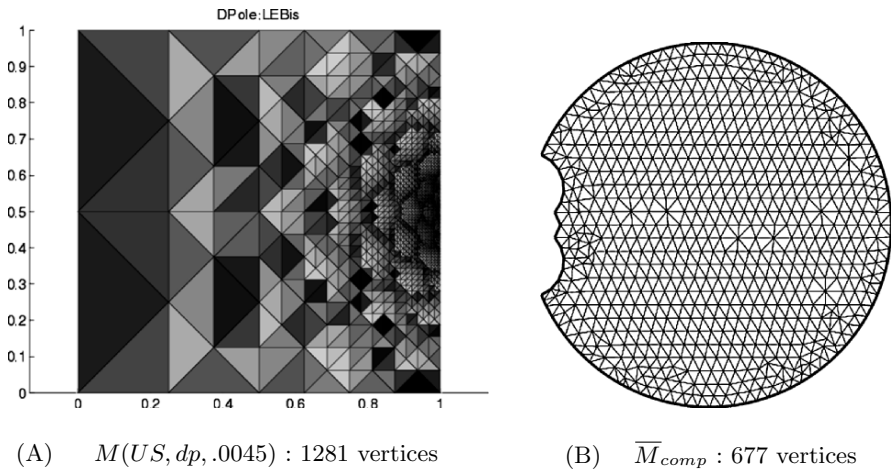


Fig. 8. ADR and comparison meshes for double pole data function

For a two triangle initial mesh, with $errTol = .0045$, and the weak encroachment criterion, all methods produce the IRATM shown in Figure 8(A) The D’Azevedo transformation for a double pole at (c_1, c_2) is

$$u = \sqrt{6}(1-(x-c_1))/d^2 ; v = \sqrt{6}(y-c_2)/d^2 \text{ for } d^2 = (x-c_1)^2 + (y-c_2)^2 \quad (19)$$

The domain, \overline{D} in the (u, v) plane corresponding to $D=$ the unit square is shown in Figure 8(B). It is bounded by arcs from four circles. These circles have centers denoted (uc_k, vc_k) and radii denote by r_k for $k = 1 \dots 4$ in the following table

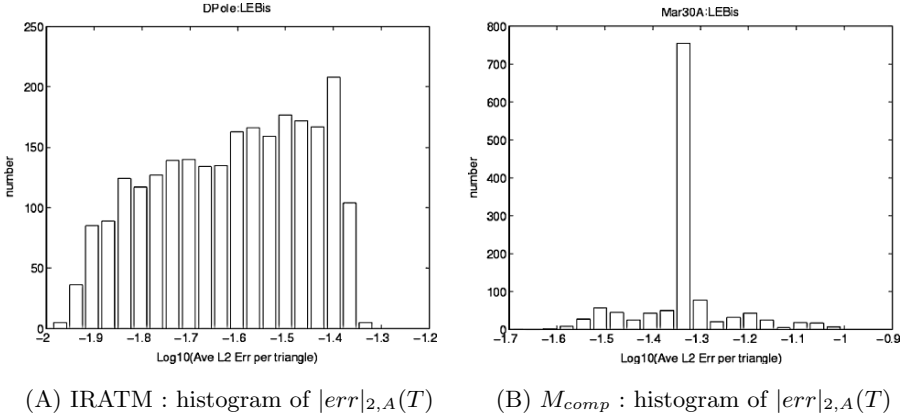


Fig. 9. Error spectra for adaptive and comparison meshes for double pole data function

$$\begin{aligned}
 (uc_1, vc_1) &= s(1, -1/(2 * c_2)) ; r_1 = s/(2c_2) \\
 (uc_2, vc_2) &= s(1 + 1/(2(c_1 - 1)) ; r_2 = s/(2|c_1 - 1|) \\
 (uc_3, vc_3) &= s(1, 1/(2 * c_2)) ; r_3 = s/(2c_2) \\
 (uc_4, vc_4) &= s(1 + 1/(2c_1)) ; r_4 = s/(2c_1)
 \end{aligned}$$

where $s = \sqrt{6}$.

The error histograms for $M(US, dp, .0045)$ of Figure 8(A)) and M_{comp} , transformed from Figure 8(B), are shown in Figures 9(A) and (B) respectively.

7 Observations and Conclusions

There are several observations that we have made that are consistent across the computations that we have described, and others that we performed in the course of this study. A primary objective was to estimate the efficiency of ADR applied to isotropic data functions and using Euclidean geometry Delaunay meshes. We observed that the meshes are typically about twice the size of maximal efficiency meshes for these cases. It seems reasonable to us that a similar efficiency would be obtained by ADR for anisotropic data, using appropriate stretched coordinates.

Our review of §2 identified several versions of adaptive refinement and in particular, we reported on computations using the alternative choices of insertion vertex, $LEBis(T)$ and $CC(T)$. In §3, we noted that the different versions all produce that same resulting mesh, if the initial mesh is an IRATM, and if the implementations in inexact arithmetic use a weak encroachment criterion. The relevance of this observation for our efficiency study is that any efficiency differences between the versions must come from features associated with the initial mesh. If the initial mesh is not an IRATM, or a strong encroachment

criterion is used, then difference in the resulting meshes were quite apparent, but differences in efficiency were insignificant, (Figures 4, 6).

The form of the error histograms for the ADR meshes provide some insight into the nature of their efficiency. Figures 2(B) (complex exponential) and 9(B) both show that $\log_{10}(errTol) - .6 \leq \log_{10}(|err|_{2,A}(T)) \leq \log_{10}(errTol)$ and $10^{-.6} \approx 1/4$. I.e.

$$errTol/4 \leq |err|_{2,A}(T) \leq errTol$$

The lower bound of $errTol/4$ for $|err|_{2,A}(T)$ was universally observed in our computations, and the distribution of $|err|_{2,A}(T)$ in the interval $(errTol/4, errTol)$ is typically fairly uniform as shown in Figures 2(B) and 9(B). So the average error is about $5errTol/8$, which is roughly consistent with the mesh having twice as many triangles as would be needed to meet the error criterion of $errTol$.

Perhaps this amount of inefficiency in ADR meshes is acceptable for many purposes. But if we want to achieve more efficiency, where should we look? Perhaps some form of smoothing could focus the error histogram distribution at its average value enough to be a significant remedy. However, it does not seem obvious how this would be achieved, and at what price. Note that D'Azevedo, [10], demonstrated a similar level of efficiency in the meshes generated by PLTMG, which incorporated a smoothing.

We have presented efficiency of ADR for refinement based on the L_2 average error, $\|err\|_{2,A}(T)$. It would be interesting to know the extent to which our observations would carry over to an energy average norm, i.e. $(\int_T (\partial err/\partial x)^2 + (\partial err/\partial y)^2 dA / A)^{1/2}$. It would also be very interesting to know about the efficiency of ADR for three dimensions. However, there are substantial difficulties in extending the technique of this paper to 3-D. In particular, neither the theory of maximal efficiency triangles for quadratic data nor techniques for computing a global transform to stretched coordinates are known.

References

1. A. Adler. On the bisection method for triangles. *Mathematics of Computation*, 40:571–574, 1983. similarity classes.
2. T. J. Baker. Triangulations, mesh generation and point placement strategies. In D. Caughey, editor, *Computing the Future*, pages 61–75. John Wiley, 1994.
3. R. E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*. SIAM, Philadelphia, 1998.
4. R. E. Bank and A. H. Sherman. The use of adaptive grid refinement for badly behaved elliptic partial differential equations. In R. Vichnevetsky and R. S. Stepleman, editors, *Advances in Computer Methods for Partial Differential Equations-III*, pages 33–39. IMACS, 1979.
5. R. E. Bank and A. Weiser. Some a posteriori error estimators for elliptic partial differential equations. *Math Comp*, 44(170):283–301, 1985.

6. M. Berzins. A solution-based triangular and tetrahedral mesh quality indicator. *SIAM J. Sci. Stat. Comput.*, 19:2051–2060, 1998.
7. H. Borouchaki and P. L. George. Aspects of 2-d delaunay mesh generation. *International Journal for Numerical Methods in Engineering*, 40:1957–1975, 1997.
8. L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. 9th Annual Comp. Geometry*. ACM Press, 1993. QA448.D38S87x.
9. E. F. D’Azevedo. Optimal triangular mesh generation by coordinate transformation. *SIAM J. Sci. Stat. Comput.*, 12:755–786, 1991.
10. E. F. D’Azevedo. On adaptive mesh generation in two-dimensions. In S Owen, editor, *Proceedings: 8th International Meshing Round Table*. Sandia National Laboratories, 1999.
11. E. F. D’Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM J. Sci. Stat. Comput.*, 10:1063–1075, 1989.
12. W. H. Frey. Selective refinement: A new strategy for automatic node placement in graded triangular meshes. *International Journal for Numerical Methods in Engineering*, 24:2183–2200, 1987.
13. P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing*. Hermes, 1998.
14. D. J. Mavriplis. Adaptive mesh generation for viscous flows using Delaunay triangulation. *Journal of Computational Physics*, 90:271–291, 1990.
15. E. Nadler. *Piecewise Linear Approximation on Triangles of a Planar Region*. PhD thesis, Brown University, 1985. order Number DA8519885.
16. J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics*, 72, 1987.
17. S. Rippa. Long and thin triangles can be good for linear interpolation. *SIAM J. Numer. Analysis*, 9:257–270, 1992.
18. M.-C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.
19. M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *Int. J. Num. Methods*, 40:3313–3324, 1997.
20. M. C. Rivara and N. Hitschfeld. LEPP-Delaunay algorithm: a robust tool for producing size-optimal quality triangulations. In *Proc. of the 8th Int. Meshing Roundtable*, pages 205–220, October 1999.
21. M.-C. Rivara and M. Palma. New LEPP-Algorithms for quality polygon and volume triangulation: Implementation issues and practical behavior. In *Trends in Unstructured Mesh Generation*, volume AMD-Vol. 220, pages 1–8. American Society of Mechanical Engineers, 1997. The Joint ASME/ASCE/SES Summer Meeting, Evanston, Illinois, USA, July 1997.
22. J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J of Algorithms*, 18:548–585, 1995.
23. G. Sewell. A finite element program with automatic user-controlled mesh grading. In R. Vichnevetsky and R. S. Stepleman, editors, *Advances in Computer Methods for Partial Differential Equations- III*, pages 8–10. IMACS, 1979.
24. J. R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In S. Owen, editor, *Proceedings: 11th International Meshing Round Table*. Sandia National Laboratories, 2002.
25. J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In ACM, editor, *First Workshop on Applied Computational Geometry*, pages 124–133. (Philadelphia, Pennsylvania), 1996.

26. R. B. Simpson. Anisotropic mesh transformations and optimal error control. *Applied Num. Math.*, 14:183–198, 1994.
27. I. S. Sokolnikoff. *Tensor Analysis. Theory and Applications to Geometry and echanics of Continua.* Wiley, 2nd edition, 1964.
28. G. Strang and G. J. Fix. *An Analysis of the Finite Element Method.* Prentice Hall, 1973.
29. N. Sukumar. Voronoi cell finite difference method for the diffusion operator on arbitrary unstructured grids. *International Journal for Numerical Methods in Engineering*, 57:1–34, 2003. Laplacian.

Mesh Optimization

Small Polyhedron Reconnection: A New Way to Eliminate Poorly-Shaped Tetrahedra

Jianfei Liu and Shuli Sun ¹

LTCS, Department of Mechanics and Aerospace Engineering, College of Engineering, Peking University, Beijing 100871, China

Abstract. Local transformation, or topological reconnection, is one of effective procedures of mesh improvement method, especially for three-dimensional tetrahedral mesh. Although the existing local transformations such as 2-3/3-2 flip are effective in removing poorly-shaped tetrahedra, it is still possible to improve the quality of mesh further by expanding the space of transformation region. The authors recently proposed a new local transformation operation, *small polyhedron reconnection* (or *SPR* for abbreviating), which seeks the optimal tetrahedralization of a polyhedron with a certain number of vertexes and faces (typically composed of 20 to 40 tetrahedral elements). In this paper, the framework of SPR approach for mesh quality improvement based on the SPR operation is presented. The main idea is to take a poorly-shaped or “worst” element as the core and construct a small polyhedron by adding 20-40 elements surrounding it, then find the optimal tetrahedralization of this small polyhedron through SPR operation. By replacing the original tetrahedra with the optimal tetrahedralization, the quality of the mesh is improved. Experimental investigations with tetrahedral finite element meshes show that the SPR approach is quite effective in improvement of mesh quality with acceptable time cost, and works well in combining with a smoothing approach. Although further researches are required for a more definite conclusion, the presented approach can be utilized as a powerful and effective tool for tetrahedral mesh generation and mesh improvement. We believe that the superior performance of the SPR approach makes it worthy of further study.

¹ Corresponding author, email: SUNSL@mech.pku.edu.cn

Keywords: Mesh improvement, tetrahedral mesh, local transformation, optimal tetrahedralization, small polyhedron reconnection (SPR)

1. Introduction

Geometrical optimization (also called node repositioning or smoothing) and topological optimization (also called local transformation or reconnection) are two main categories of mesh improvement procedure. Geometrical optimization relocates mesh points to improve mesh quality without changing mesh topology [1-7], while topological optimization changes the topology of a mesh, i.e. node-element connectivity relationship [1-3, 8, 9]. This paper will focus on the latter, local transformation.

The most frequently used and most effective operations of reconnection for tetrahedral mesh are so-called basic or elementary flips [10], e.g. 2-3 flip, 3-2 flip, 2-2 flip, 4-4 flip. These topological transformations are usually called “local”, since only a small number of tetrahedra (typically fewer than 5) are removed or introduced by a single transformation. Such flips are simple, easy to implement, but effective in removing poorly-shaped tetrahedra [2, 3, 8]. However, since these basic local transformations only simply make a selection from several possible configurations within a relatively small region composed of several tetrahedra, the effect for mesh quality improvement is limited.

In order to break such a limitation and improve the quality of mesh further, the authors recently proposed the strategy of *optimal tetrahedralization for small polyhedron* and corresponding *small polyhedron reconnection (SPR)* operation [11], which seeks the optimal tetrahedralization of a polyhedron with a certain number of vertexes and faces instead of choosing the best configuration from several possibilities within a small region that consists of a small number of tetrahedra. For a SPR operation, since the region – usually composed of 20 to 40 tetrahedral elements – is much larger than that in the local transformations mentioned,

more quality improvement is expected. Up to now, to the best knowledge of the authors, no relevant studies have been reported in other literatures.

The previous work of the authors [11] mainly emphasized the concept and the idea of the SPR operation from the viewpoint of local topological transformation. The efficiency of the SPR operation is also discussed and tested. The time complexity of the searching algorithm in the SPR operation seems too high and the computational cost may not be afforded if the size of the small polyhedron is too large. However, by some deliberate speedup strategies, the efficiency of optimal searching algorithm can be significantly enhanced and the SPR operation can be applied to practical mesh improvement with acceptable payment of time cost.

In this paper, the framework of SPR approach for mesh quality improvement based on the SPR operation is presented. The main idea is as follows. First locate a poorly-shaped or “worst” element (in sense of some quality measurement). Then take this “worst” element as the core and construct a small polyhedron by adding 20-40 elements surrounding it. Next find the optimal tetrahedralization of this small polyhedron through the SPR operation. By replacing the original tetrahedra with the optimal tetrahedralization, the quality of the mesh is improved. The cycle continues until the improvement reaches its limit, that is no better tetrahedralization existed for the small polyhedron most recently constructed.

2. SPR Operation: Optimal Tetrahedralization for Small Polyhedron

To break the limitation of the existing elementary flips, the authors recently proposed a new local reconnection strategy [11], *optimal tetrahedralization for small polyhedron*, which is illustrated in form of two-dimensional case in Fig. 1. Rather than simply making a selection from several possible configurations within a small region that consists of

a small number of tetrahedra as previous local transformation usually does, the new reconnection strategy seeks the optimal tetrahedralization of a polyhedron with a certain number of vertexes and faces. Since the region – usually composed of 20 to 40 tetrahedral elements – is much larger than that in the local transformations mentioned, more quality improvement is expected.

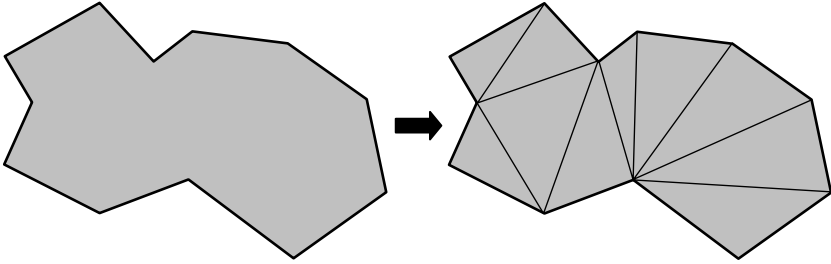


Fig. 1. Two-dimensional illustration for optimal tetrahedralization for small polyhedron

According to the strategy of optimal tetrahedralization for small polyhedron, two kinds of small polyhedron reconnection (SPR) operations are defined as follows.

SPR operation 1: For a given polyhedron with a certain number of triangles on boundary, seeks its optimal tetrahedralization without Steiner nodes added.

SPR operation 2: For a given polyhedron with a certain number of triangles on boundary, seeks its optimal tetrahedralization without Steiner nodes added under some extra geometric restrictions.

Note that the number of triangles on the boundary, S , is taken here to denote the size of the polyhedron instead of the number of tetrahedral elements. The SPR operation 2 is mainly applicable for boundary recovery and the details are not discussed here. This paper focuses on SPR operation 1 (for conciseness omits “1” in following text). In order to keep the

completeness of the method proposed, the algorithm of the SPR operation [12] is first introduced. .

First choose a triangle F on the boundary of the polyhedron P , and construct an element (denoted by ELE) by F and one of the other vertexes of the polyhedron. Thus the original polyhedron is divided into the element ELE and a new smaller polyhedron (denoted by Q). Next solve the smaller problem for the new smaller polyhedron Q by the same algorithm recursively, and then merge its result with the element ELE to get a feasible solution for the original polyhedron P . Here, the so-called feasible solution is in some sense optimal, since it includes the optimal solution of the smaller polyhedron Q . This process is repeated for all the remain vertexes. Finally choose the best tetrahedralization from all feasible solutions. Thus the final solution is exactly the optimal solution for the polyhedron P . The recursive procedure is illustrated in form of two-dimensional case in Fig. 2, and the pseudocode for the algorithm is listed in *Algorithm 1*.

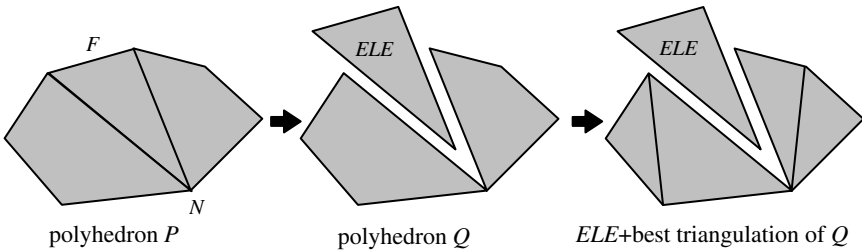


Fig. 2. Recursive procedure for the SPR operation illustrated in form of two-dimensional case (ELE + the best triangulation of $Q \Rightarrow$ a triangulation of P)

By the way, some more general and sophisticated local transformations, such as composite transformation operations [9], the general edge flip [10], may be considered as special cases of the SPR operation.

Algorithm1: The recursive algorithm of the SPR operation

int OptimalTetMeshForSmallPolyhedron (q_0, P, T)

input: q_0 , quality of the initial mesh;

P , the small polyhedron.

output: T , the best triangulation. If there is no triangulation with quality better than q_0 , T will be *NULL*.

return value: “*succeed*” or “*fail*”.

temporary variables: T_c , the best triangulation among these already tested;

q_c , quality of T_c ;

r_i , return value of the recursive call.

```

1   $q_c=q_0, T_c=NULL$ 
2  select a triangle  $F$  on the polyhedron
3  for each vertex  $N$  on the polyhedron, do
    {
4    if ( $F$  and  $N$  can construct a valid tetrahedron  $ELE$ 
5      and quality of  $ELE$  is better than  $q_c$ )
        {
6          remove  $ELE$  from  $P$ , construct a new smaller
          polyhedron  $Q$ 
7           $r_i = \text{OptimalTetMeshForSmallPolyhedron}(q_c, Q, T_c)$ 
8          if ( $r_i$  is “succeed”)
                {
9              merge  $T_c$  &  $ELE$  to create a new triangulation of  $P$ 
10             update  $T_c$  and  $q_c$ 
                }
        }
    }
11 if (a better mesh found) {  $T=T_c$ , return “succeed” }
12 else return “fail”

```

The algorithm given above is supposed to treat all of the triangulation cases and seems very time consuming. However, what's exhilarating is that many of the triangulation tries will be aborted and rejected in early stage since they produce a bad element or can not pass the valid test (such as overlapping or gap occurs). Similarly, though a large number of sub-problems appear in the searching process, many of them will be blocked by the valid test or quality test of the first element.

Moreover, if an initial triangulation for a polyhedron already exists, the optimal search process can be greatly accelerated, although the initial triangulation is not necessary for the SPR operation. This is the usual case for mesh improving.

The valid test and quality test discussed above has already made the SPR operation be able to improve quality for practical finite element meshes within acceptable time cost when the size of small polyhedron is limited to 15.

Additionally, a few further strategies are discussed in following to accelerate above searching algorithm. If a polyhedron can be subdivided into several smaller ones in digging process, it is unwise to still treat them as a whole one. We may readily achieve substantial speedup by solving several small sub-problems on smaller polyhedrons separately. Then the result is obtained by merging the result of all separate sub-problems. The testing results indicate this strategy can greatly enhance the algorithm efficiency. Furthermore a natural idea to speed the operation further is guiding the process to separate the small polyhedron as earlier as possible. So we select the digging face at the location where a tetrahedral element has just been removed.

In the thorough search algorithm, there is a drawback: the same sub-problems may be encountered several times. Here a simple strategy, storing and searching, is adopted to eliminate the repeated calculation of the same sub-problem. The sub-problems that have already been solved are stored in a bintree, in which each solved sub-problem has a record with information including its geometric description and the result of its optimal tetrahedralization. Before to be solved, any sub-problem will be searched

first in the bintree. If its record has existed, just retrieve the result and return.

3. Using the SPR Operation to Remove Slivers and Other Poorly-Shaped Tetrahedra

The SPR operation can be applied to improve the mesh quality by removing slivers and other poorly-shaped element in a step by step manner. First find the worst element according to a specific quality measure. Then construct a small polyhedron that includes the worst element and its neighbors, and perform SPR operation to find out the best tetrahedralization of this polyhedron to improve the quality of local region adjacent to the worst element. Next, find another worst element and repeat above procedure. The procedure will stop when the tetrahedralization of the polyhedron that includes current worst element can not be improved. The SPR operations are usually performed in limited times in practice and the payment for time cost is reasonable.

There are two key steps in the above procedure: 1) Construction of the small polyhedron, 2) Optimal tetrahedralization of the small polyhedron by the SPR operation.

Suppose a bad element is a sliver, that its four nodes are nearly coplanar. This element together with some other elements which share one of 2 particular sliver edges with bigger dihedral can compose a small polyhedron (Fig. 3 and *Algorithm 2*).

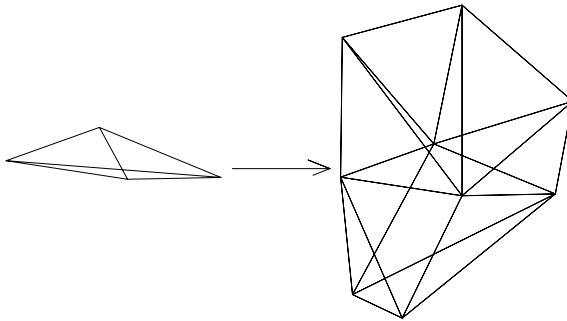


Fig. 3. A small polyhedron created surrounding a sliver

Algorithm 2: Constructing small polyhedron surrounding the worst element

Input: mesh M , the worst element E of M .

Output: a small polyhedron P with E as its core.

- 1 Calculate the six dihedral angles of E .
- 2 Get the two maximum angles, suppose their related edges are ab and cd .
- 3 Get all elements around ab and cd .
- 4 Merge all those elements to produce polyhedron P .

Usually, applying a SPR operation to such a polyhedron will eliminate thin element appropriately. In some cases, the polyhedron created is not big enough to produce a good local mesh. More elements should be included to mend the polyhedron, which is supposed to make the polyhedron something smoother. Our testing shows that using the small polyhedrons created in our algorithm can usually improve the mesh fairly well in a low time cost.

By the way, we also notice recent work of Moore and Saigal [13] to deal with sliver shaped elements in 3-dimensional finite element models, which first merges the slivers with neighboring elements to create a polyhedron, and then subdivides the polyhedron into a collection of local tetrahedra by connecting a temporary centroidal node added to all of the external triangular facets of the polyhedron, rather than searching for the best triangulation of the polyhedron without extra node added as we proposed.

4. Examples and Discussions

Several examples of finite element mesh are given to demonstrate the effectiveness of the presented SPR approach. The size of the small polyhedron, S , defined by the number of triangles, is set to 25. The finite element meshes in examples are generated by tetrahedral mesh generation

package **AutoMesh3D** [14] through the ball-packing method [15, 16]. The presented SPR procedure is already embedded in **AutoMesh3D**. The γ coefficient [2, 17] is adopted as the quality measure for tetrahedral element. Most elements with γ less 0.1 are slivers in following examples. All tests are performed on the following platform: A Pentium IV PC (2.4 GHz CPU and 256 MB RAM) with compiler of Visual C++ 6.0.

The first finite element mesh shown in Fig. 4 consists of 22392 nodes and 113975 tetrahedral elements initially. Its quality is not good. There are 34 elements with the quality value below 0.03, and the lowest value is 0.00118. The statistics of initial quality and quality after the presented SPR approach are listed in Table 1, which shows remarkable improvement of mesh quality by the SPR approach. The minimum value of γ increases to 0.321. The substantial improvement in quality of large number of elements indicates that, as a new local transformation procedure, the SPR approach works effectively on optimizing mesh topology around the worst element, and hence improves the quality of whole mesh. In this example, the SPR operations with total number of 5754 are performed, and the running time (about 260 seconds) is acceptable considering substantial improvement in mesh quality.

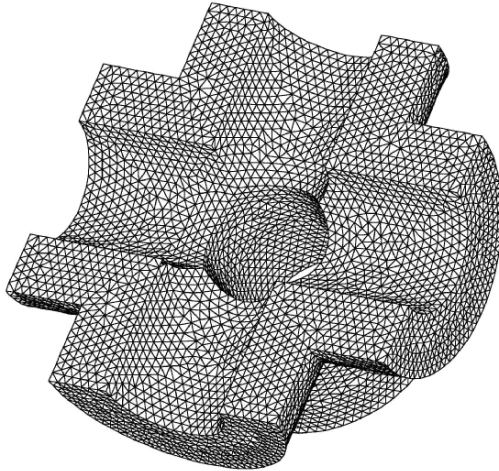


Fig. 4. The first finite element mesh

Table 1. Statistics of the quality distribution of elements in the first mesh

Range of γ	0.00~0.03	0.03~0.12	0.12~0.30	0.30~0.66	> 0.66
Initial mesh (min. γ 0.00118)	34	423	1763	13526	98229
After SPR only (min. γ 0.321)	0	0	0	10495	100975

The second finite element mesh includes 2726 nodes and 8359 tetrahedral elements initially (Fig. 5). Its quality is also not good enough. There are 13 elements with the quality value below 0.03, and the lowest value is 0.0036.

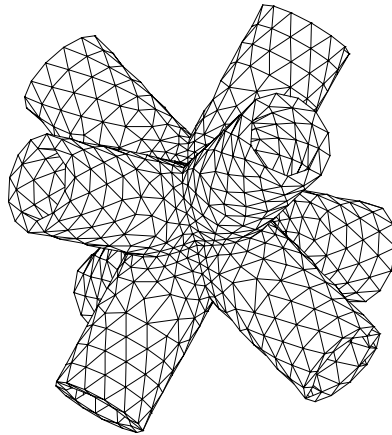


Fig. 5. The second finite element mesh

The elementary local transformations (or ELT for abbreviating) and the presented SPR approach are applied to the initial mesh, respectively. Due to the restriction of the geometry, the small polyhedrons created in the SPR operation are not big enough. Thus the benefit to mesh quality improvement is not evident compared with example 1. Table 2 shows the statistics of initial quality and quality after optimization. Both ELT and SPR procedures improve the mesh quality; however, as expected, the SPR

Table 2. Statistics of the quality distribution of elements in thesecond mesh

Range of γ	0.00~0.03	0.03~0.12	0.12~0.30	0.30~0.66	> 0.66
Initial mesh (min. γ 0.0036)	13	33	188	2500	5625
After ELT only (min. γ 0.181)	0	0	33	2304	5726
After SPR only (min. γ 0.275)	0	0	4	2358	5598

approach gives better result. The minimum value of γ increases from 0.0036 to 0.275 and there are only 4 elements with quality value lower than 0.30. The running time for SPR is about 7.5 seconds. We believe that the superiority in effectiveness makes the SPR approach more useful and become a potential replacement for previous local transformations in mesh topological optimization.

The results of above examples indicate that the proposed SPR procedure is able to significantly improve the quality of tetrahedral mesh. In practice, the topological modification and node reposition should be combined together to get more effective results. In next example, it can be seen that the combination of the proposed SPR procedure and smoothing will achieve substantial improvement in mesh quality.

The third finite element mesh illustrated in Fig. 6 consists of 11007 nodes and 53710 tetrahedral elements, and the minimum value of γ is 0.0110 initially. First, ELT and SPR procedures are applied to the initial mesh, respectively. The results listed in Table 3 indicate that the mesh quality has only limited improvement after ELT or SPR procedure. Almost same results are obtained for the two approaches. The minimum value of γ increases from 0.0110 to 0.0195. It is found that, by monitoring the optimization procedure, the processes for both approaches are quickly blocked by the same worst element, since no further improvement can be made by topological modification alone to the local small polyhedron that includes current worst element.

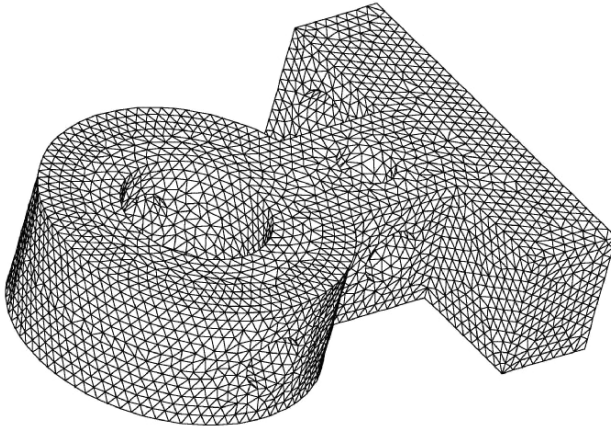


Fig. 6. The third finite element mesh

Table 3. Statistics of the quality distribution of elements in the third mesh

Range of γ	0.00~0.03	0.03~0.12	0.12~0.30	0.30~0.66	> 0.66
Initial mesh (min. γ 0.0110)	26	159	794	7605	45126
After ELT only (min. γ 0.0195)	11	159	794	7601	45130
After SPR only (min. γ 0.0195)	11	159	794	7600	45131
After ELT + smoothing + ELT (min. γ 0.0990)	0	1	494	11211	41988
After SPR + smoothing + SPR (min. γ 0.332)	0	0	0	9948	42598

In order to obtain further improvement in mesh quality, smoothing or node reposition is applied to combine with topological optimization. Here, an efficient smoothing approach based on chaos searching algorithm [5] is adopted. The running time for smoothing procedure is 76 seconds. After smoothing procedure, ELT and SPR procedures are performed respectively again. The direct effect on quality improvement by smoothing is not very distinct; however, the smoothing procedure has optimized node

distribution or configuration around the worst element, and such an improvement provides favorable conditions for topological optimization and makes topological optimization work more effectively. It can be seen from Table 3 that both ELT and SPR procedures do actually take effect after the smoothing procedure. Similarly, the SPR procedure gives much better result while the running time of 120 seconds is acceptable. The minimum value of γ increases to 0.332.

Compared with ELT, the presented SPR approach is obviously more suitable for combining with smoothing approach, and combination of SPR and smoothing approach is a better choice for mesh improvement. The time cost of SPR approach is reasonable and worthy to be paid.

It can also be observed in above examples that the number of elements generally decreases by several percentages after topological optimization, since most of the bad elements which usually occupy small volumes are removed.

By the way, same quality measure should be adopted in smoothing and topological transformation procedures. Otherwise the optimization process may probably suffer “zigzag” problem since some quality measures are found to induce inconsistent evaluation for quality change of element in some circumstances [5, 18].

5. Conclusion and Future Work

The small polyhedron reconnection operation is a new and very effective way to improve tetrahedral meshes. Although further speedup is expected for the searching algorithm, examples show that the presented SPR approach can be applied to practical mesh improvement with acceptable payment of time cost and is able to give much better results than the most commonly used local transformations. In addition, the presented SPR approach works well in combining with smoothing approach. We believe that the superiority in effectiveness makes the SPR approach more useful with the further speedup of its efficiency and

become a potential replacement for previous local transformations in mesh topological optimization.

The superior performance of the SPR approach makes it worthy of further study. Some works are in progress, including how to construct more appropriate polyhedron, developing more suitable data structure for supporting searching algorithm to eliminate the repeated calculation of the same sub-problem, choosing smartly digging face on the small polyhedron where the new elements are to be created, selecting the optimal digging directions and subdividing the polyhedron into several sub-polyhedrons as earlier as possible, etc. If a good tetrahedralization can be obtained in early stage, it will stop many unnecessary tries and block a lot of sub-problems to be created and treated.

The current work on the SPR approach mainly focuses on isotropic mesh. If changing the distance metric in quality measurement, the SPR approach might be extended to improvement of anisotropic mesh.

Acknowledgements

The authors would like to thank reviewers for their constructive comments and suggestions.

References

1. Zavattieri PD, Dari EA, Buscaglia GC (1996) Optimization strategies in unstructured mesh generation. *Int J Numer Meth Engng*, vol 39, no 12: 2055-2071
2. Lo SH (1997) Optimization of tetrahedral meshes based on element shape measures. *Comput Struct*, vol 63, no 5: 951-961
3. Freitag LA, Ollivier-Gooch C (1997) Tetrahedral mesh improvement using swapping and smoothing. *Int J Numer Meth Engng*, vol 40, no 21: 3979-4002
4. Freitag LA, Ollivier-Gooch C (2000) A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *Int J Comput Geom Appl*, vol 10, no 4: 361-382

5. Sun SL, Liu JF (2003) An efficient optimization procedure for tetrahedral meshes by chaos search algorithm. *J Comput Sci Technol*, vol 18, no 6: 796-803
6. Chen ZJ, Tristano JR, Kwok W (2004) Construction of an objective function for optimization-based smoothing. *Engineering with Computers*, vol 20, no 3: 184-192
7. Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M (2005) Variational tetrahedral meshing. *ACM Transactions on Graphics*, vol 24, no 3: 617-625
8. Joe B (1991) Construction of three-dimensional Delaunay triangulations using local transformations. *Comput Aided Geom Design*, vol 8: 123-142
9. Joe B (1995) Construction of 3-dimensional improved-quality triangulations using local transformations. *SIAM J Sci Comput*, vol 16, no 6: 1292-1307
10. George PL, Borouchaki H (2003) Back to edge flips in 3 dimensions. In: *Proceedings of 12th International Meshing Roundtable*, Sandia National Laboratories, pp 393-402
11. Liu JF, Sun SL, Wang DC (2005) Optimal tetrahedralization for small polyhedron: A new local transformation strategy for 3-D mesh generation and mesh improvement. Submitted to *CMES: Computer Modeling In Engineering & Sciences*
12. Liu JF, Sun SL, Chen YQ (2006) The algorithm and implementation of small polyhedron re-connection operation for tetrahedral mesh improvement. Submitted to *ACM Transactions on Graphics*
13. Moore RH, Saigal S (2005) Eliminating slivers in three-dimensional finite element models. *CMES: Computer Modeling In Engineering & Sciences*, vol 7, no 3: 283-291
14. Liu JF (2001) User manual of AutoMesh3D—A mesh generator for 3-D solid modeling. Technical report, Department of Mechanics and Engineering Science, Peking University (In Chinese)
15. Liu JF (1991) Automatic triangulation of n-D domains. In: *Proceedings of CAD/Graphics'91*, Hangzhou, China, pp 238-241
16. Liu JF (2003) Automatic mesh generation of 3-D geometric models. *Acta Mech Sin*, vol 19, no 3: 285-288

17. Lo SH (1991) Volume discretization into tetrahedral—II. 3D-triangulation by advancing front approach. *Comput Struct*, vol 39, no 5: 501-511
18. Nie CG, Liu JF, Sun SL (2003) Study on quality measures for tetrahedral mesh. *Chinese Journal of Computational Mechanics*, vol 20, no 5: 579-582 (in Chinese)

Optimal Mesh for P_1 Interpolation in H^1 Seminorm

Jean-François Lagüe¹ and Frédéric Hecht¹

¹ Université Pierre et Marie Curie

lague@ann.jussieu.fr

² Université Pierre et Marie Curie

hecht@ann.jussieu.fr

Summary. In this paper we present one approach to build optimal meshes for P_1 interpolation. Considering classical geometric error estimates based on the Hessian matrix of a solution, we show it is possible to generate optimal meshes in H^1 seminorm via a simple minimization procedure.

1 Introduction

1.1 Statement of the Problem

Let Ω be a domain of \mathbb{R}^2 and u a solution of an elliptic PDE problem on Ω under Dirichlet boundary conditions on $\partial\Omega$.

The energy norm is the natural one to measure the error of a numerical approximation of the solution of this problem and the Cea's lemma [4] give a bound for the error.

Cea's lemma. If the hypotheses of the Lax-Milgram theorem are satisfied, there exists a constant C independent of the subspaces V_h such that

$$\|u - u_h\| < C \inf_{v \in V_h} \|u - v\|$$

where V_h denotes the finite element spaces, u_h the discrete solution associated with V_h and $\|\cdot\|$ the norm in the space V .

a posteriori mesh adaptation. Thanks to Cea's lemma, the error made over the mesh is bounded by the interpolation error. Hence, controlling the interpolation error allows control the approximation error. This has led to a number of a posteriori mesh adaptation procedures with metric specifications based on the control of the interpolation error ([9, 7] and the references in [13]). However, these methods are usually based on the L^∞ norm of the interpolation error instead of a more physical norm.

The exact solution is usually unknown; we only know the discrete solution u_h and reconstruction methods are used to estimate the error estimator since it is theoretically bounded by derivatives one order higher than the order of the field approximation.

Surface mesh optimization. If we consider the solution u as a Cartesian surface, the problem of minimizing the interpolation error consists in defining a mesh so that the surface mesh is as smooth as possible. This has led to the notion of geometric mesh [9] : a geometric mesh is a piecewise planar approximation of the surface so that the distance between the underlying surface and the element of the triangulation does not exceed given tolerance threshold. To make sure that a mesh is a geometric mesh, one generally introduces some measurements of quality like planarity, roughness or deviation. In particular the roughness of a piecewise linear surface is defined as the square L^2 norm of the gradient of the surface integrated on the triangulation [15]. Thus, controlling the H^1 seminorm can be seen as an optimization procedure of the quality of a surface.

In this paper, we assume that the discrete solution u_h is a P_1 continuous finite element function and that we measure the error in H^1 seminorm. We shall see that the computation of this error involves second order derivatives of the computed scalar field. One way to estimate these derivatives is to use reconstruction methods (see [14] for a comparison of recently published recovery methods).

1.2 Outline

The purpose of this paper is to describe an adaptation procedure in order to build an optimal mesh by controlling the H^1 seminorm of the interpolation error. First we go back to the H^1 seminorm of the interpolation error on one element of the mesh. We show how it can be written using the Hessian matrix of u and we recall how the second order derivatives of the computed scalar field can be estimated over the domain using recovery methods. Then an adaptation procedure based on a minimization problem is described. Eventually, analytical examples are provided to illustrate this approach.

2 Optimal Mesh

The concept of optimal mesh theoretically refers to a set of meshes. For example, if several meshes are given, one will be able to say that the optimal mesh is that for which the selected measurement is optimal. In this work, a triangulation consisting of N nodes will be said optimal in H^1 seminorm if it is the triangulation that minimizes the interpolation error among all conforming N -nodes triangulations.

3 Expression of the H^1 Seminorm of the P_1 Interpolation Error

We consider a mesh $\mathcal{T}(\Omega)$ composed of linear triangles and denote $K = [a, b, c]$ as the reference element, $\Pi_h u$ as the piecewise linear interpolate of the solution u according to $\mathcal{T}(\Omega)$. The usual \mathbb{R}^2 scalar product is denoted by $\langle \cdot, \cdot \rangle$, and the Hessian matrix of u by H_u .

3.1 Expression of the Error Gradient on One Element K

Let us write $e_K = u - \Pi_h u$ the interpolation error on element K . It vanishes at the vertices of the triangle and has the same Hessian matrix as u .

Using a Taylor expansion at each vertex v_i of K (where the error vanishes) we can write,

$$e_K(v_i) = e_K(x) + \langle \overline{v_i \vec{x}}, \nabla(e_K)(x) \rangle + \frac{1}{2} \langle \overline{v_i \vec{x}}, H_u \overline{v_i \vec{x}} \rangle + \varepsilon_i \|\overline{v_i \vec{x}}\|^2, \forall x \in K, \quad (1)$$

Let us denote $q_{v_i}(x) = \frac{1}{2} \langle \overline{v_i \vec{x}}, H_u(x) \overline{v_i \vec{x}} \rangle$.

Then, by neglecting the terms of ε_i , we have

$$\begin{cases} \langle \overline{v_0 \vec{x}}, \nabla(e_K)(x) \rangle + q_{v_0}(x) = 0 \\ \langle \overline{v_1 \vec{x}}, \nabla(e_K)(x) \rangle + q_{v_1}(x) = 0 \\ \langle \overline{v_2 \vec{x}}, \nabla(e_K)(x) \rangle + q_{v_2}(x) = 0 \end{cases}$$

and consequently the following system

$$\begin{cases} \langle \overline{v_0 v_1}, \nabla(e_K)(x) \rangle + q_{v_0}(x) - q_{v_1}(x) = 0 \\ \langle \overline{v_0 v_2}, \nabla(e_K)(x) \rangle + q_{v_0}(x) - q_{v_2}(x) = 0 \end{cases}$$

which can be written

$$A \nabla(e_K)(x) = -F(x) \quad (2)$$

where $A = (\overline{v_0 v_1} \ \overline{v_0 v_2})^t$ and $F(x) = \begin{pmatrix} q_{v_0}(x) - q_{v_1}(x) \\ q_{v_0}(x) - q_{v_2}(x) \end{pmatrix}$.

3.2 Seminorm

The H^1 seminorm of the interpolation error on an element K is

$$|e_K|_{H^1}^2 = \int_K \|\nabla(e_K)(x)\|^2.$$

Finally, with the approximation (2) we consider

$$|e_K|_{H^1}^2 \approx \int_K \|A^{-1} F(x)\|^2. \quad (3)$$

3.3 Remark on the Numerical Computation of the Error Gradient

To compute the error gradient we need to get an estimate of the Hessian matrix on K . Integrals of the error gradient can easily be obtained via a quadrature formula.

Estimation of the Hessian The Hessian of u_h , a P_1 finite element function, can be approximated by using a recovery method. We can cite for example the generalized finite difference (a variant of Green’s formula [10, 10.4.2]), the Zienkiewicz-Zhu [18] recovery procedure, the simple linear fitting [6], the quadratic fitting [17] or the double L^2 projection

$$H = I_{L^2}(\nabla(I_{L^2}(\nabla u))) \tag{4}$$

where I_{L^2} is the L^2 projection on the P^1 finite element space [1].

We refer to [14] for a comparison of these different recovery techniques.

Moreover, the adaptation process may be controlled by modifying the eigenvalues in the spectral decomposition of H_u (see [16] for an analysis of the effect of such a control strategy on the interpolation error).

Here, to evaluate the Hessian, we use formula (4). As it has been said in [13], we have no convergence proof of this scheme but result is better with using it [1].

Interpolation scheme In order to get the value of the Hessian everywhere on the mesh, interpolation schemes are used since the Hessian is only known at each vertex of the mesh.

4 Application to Mesh Adaptation

The procedure described below minimizes the interpolation error among all triangulations having the same number of vertices.

Following [8], we introduce a functional on the mesh based on the error gradient to drive the adaptation procedure. Hence, we define

$$\mathcal{J}(\mathcal{T}) = \sum_{K \in \mathcal{T}} \int_K \|\nabla(e_K)(x)\|^2. \tag{5}$$

Using (3) we can write the expression of \mathcal{J} with the Hessian matrix of u and we consider now

$$\mathcal{J}(\mathcal{T}) = \sum_{K \in \mathcal{T}} \int_K \|A^{-1}F(x)\|^2. \tag{6}$$

The procedure consists in minimizing functional \mathcal{J} using local operators on the initial mesh.

4.1 Operators for Mesh Improvement

There are four main types of mesh improvement, topologic or geometric: edge refinement, edge swapping, vertex suppression, vertex displacement [10]. Since we want to adapt the mesh while preserving the same number of vertices, we only consider node reconnection and vertex displacement.

Edge swapping. It is a simple operation in 2D. Only two configurations are possible as an edge has only two adjacent triangles. The edge between two triangles K_1 et K_2 will be swapped if:

$$\int_{K_1} \|A^{-1}F(x)\|^2 + \int_{K_2} \|A^{-1}F(x)\|^2 > \int_{K'_1} \|A^{-1}F(x)\|^2 + \int_{K'_2} \|A^{-1}F(x)\|^2$$

where (K'_1, K'_2) is the alternate configuration.

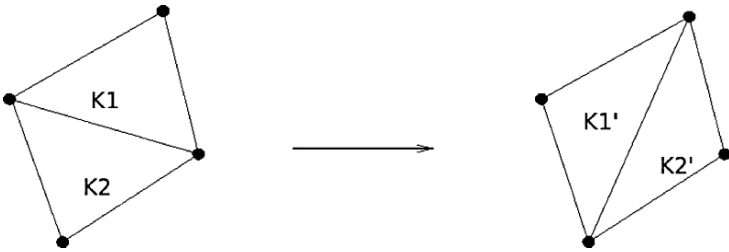


Fig. 1. swap

Vertex displacement. This procedure preserves the connectivity of the mesh. In order to get the position of the nodes that minimises the error we use a minimisation procedure.

Let $V = (v_1, v_2, \dots, v_N) \in \mathbb{R}^{2N}$ where N is the number of the vertices of \mathcal{T} .

Let us consider

$$\mathcal{F} : \mathbb{R}^{2N} \rightarrow \mathbb{R}, V \rightarrow \mathcal{F}(V) = \sum_{K \in \mathcal{T}} \int_K \|A^{-1}F(x)\|^2$$

and the following optimization problem

$$\min_{v \in \mathbb{R}^{2N}} \mathcal{F}(v),$$

for which classical optimization methods [3] can be used. We use a gradient method to minimize \mathcal{J} :

$$\forall i \in \{0, \dots, N\} \quad v_i^{n+1} = v_i^n - \rho(\nabla \mathcal{J}(V^n))_i, \quad \rho > 0.$$

The main difficulty of this procedure is to move the points without creating overlapping elements. To ensure that the movement does not destroy a valid triangulation, we adapt the step size ρ so that the new point x^* stays in the shell around x (i.e. all the elements around x).

4.2 Algorithm

The algorithm consists in applying iteratively these operators on the mesh until the mesh stabilizes.

- We start with a triangulation \mathcal{T}_h^0 of the computational domain.
- Repeat:
 - Compute the discrete Hessian at each vertex of the mesh $\mathcal{T}_h^{(i)}$
 - Generate the mesh $\mathcal{T}_h^{(i+1)}$ which minimize (5) with the following local operators:
 - Vertex displacement
 - Edge swapping
 - Terminate the loop if the number of swaps is null and if the vertices don't move.

Remarks:

Since the edge swapping is performed if and only if the functional (6) decrease, the number of swap is bounded.

5 Experimental Results

To illustrate the proposed method, we present some results for a few examples : three examples with given analytical solutions and one for a partial differential equation.

Here the goal is to generate optimal triangulation of these surfaces by using the adaptation scheme defined previously. The resulting triangulation is considered optimal in so far as it is the triangulation that minimizes the interpolation error in H^1 seminorm among all the surface triangulations (and ensure the best error equirepartition). As we want compare our procedure with classical adaptation schemes, we start with a mesh of the parameter space (we obtain the surface mesh by considering the value of the analytical function at each mesh vertex) obtained from a size map built by considering the absolute value of the Hessian matrix. More precisely the initial mesh is built by using a metric based adaptation procedure which equilibrates the interpolation error in L^∞ norm over the mesh and the metric field is computed using the absolute value of the Hessian matrix.

For each example we present the value of the functional before and after optimization. To study the convergence of the interpolation error as the mesh is refined, we plot the H^1 seminorm of the error as functions of the number of elements *nbt*. In all these examples, for technical reason, all the boundary points are fixed.

5.1 Analytical Examples

We consider an initial mesh (built with the FreeFem++ software [12]) for which the interpolation error in L^∞ norm is upperbounded by a threshold value ε .

Constant Hessian surfaces We consider here adaptive meshes for $f_1(x, y) = x^2 + y^2$ and $f_2(x, y) = x^2 - y^2$ (Fig. 2).

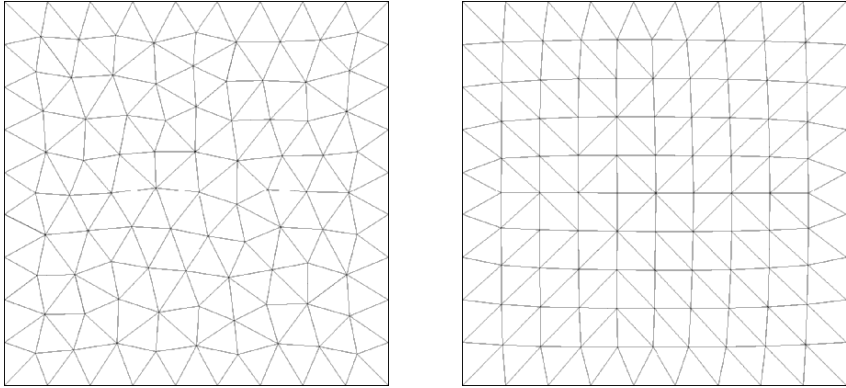


Fig. 2. Resulting meshes in the parameter space for two different types of surfaces: $f(x, y) = x^2 + y^2$. (left) and $f(x, y) = x^2 - y^2$. (center), $(x, y) \in [-1, 1] \times [-1, 1]$, where the triangles are right-angled and aligned along the axes.

Non constant Hessian surfaces We consider here adaptive meshes $f_3(x, y) = x^3 + y^3$ (Fig. 3) and $f_4(x, y) = x^2y + y^3 + \tanh(10(\sin(5y) - 2x))$ (Fig. 5) which has a more complicated structure and exhibits an highly anisotropic feature (this function simulates a solution with a shock layer : the equation of the smooth line is $\sin(5x) = 2x$)

Both procedures equilibrate the interpolation error. Nevertheless, if we compare the initial mesh to the final one, the pattern of the triangles is different between the elliptic and the hyperbolic regions. We obtain the same triangulation in the elliptic region (where the Hessian matrix is positive definite) but, as expected not in the hyperbolic region: for quadratics functions, the optimal triangles which produce the smallest H^1 seminorm of the interpolation error are acute isocelles triangles aligned with the solution [5].

We observe the same behaviour for more general functions (see the example Fig. 5).

However, the value of the functional in the case of the adaptation in L^∞ norm and in our case are close to each other for smooth functions (see Fig. 4) especially for functions whose adapted mesh has little anisotropy: in this case,

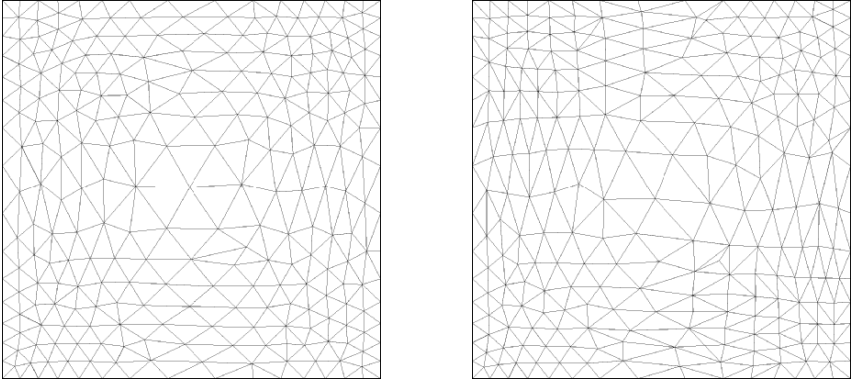


Fig. 3. Initial (left) and resulting (right) mesh in the parameter space for $f_3(x, y) = x^3 + y^3, (x, y) \in [-1, 1] \times [-1, 1]$

the classical metric based adaption technique gives good meshes even in the H^1 seminorm. This is not surprising in so far as that in this case the corresponding mesh is close to a Delaunay mesh for which roughness (and consequently the cost function \mathcal{J}) is minimal [15].

5.2 Partial Differential Equation Examples

This example solves the boundary value problem of Poisson’s equation on an L-shape with Dirichlet boundary condition:

$$-\Delta u = 1, \Omega =]0, 1[^2 -]1/2, 1[^2$$

Numerical results are shown in Table 1. Similar observations as in the previous analytical cases can be made. For a PDE with shock the key point is the reconstruction of the Hessian which can be quite difficult on highly anisotropic case [13].

Table 1. Value of the H^1 norm and L^2 norm for initial (right) and resulting mesh (left) as function of the number of elements(nbt) or vertices (nbv)

H1	L2	H1	L2	nbv	nbt
0.0409041	0.00225554	0.0426227	0.00234673	57	84
0.0129854	0.000262934	0.0136801	0.000255626	271	473
0.00906429	0.000128142	0.00944111	0.000135652	368	659
0.00633305	6.43262e-05	0.0065542	6.88933e-05	684	1264
0.00470695	3.71549e-05	0.00493037	4.07533e-05	1141	2144

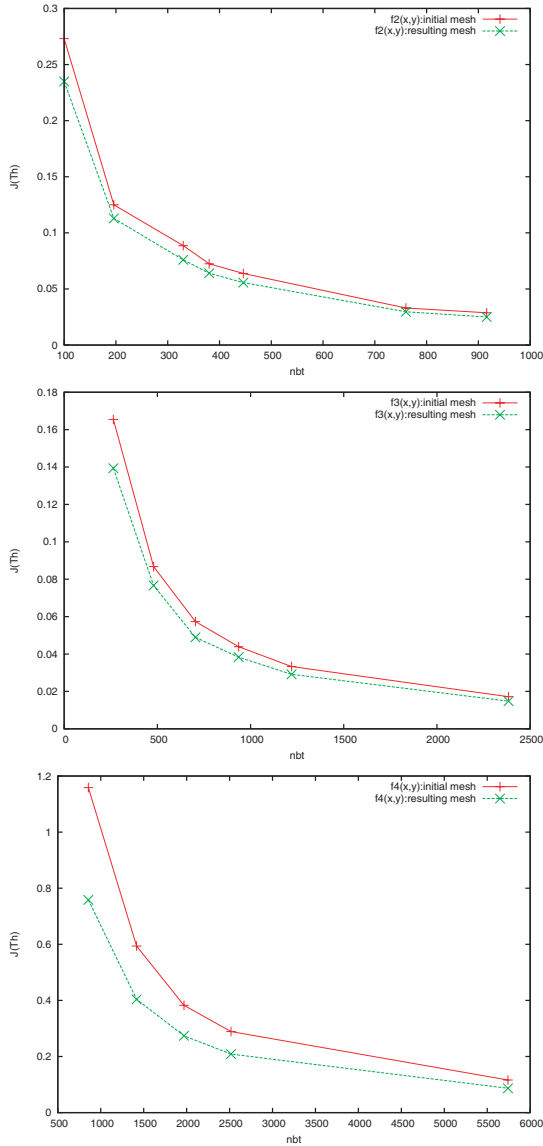


Fig. 4. Plot of $J(Th)$ as functions of nbt for $f_2(x, y)$, $f_3(x, y)$ et $f_4(x, y)$

5.3 Towards Quad-Triangles Meshes

Moreover, as for saddle-shaped functions, quadrilaterals may offer a higher-order approximation on a mesh [11], this procedure can be used for building (anisotropic) quadrilaterals in the hyperbolic region through a simple triangle-to-quad conversion [2].

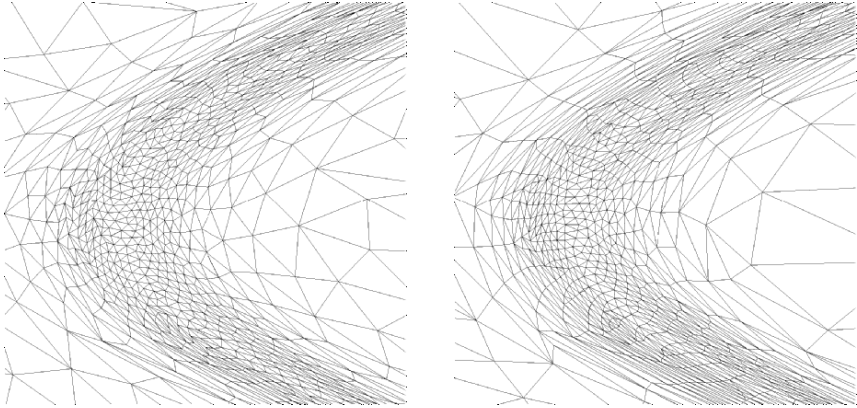


Fig. 5. Zoom of meshes of $f_4(x, y) = x^2y + y^3 + \tanh(10(\sin(5y) - 2x))$ in the parameter space (initial mesh on the left and final mesh on the right), $(x, y) \in [-1, 1] \times [-1, 1]$, $x^2 + y^2 < 1$.

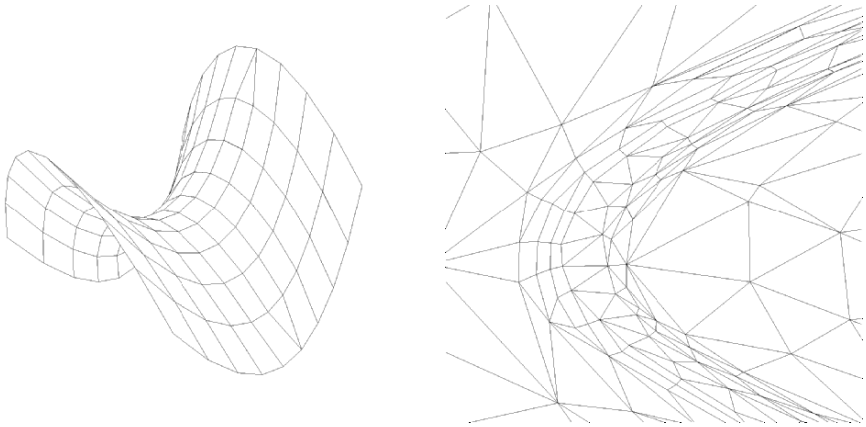


Fig. 6. Resulting surface meshes for $f_2(x, y)$ (left) and $f_4(x, y)$ (right) where triangulated mesh has been changed into a quad-dominant mesh in hyperbolic region through a simple triangle-to-quad conversion.

6 Conclusion

In this paper we have presented a simple methodology based on local mesh operators for building optimal meshes for the H^1 seminorm of the interpolation error. Numerical results were presented to illustrate our approach.

- We have shown through numerical experiments that classical methods based on the absolute value of the Hessian matrix of the solution allow the construction of good meshes even for the H^1 seminorm, especially from the finite element viewpoint.

- From a surface meshing viewpoint it would be interesting to study in detail the optimization in H^1 seminorm especially for building quad-dominant meshes in anisotropic regions.

This method can be extended to 3D. In this case the local operators for mesh modifications are much more complex : the edge swap operator modifies the shell of an edge (ie all the tetrahedra around an edge) and the number of possibly new tetrahedralizations of a shell of n elements is given by the Catalan number $C(n) = \frac{(2n-2)!}{n!(n-1)!}$ which shows the complexity of this operator.

References

1. Alauzet F., Adaptation de maillage anisotrope en trois dimensions. Application aux simulations instationnaire en Mecanique des Fluides, PhD thesis, Universit des Sciences et Techniques du Languedoc, France, 2003.
2. Borouchaki H. and Frey P.J., Adaptive triangular-quadrilateral mesh generation. *Int J Numer Meth Engng* 41:915-934, 1998.
3. Ciarlet P.G. Introduction à l'analyse numrique matricielle et à l'optimisation, *Masson*, Paris, 1982.
4. Ciarlet P.G. Basic error estimates for elliptics problems. *in Ciarlet P.G., Lions J.-L. (eds), Handbook of numerical analysis*, vol II North Holland Amsterdam 1991.
5. Cao W. On the error of linear interpolation and the orientation, aspect ratio, and internal angles of a triangles, *SIAM J. Numer. Anal.* Vol. 43, No 1, pp. 19-40.
6. Dompierre J., Labb P., Guibault F., Controlling Approximation Error, *Second M.I.T Conference on Computational Fluid and Solid Mechanics*. Cambridge, Elsevier, p. 1929-1932, 2003.
7. Fortin M. Estimation d'erreur a posteriori et adaptation de maillages, *Revue européenne des éléments finis*, 9(4), 2000
8. Fortin M. Anisotropic mesh generation, *école CEA-INRIA-EDF : A posteriori estimation and output bound computation*, 2004.
9. Frey P.J., Borouchaki H. Surface meshing using a geometric error estimator. *Int J Numer Meth Engng* 58(2):277-45, 2003.
10. Frey P.J. Georges P.-L. Mesh generation. Application to finite elements. Paris, Oxford: Hermes Sciences Publ., 2000.
11. D'Azevedo E., Are Bilinear Quadrilaterals Better Than Linear Triangles? *SIAM Journal on Scientific Computing* Vol. 22, No 1, pp. 198-217.
12. F.Hecht, K.Ohtsuka and O.Pironneau. *FreeFem++ manual* Universite Pierre et Marie Curie, <http://www.freefem.org/ff++/index.htm>
13. F.Hecht. A few snags in mesh adaptation loops. *In Proc.of 14th Int. Meshing Roundtable*, San Diego, USA, 2005
14. Manole C.-M., Vallet M.-G., Dompierre J. and GuibaultF. Benchmarking Second Order Derivative Recovery of a Piecewise Linear Scalar Field, *Proceedings of the 17th IMACS World congress Scientific Computation, Applied Mathematic and Simulation*, 2005.
15. S. Rippa, Minimal roughness property of the Delaunay triangulation, *Comput. Aided Geom Design*, 7: pp. 489-497, 1990

16. Vassilevski Yu.V., Lipnikov K.N. Opimal Triangulations: Existence, Approximation, and double differentiation of P_1 Finite Element Functions. *Comput. Math. Math. Phys.* Vol 43, No 6, 2003, pp. 827-835.
17. Zhang X. D., Accuracy concern for Hessian Metric, Internal note, CERCA, 2001.
18. Zienkiewicz O.C, Zhu J.Z. The superconvergent patch recovery and a posteriori error estimates. Part I: the recovery technique. *Int J Numer Meth Engng* 33(7):1331-1364, 1992.

Mesh Smoothing Based on Riemannian Metric Non-Conformity Minimization

Yannick Sirois¹, Julien Dompierre¹, Marie-Gabrielle Vallet¹, and François Guibault¹

École Polytechnique de Montréal
C. P. 6079, succ. Centre-Ville, Montréal (Québec), H3C 3A7, Canada
[yannick.sirois|julien.dompierre]@polymtl.ca
[marie-gabrielle.vallet|francois.guibault]@polymtl.ca

Key words: Mesh quality, mesh optimization, mesh adaptation, element size, element shape, metric comparison, boundary orthogonality, anisotropy.

Summary. A mesh smoothing method based on Riemannian metric comparison is presented in this paper. This method minimizes a cost function constructed from a measure of metric non-conformity that compares two metrics: the metric that transforms the element into a reference element and a specified Riemannian metric, that contains the target size and shape of the elements. This combination of metrics allows to cast the proposed mesh smoothing method in a very general frame, valid for any dimension and type of element. Numerical examples show that the proposed method generates high quality meshes as measured both in terms of element characteristics and also in terms of orthogonality at the boundary and overall smoothness, when compared to other known methods.

1 Introduction

In the context of numerical simulations, particularly in computational fluid dynamics (CFD), the concept of mesh quality is always an issue. Smoothing is a mesh modification method that can be used to increase mesh quality in many ways. Most often, simple smoothing algorithms are used after initial mesh generation or topological modifications to an existing mesh, in order to equidistribute variations of size or shape globally or locally, see [1, 2] for examples.

In this paper, a new mesh smoothing method based on the minimization of metric non-conformity is proposed. The presented method, instead of optimizing size or shape functions, directly compares an element's actual metric to a desired target metric. These metrics contain, in a single matrix entity,

details on local size and shape. Since the algorithm is only dependant on a specified metric, it can be used in different settings such as initial mesh generation, where the specified metric is constructed from geometric information, or in *a posteriori* adaptation, where the metric is computed from a numerical solution. Assuming that a correctly defined metric is specified, this paper explains how a mesh smoothing method can be devised to generate high quality meshes with respect to the metric, while respecting high constraints for the mesh such as constant number of vertices and constant connectivity between vertices of the mesh.

The first section of the paper presents some of the works related to mesh smoothing and discusses why a new smoothing algorithm is needed, that simultaneously accounts for both size and shape of elements. The concepts of Riemannian metrics and non-conformity are explained next, in Sect. 3. The paper then goes on to explain the smoothing method used to optimize the non-conformity of a mesh (Sect. 4) and presents the prototype algorithm used to validate the method. Numerical examples that illustrate the versatility of the method and the quality of the resulting meshes are presented in the final part of the paper, and conclusions are drawn.

2 Mesh Optimization by Smoothing

Smoothing methods can be separated into two categories: methods that optimize size distribution and methods that optimize element shape.

In the size distribution methods category, the most common type of smoothing is certainly Laplacian smoothing, where a vertex is moved to the center of its neighbors. Examples of other size distribution methods include physical analogies such as the spring analogy [3, 4] and particle potential minimization [5], methods based on the elliptical Poisson system [6, 7, 8, 9] as well as “center of mass” methods [10, 11]. These methods have been used in adaptive setting: they all have some kind of weight function or concentration function that allows for spacing or size specification. Their main drawback is that they provide very little control over element shape, since they are only based on the measure of distance between points. They are not appropriate when orthogonality or other shape properties are desired. Moreover, these methods are subject to geometric tensions. This means that vertices are attracted in concave corners, and this pulling effect can even result in the mesh folding outside the geometry, since optimal positioning of nodes is based on length and is not aware of domain boundaries. This behavior can be controlled using constraints on the optimization process to enforce boundaries, or concentration functions to reduce tensions. But these processes must often be adjusted somewhat manually for a given class of geometries. This hints to the fact that, as they are formulated, these optimization processes do not entirely incorporate the underlying engineering and computational objectives of smoothing.

The second category of methods is shape based optimization. Some of the best known methods in this category apply a complex optimization algorithm to reduce a cost function based, for example, on angular criteria [12, 13] or on shape distortion measures such as those presented in [14, 15, 16]. Most often, these smoothing methods are used as a final step during mesh generation, to regulate shape variations from an ideal shape, for example a square for a two-dimensional quadrilateral element. The resulting meshes exhibit very smooth shape distribution. The inherent limitation of these methods lies in the fact that the definition of a perfect element shape is global. When a vertex is moved, the optimization process tries to satisfy a specific shape which is the same over the whole domain. These approaches are excellent to correct unsatisfactory shape distortions in a generated mesh, but lack the capacity to adapt vertex distribution to complex flow characteristics, that locally exhibit highly anisotropic features. For this latter purpose, it is necessary to be able to locally specify the exact shape desired, including anisotropy.

From the previous analysis, it becomes apparent that current methods lack one of two kinds of control, either on size or shape. In the present work, the goal is to unify these controls into a single target specification, and devise a vertex relocation method capable of satisfying at best this specification. For example, it could be necessary, in the same mesh, to specify highly anisotropic orthogonal elements to resolve boundary flow near an airfoil while also specifying, in another region, highly anisotropic elements stretched in a specific orientation and size to resolve a shock wave. In this case, a variation of a shape based approach might seem best suited for the boundary layer part, while a size based approach would probably yield the best results for the shock region.

Two or more smoothing methods can be combined either by successively applying each one, sequentially or iteratively, or by minimizing a single cost function obtained as an arbitrary combination of several simpler functions. However, this type of combined method results in heuristic approaches that are application and case dependant and thus, not as general as desired. In the present work, a single cost function is used, rather than an arbitrary combination of functions, in order to prevent spurious properties in resulting meshes and case specific modifications to the function.

To obtain high quality meshes with local control of mesh characteristics, a number of desired properties of the smoothing method have been identified. The smoothing method should allow to:

1. simultaneously optimize both element size and shape;
2. specify variable anisotropic size and shape targets over the domain;
3. minimize a single cost function;
4. smooth both structured or unstructured meshes in 2D and in 3D;
5. construct non-folding meshes without constraining the optimization process.

The first three properties can be met using a cost function based on a Riemannian metric, as the next sections will show. Also, since a metric-based

specification of the target mesh characteristics is independent of element type, the use of a cost function based on metric comparison ensures that the optimization process be independent of the mesh and element types as well.

Furthermore, the present work aims to develop a general mesh smoothing method that naturally converges towards non-folded meshes. Hence a formulation of the smoothing problem is chosen that lends itself to unconstrained optimization. Indeed, it is postulated that for the optimization process to naturally result in high quality meshes without constraints, essentially entails that the overall process be specifying a correct form of the mesh smoothing problem. Here, a correct form of the smoothing problem refers to a formulation where element size, element shape, presence of domain boundaries and fixed mesh connectivity are intrinsically accounted for.

3 The Concept of Metric and Non-Conformity

The use of a Riemannian metric as a size and shape specification map for adaptation of a mesh is a central concept to this paper. It has been first introduced in [17] as a way to describe the size, stretching and orientation of the mesh elements in a single matrix entity. It has been shown in works such as [18, 19] to allow the control of mesh characteristics through the specification of a single tensor defined on the domain.

A specified metric \mathcal{M}_s can be constructed from *a posteriori* error estimation or user defined functions as well as geometric properties. The Riemannian metric is a general entity that can be used in any adaptation process, independent of how it is constructed and what characteristics the user wants to achieve through the adaptation process.

Smoothing using a metric involves moving mesh vertices so that each element be as close as possible to the ideal size and shape, as measured in the space defined by the specified metric. These ideal elements are the unit side equilateral triangles or the unit squares in two dimensions and their equivalents in three dimensions. Being of the ideal size and shape in the metric will result in an element being of the specified size, stretching and orientation, according to the metric.

The quality of a mesh can be measured using the non-conformity measure presented for simplices in [20, 21] and extended to non-simplices in [22]. The central idea is that the actual metric \mathcal{M}_K of an element K , the metric that defines the transformation between the element in its present state and the ideal element described before, must be equal to the specified metric:

$$\mathcal{M}_K = \mathcal{M}_s. \quad (1)$$

Two residuals can be computed from Eq. (1) and, when added, result in the following non-conformity tensor:

$$T_{nc} = \mathcal{M}_K \mathcal{M}_s^{-1} + \mathcal{M}_s \mathcal{M}_K^{-1} - 2I \quad (2)$$

where T_{nc} is the non-conformity tensor and I the identity tensor. A norm is then taken on this tensor to obtain a single non-conformity measure ε_K . There are a number of possibilities for the choice of a matrix norm, and effects of this choice still need to be determined, but the Frobenius norm is used throughout this paper:

$$\varepsilon_K = \|T_{nc}\| = \sqrt{\text{tr}(T_{nc}^T T_{nc})}. \quad (3)$$

A global non-conformity measure $\varepsilon_{\mathcal{T}}$ for a whole mesh \mathcal{T} has been defined by Labbé et al [20] as the average of the elementary values:

$$\varepsilon_{\mathcal{T}} = \frac{1}{n} \sum_i^n \varepsilon_{K_i} \quad (4)$$

where n is the number of elements in the mesh and ε_{K_i} the non-conformity measure of element i . This measure is used in Sect. 5 to compare the global quality among meshes obtained in numerical examples.

4 Non-Conformity Minimization

Smoothing a mesh using a Riemannian metric as a control function is not a recent idea. A review of some tested methods can be found in [23]. In these earlier methods, target size and shape are usually achieved by the use of optimization algorithms based on geometric properties measured using the distance between vertices. The distance $L_{\mathcal{M}_s}$ between vertices A and B is measured in the metric space using the vertices known coordinates in geometric space:

$$L_{\mathcal{M}_s} = \left(\overline{AB}^T \mathcal{M}_s \overline{AB} \right)^{1/2}. \quad (5)$$

Since all these methods use Eq. (5) in the definition of the weights or the cost function, it means that they are length based. As discussed in Sect. 2, it is inherent to these methods that they can result in regions of undesired element concentration, even folding, in highly curved regions or in areas with strong variations in the specified metric. The smoothing algorithm proposed here prevents these effects by optimizing a function based on an element based metric comparison instead of using a measure of edge lengths.

4.1 Choice of the Cost Function

Several cost functions can be constructed based on the elementary non-conformity measure and the choice discussed here is independent of the resolution algorithm presented in Sect. 4.2. For our tests, the cost function f chosen for optimization is the sum of the squared non-conformity measures of the elements, as follow:

$$f = \sum_{K_i \in \mathcal{T}} \varepsilon_{K_i}^2 \quad (6)$$

Knowing that the non-conformity measure vanishes for a perfect element and increases exponentially as the element is modified from its perfect state, this function will put more emphasis on bad elements than an algebraic average. The results obtained using this cost function have been compared to those obtained using the simple summation of the non-conformity measure, i.e. taking $\varepsilon_{\mathcal{T}}$ as the cost function. In general, Eq. (6) generates better results in regions where the non-conformity measure is quite similar over neighboring elements, as it increases the distance between the values of the elements non-conformity measures. Minimization of this function prioritizes reducing distortions of the worse elements.

The optimization algorithm described in the next section uses Gauss-Seidel iterations on the mesh vertices. Assuming a fixed position for all vertices except one, the optimal position for this vertex V is the position where the cost function f is minimized. Moving of vertex V only affects its neighborhood $N(V)$ composed of elements K_i that own the vertex V . The optimal position (x_V, y_V) for vertex V is the one minimizing the contribution of the neighborhood $N(V)$ to the cost function f

$$f_V = \sum_{K_i \in N(V)} \varepsilon_{K_i}^2. \tag{7}$$

Figure 1 shows examples of two-dimensional neighborhoods that will be used for the calculations presented next.

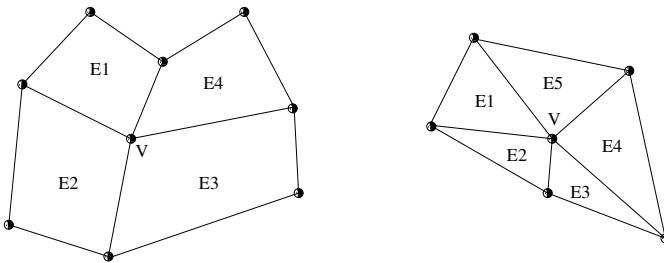


Fig. 1. Neighborhood N of a vertex V

4.2 Minimization Algorithm

In order to minimize a cost function based on the non-conformity measure, the algorithm presented by Seveno in [24] has been chosen. This algorithm can be rated as a “brute force” method, since it is not based on standard optimization methods, using, for example, computation of gradients. The optimal position of a vertex is determined by sampling the cost function at various positions around the current vertex position and choosing the one with the lowest value

of the cost function. This sampling is performed a number of times for each vertex, using a dynamic step size in each direction until convergence. The global algorithm is a Gauss-Seidel scheme where each vertex is moved in an arbitrary order, generally being the one in which vertices were listed in the mesh input file, and the new computed position is updated right away to influence the displacement of the next vertices. The algorithm is presented for the two-dimensional case for simplicity, but its three-dimensional version is a direct extension.

Algorithm 1 describes the global smoothing procedure. The maximum number of iterations is set by the user in order for the code to stop if convergence cannot be obtained. During each global iteration, the algorithm tries to displace every vertex. The procedure is stopped completely and said to have converged if the maximum displacement of all vertices is lower than a user specified value usually around 0.1% of the average edge length of the neighborhood. This criterion is very restrictive but is necessary in cases where some vertices need to travel a great distance in the domain. The mesh could be considered converged by a more relaxed criterion such as the average displacement, but it could happen that some vertices do not completely travel to their optimum positions.

Algorithm 1 Global Procedure

```

for iter = 1 to maximum number of iterations do
  for vertex = 1 to last vertex do
    Move vertex using Algorithm 2
    Compute vertex total displacement
  end for
  Compute maximum vertex displacement
  if Maximum vertex displacement < threshold value then
    End the global iteration cycle
  end if
end for

```

The displacement algorithm differs from the one proposed by Seveno's only in the evaluation of a different cost function and in the use of a local definition of the displacement steps based on an averaged local element size. The general idea of the method is described in pseudo-code in Algorithm 2 for the displacement of a single vertex inside one global Gauss-Seidel iteration over all vertices.

After computing the cost function for the initial position of the vertex, the algorithm tries to move the vertex to eight different positions around its current one. It is moved by a distance δ in each direction. This δ starts at a user defined value $\delta_{initial}$ and is dynamically updated at each iteration of the **while** loop. After testing the eight positions, the vertex is moved to the one that reduces the cost function the most and the value of δ is increased

Algorithm 2 Optimization of a vertex position

```

 $\delta = \delta_{initial}$ 
 $CF_{initial} = \text{compute cost function for current position}$ 
 $CF_{best} = CF_{initial}$ 
while  $\delta_{min} \leq \delta \leq \delta_{max}$  do
  for  $i = -1$  to  $1$  do
    for  $j = -1$  to  $1$  do
      Test position =  $(x + i\delta, y + j\delta)$ 
       $CF_{test} = \text{compute cost function for test position}$ 
      if  $CF_{test} < CF_{best}$  then
         $CF_{best} = CF_{test}$ 
         $i_{best} = i$ 
         $j_{best} = j$ 
      end if
    end for
  end for
  if  $CF_{best} < CF_{initial}$  then
    Move vertex to  $(x + i_{best}\delta, y + j_{best}\delta)$ 
     $\delta = c\delta$ 
  else
     $\delta = \delta/c$ 
  end if
end while

```

by a coefficient c to anticipate further movement. If none of the eight tested positions reduces the cost function, the vertex stays at its initial position and the value of δ is reduced by the factor c . The reduction of δ ensures that points closer to the initial position will be tested, to see whether the optimum position lies between the actual position and the ones previously tested. For all examples presented in Sect. 5, the following values have been used:

$$\delta_{initial} = 0.1L \quad (8)$$

$$\delta_{min} = 0.01L \quad (9)$$

$$\delta_{max} = 0.5L \quad (10)$$

$$c = 5 \quad (11)$$

where L is the average edge length in the neighborhood $N(V)$.

In order to move vertices on the domain boundaries, a similar algorithm is used. In this case, the algorithm relies on the parametric definitions of the geometric entities that support the boundary edges and faces of the computational domain. In these cases, the variable δ is defined in terms of the parametrization of the underlying entity and the displacement is computed from the parameter value at which the vertex was initially located. Then, the positions tested are evaluated on the entity using a modified value of this parameter.

This algorithm yields very good results, as shown in Sect. 5. The main disadvantage of using this “brute force” approach is that it is slow. The testing of so many positions has a serious impact on the computing performance. This is caused by the fact that every time a position is tested, the non-conformity measure must be evaluated on all the neighboring elements. In order to compute these non-conformity measures, an average specified metric \mathcal{M}_s must be evaluated on each element using numerical integration as discussed in [22]. This operation is very costly because each integration point must be localized and interpolated on the background mesh, which is usually the mesh before smoothing, onto which the specified Riemannian metric field \mathcal{M}_s as been defined.

The next section shows that, at convergence, this algorithm generates meshes of higher metric conformity than other previously used methods, such as the spring analogy or edge length equidistribution.

5 Numerical Examples

This section presents a series of academic test cases for which minimization of the mesh non-conformity measure using the proposed algorithm yields high quality meshes when compared to previously published smoothing methods. The main assumption regarding these tests is that the specified Riemannian metric used to control the smoothing process can be computed correctly and adequately represents a user’s needs.

5.1 Optimization of Shape

The first example presented is really simple. It is a rectangle geometry with a structured 4×4 quadrilateral element mesh. The purpose of this example is to show how sensitive to shape distortion the proposed algorithm is, compared to edge length equidistribution. Obviously, the smoothest mesh, without using concentrations, on this geometry is the one where all quadrilaterals are perfect rectangles that fit the geometry.

Figure 2 shows the results of smoothing the mesh on this geometry using edge length equidistribution in the image at left and minimization of non-conformity measure in the image at right using a uniform Euclidean specified metric. Smoothing using the first method generates a valid mesh. The elements are roughly the same size and vertical edges were lengthened so that edge lengths become more uniform. However, shape is not optimized at all. The color scale shows the resulting non-conformity measure of each element. The global non-conformity measure for this mesh is $\varepsilon_{\mathcal{T}} = 102.061$ and the elementary non-conformity measure ε_K varies from 41 to 213. By comparison, the second mesh is much better. Its non-conformity measure is $\varepsilon_{\mathcal{T}} = 56.966$ and is the same for each element since the specified metric is uniform. While not perfect, one must keep in mind that the reference element is a unit square, not a rectangle. But it can be seen that not only size was optimized but the

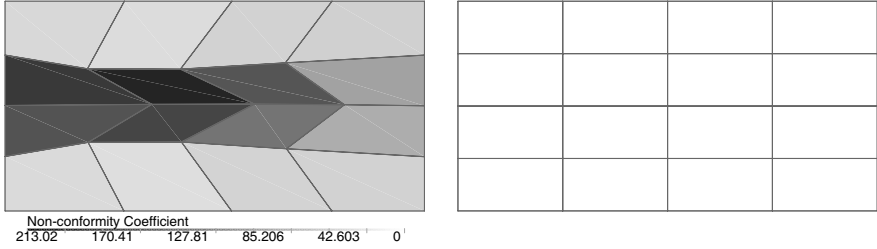


Fig. 2. Edge length equidistribution versus minimization of non-conformity measure in a rectangular domain

shape was also respected, yielding elements closer to the unit square than in the other mesh.

5.2 Smoothing in Curved Regions

One of the main concerns while smoothing meshes is the introduction of geometric tension, as described in Sect. 2. In this section, the example uses a simple curved geometry in order to show that, compared to other smoothing methods, effect of geometric curvature on final vertex position is minimal for metric non-conformity based smoothing.

In this example, the computational domain is a two dimensional duct with a 180° elbow, as shown in Fig. 3. The initial mesh used for the smoothing procedure, also shown in Fig. 3, is a standard transfinite interpolation. If this mesh were to be smoothed using widely used methods such as spring analogy or standard elliptic smoothing, the results would not be satisfactory because of geometric tensions in the curved regions. Images of the resulting meshes can be found in [25].

For methods based on length only, such as the spring analogy, geometric pulling on the vertices is not negligible. Since there is absolutely no control on the shape of elements, vertices can be pulled outside the geometry, in between the two arms of the duct, to reach the optimum positioning, as shown in [25]. In the case of the elliptic smoothing, it is known that vertices are naturally concentrated in the concave region while moving away from the outer boundary. It is not necessary to mention, that even if the resulting mesh is guaranteed to be valid, this type of concentration is not a desired effect.

These geometric tensions are very hard, if not impossible, to get rid of. One of the solutions to reduce their effect on vertex positioning is to include a criteria measuring shape in the optimization process and this is what is naturally accomplished through metric non-conformity minimization. Figure 4 shows the result of the smoothing of the transfinite interpolation mesh of the previous figure.

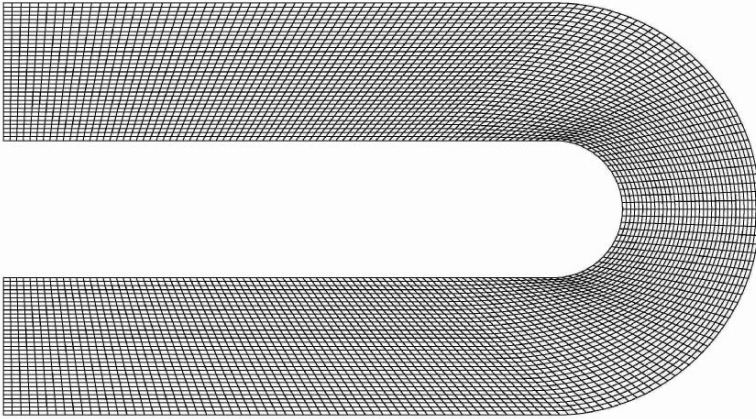


Fig. 3. U-duct with a 199×35 transfinite interpolation mesh

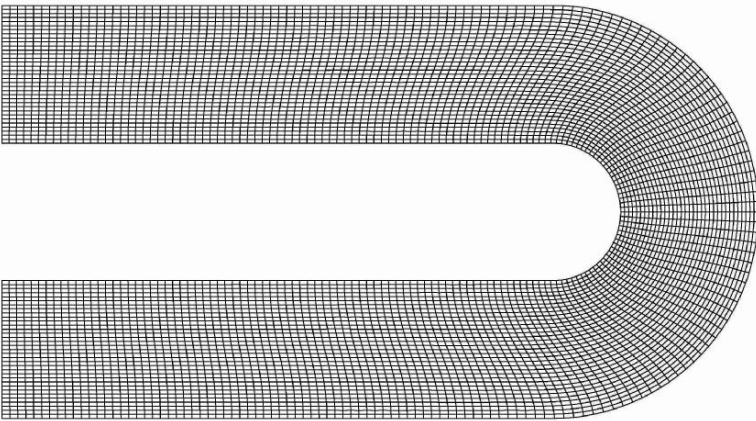


Fig. 4. Mesh smoothed using non-conformity minimization and Euclidean metric

In order to get this mesh, a locally uniform Euclidean target metric was specified. This metric is isotropic and the main diagonal is uniform and defined by the local size around each vertex. The smoothing process took 8.87 hours on an *Athlon 2400+* processor to converge in 563 global Gauss-Seidel iterations. The resulting mesh presents almost no sign of undesired concentrations in the curved region. The mesh is really smooth in all directions and everywhere enclosed in the prescribed geometry. Compared to the initial mesh that had a non-conformity measure of $\varepsilon_{\mathcal{T}} = 5.10$, this mesh reduced considerably this measure to $\varepsilon_{\mathcal{T}} = 3.61$ which is a good quality improvement considering the fixed mesh connectivity.

One can also notice the orthogonality of the elements at the boundaries. This mesh characteristic is very useful in CFD for example, but is not inherent to methods such as transfinite interpolation. Orthogonality must usually be imposed through special conditions in optimization functions near the boundary. In the case of non-conformity minimization, it is implied in the specification of the reference element. When a boundary vertex needs to be moved, it is almost certain that it cannot reach the desired element size, being restricted to sliding on the boundary. In that case, the cost function tends to minimize shape discrepancy between the reference element and the actual element to reach the lowest cost function value it can. Since the desired element is a square, the vertex will move to the position where its two adjacent elements become closest to squares, generating orthogonal elements.

5.3 Smoothing Using *a posteriori* Error Estimator

The next example shows how the method reacts to a more complex specified metric. A CFD solution is computed on the same geometry. The mesh used for this computation is shown in Fig. 5. Knowing that there will be boundary layers present, the mesh has been concentrated near the boundary beforehand. Nothing else has been done to the mesh in order to not anticipate any other flow characteristics.

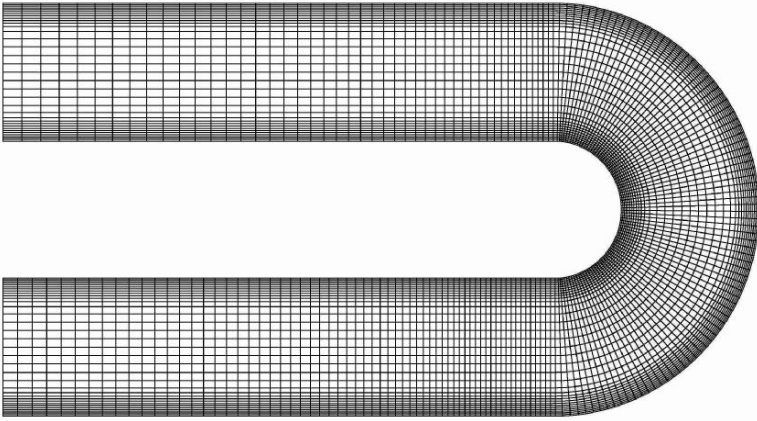


Fig. 5. Mesh provided to the CFD solver and also initial mesh used in the smoothing algorithm.

Figure 6 shows the flow speed solution obtained on the previous mesh. Distinct flow characteristics such as the boundary layers and the detachment of this layer after the elbow are noticeable in the solution. The boundary layer detachment was not anticipated and should be better resolved with

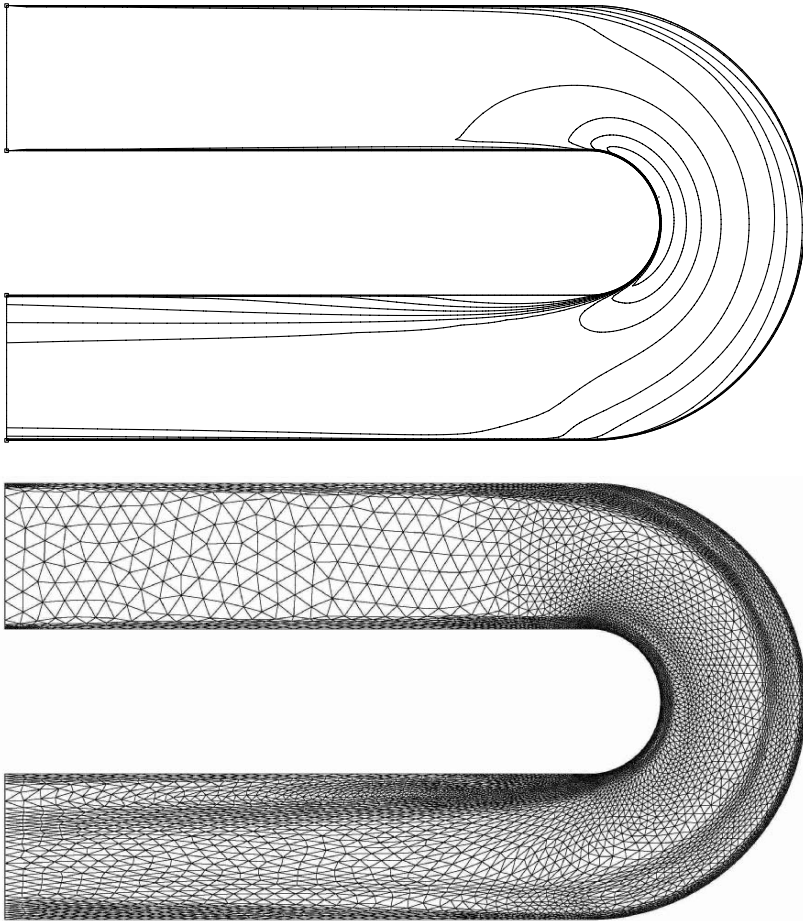


Fig. 6. A) Isovalues of speed computed on the initial mesh. B) Mesh representation of the metric computed from the CFD solution

an adapted mesh. The metric is constructed using error estimation on this solution, resulting in an anisotropic metric field and the initial mesh has a non-conformity measure of $\varepsilon_{\mathcal{T}} = 21.10$.

In order to adapt on this solution, the specified metric is computed using an error estimator based on the second derivative of the speed solution shown in Fig. 6. This estimator creates an anisotropic Riemannian metric field defined for each vertex position in the domain that gives the desired size, stretching and orientation. A representation of this specified metric is given in the form of an adapted mesh in Fig. 6 that satisfies almost perfectly the metric. Thus, a high quality smoothed mesh should resemble at best the mesh of Fig. 6 in terms of the size distribution and element shape and orientation, under the

constraint of a fixed connectivity. Here, the mesh of Fig. 6 is meant only as a visual representation of the specified metric and cannot be compared to the other meshes shown, that have a fixed topology throughout the smoothing process.

The result of smoothing the mesh of Fig. 5 using non-conformity minimization with the *a posteriori* metric field yields the mesh shown in Fig. 7, that has a non-conformity measure of $\varepsilon_{\mathcal{T}} = 7.81$. This mesh is well adapted to all flow characteristics, including the detachment of the boundary layer which can easily be seen. Again, for the same reasons as in the previous example, orthogonality at the boundaries is preserved, which is good considering the complexity of the metric specified. This example converged in 18.06 hours and 659 global iterations

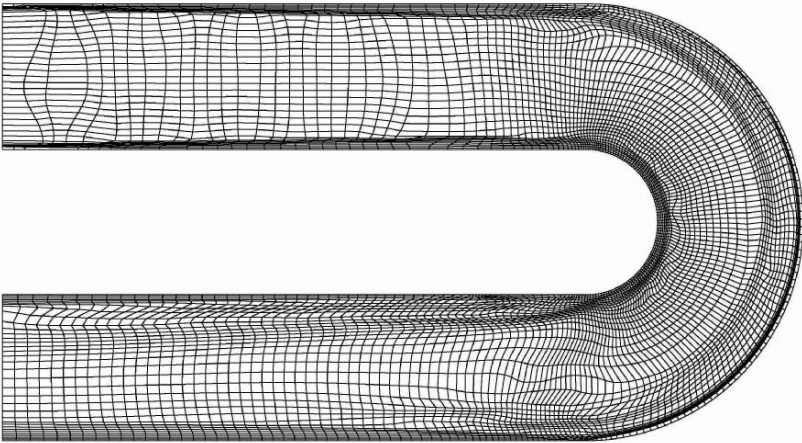


Fig. 7. Mesh smoothed using metric constructed from solution

This result can be compared to those obtained using pure spring analogy, shown in Fig. 8. This later mesh has an average non-conformity measure of $\varepsilon_{\mathcal{T}} = 16.79$ compared to 7.81 for the mesh of Fig. 7. It can be seen that compared to non-conformity minimization, it tends to concentrate elements in the elbow almost to the point of folding outside the geometry. It can also be seen in the figure that the distribution of elements is not as precise as with the previous mesh if it is compared to the representation of the specified metric of Fig. 6. Near the entrance for example, elements are too stretched along the outer wall of the duct compared to inner wall elements, even though they should be almost the same size. Moreover, orthogonality near boundaries is not preserved when using spring analogy compared to non-conformity minimization.

The same type of adaptation can be applied to a mesh with triangular elements. Fig. 9 shows the result of this adaptation using the same metric

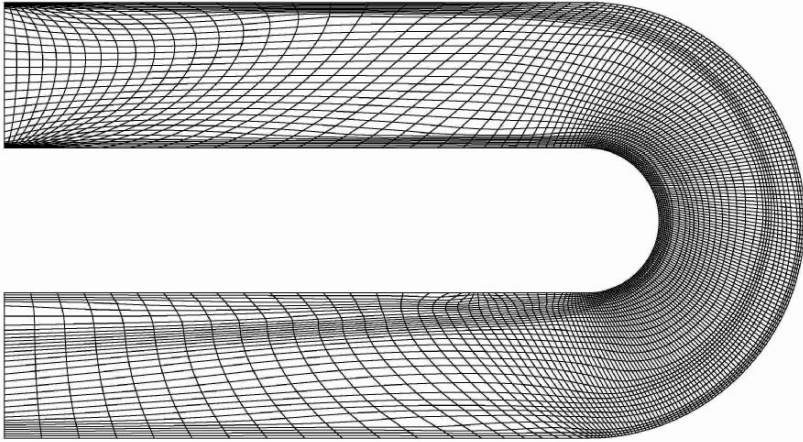


Fig. 8. Mesh adapted using pure spring analogy

reaching non-conformity measure $\varepsilon_{\mathcal{T}} = 2.53$. In this case, the original mesh is the mesh of Fig. 7 where all elements were divided into two triangles. Again, the result shows a mesh that exhibits the same characteristics as the representation of the metric in Sect. 6. This shows that a cost function based on a metric is really independent of the type of element in the mesh and that the specified metric describes the same objective for all. This example reaches convergence in 276 global iterations that took 3.46 hours. Notice that the time is considerably shorter here because there is no need to use corner sub-simplices on triangles and that the initial mesh was already adapted in its quadrilateral form.

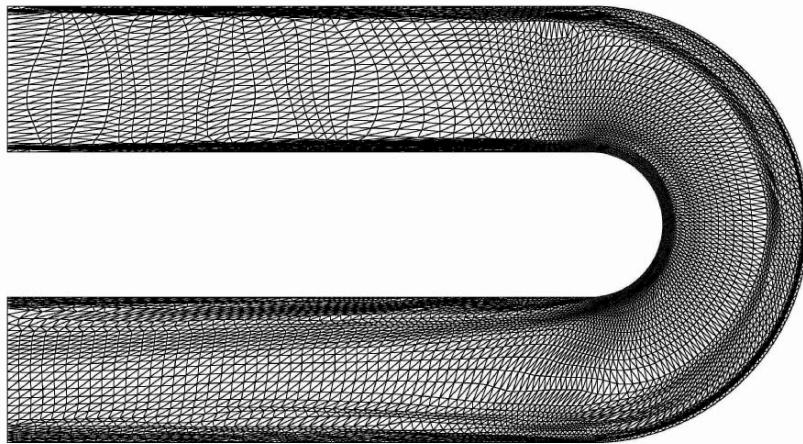


Fig. 9. Triangle mesh adapted using metric constructed from the CFD solution

At some points in both cases, mainly in the elbow, right after the boundary layer section and in the entrance section, some elements seem quite uneven and irregular, forming an odd wavy pattern. What needs to be determined here, is if it is the smoothing process or the metric itself that caused this.

In order to determine this, the mesh can be compared again to the one of Fig. 6. The mesh of Fig. 6 clearly shows, near the elbow a rapid variation from the smaller elements of the boundary layer to larger elements and then back to smaller ones. It can also be seen that the metric requires similar size and shape elements in the entrance section. This confirms that it is a feature of the specified metric. The answer to this metric from the non-conformity minimization algorithm is to try to distribute the limited number of vertices according to the specified variations, and since there are too few vertices to completely resolve the features in the metric, the result is a wavy pattern that tries to satisfy the shape specification.

This example shows that the presented algorithm is also responsive to anisotropic specification of the target size and shape while preserving general orthogonality at the boundaries. Also, it is important to notice that the resulting meshes are of higher quality than those obtained with other previously tested smoothing algorithms [25]. The times showed here are very high, this is simply because the algorithm has not been optimized for speed at the moment. It uses the restrictive criteria proposed in Sect. 4. Many modifications can be applied to the algorithm in order to increase its speed. For example, even close to convergence, every vertex is still considered and tested for displacement. Choosing to move only vertices in regions where vertices moved in previous iterations would reduce processing time in a significant way.

6 Conclusion

In the context of mesh adaptation, it is necessary to have a smoothing method that is able to smooth any type of mesh elements in terms of both size and shape and that allows to locally specify these characteristics in an anisotropic way. In this paper, a smoothing method based on the minimization of a previously defined non-conformity measure has been presented to answer this problem. The strong point of this method resides in the definition of a cost function based on metric comparison, metrics that contain in a single matrix entity, information of size, stretching and orientation desired at each specific location. The construction of this single cost function also ensures that the minimization is independent of the type of mesh elements.

Numerical examples have shown that smoothed meshes obtained using the algorithm presented are of high quality and inherently present desired characteristics such as boundary orthogonality. Also, we have seen that if the choice of displacement step is small enough, the non-conformity measure will prevent elements from going into a degenerate state, preventing folding. Compared to other methods, it does not induce geometric tensions as for length based methods and provide a way to specify shape as well as size.

The optimization algorithm used to prototype the approach, which is based on a brute force approach, is of course not the most elegant way of minimizing a cost function. While most other optimization methods use some type of derivative approach to minimize the cost function, of the cost function and since the cost function based on metric comparison is not easily differentiable, the introduction of a better optimization algorithm, although essential in practice, is left as future work.

Acknowledgments

The authors would like to thank the National Science and Engineering Research Council of Canada (NSERC) for its support of this research, under project CRDPJ 323749-05. They would also like to acknowledge the use of VU (<http://www.invisu.ca>), the package used to draw all the images.

References

1. S. R. Mathur and N. K. Madavan, "Solution-adaptive structured-unstructured grid method for unsteady turbomachinery analysis, Part I: Methodology," *Journal of Propulsion and Power*, vol. 10, pp. 576–584, July and August 1994.
2. M. K. Patel, K. A. Pericleous, and S. Baldwin, "The development of a structured mesh grid adaption technique for resolving shock discontinuities in upwind Navier-Stokes codes," *International Journal for Numerical Methods in Fluids*, vol. 20, pp. 1179–1197, 1995.
3. D. Aït-Ali-Yahia, G. Baruzzi, W. G. Habashi, M. Fortin, J. Dompierre, and M.-G. Vallet, "Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. Part II: Structured grids," *International Journal for Numerical Methods in Fluids*, vol. 39, pp. 657–673, July 2002.
4. A. Tam, D. Aït-Ali-Yahia, M. P. Robichaud, M. Moore, V. Kozel, and W. G. Habashi, "Anisotropic mesh adaptation for 3D flows on structured and unstructured grids," *Computer Methods in Applied Mechanics and Engineering*, vol. 189, pp. 1205–1230, May 2000.
5. F. Bossen, "Anisotropic mesh generation with particles," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, May 1996. CMU-CS-96-134.
6. J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin, *Numerical Grid Generation, Foundations and Applications*. New York: North-Holland, 1985.
7. P. R. Eiseman, "Alternating direction adaptive grid generation," *American Institute of Aeronautics and Astronautics Journal*, p. 1937, 1983.
8. S. P. Spekreijse, "Elliptic grid generation based on Laplace equations and algebraic transformations," *Journal of Computational Physics*, vol. 118, pp. 38–61, Apr. 1995.
9. B. K. Soni, R. Koomullil, D. S. Thompson, and H. Thornburg, "Solution adaptive grid strategies based on point redistribution," *Computer Methods in Applied Mechanics and Engineering*, vol. 189, pp. 1183–1204, 2000.
10. P. R. Eiseman, "Adaptive grid generation," *Computer Methods in Applied Mechanics and Engineering*, vol. 64(1–3), pp. 321–376, 1987.

11. S. McRae, “*r*-refinement grid adaptation algorithms and issues,” *Computer Methods in Applied Mechanics and Engineering*, vol. 189, pp. 1161–1182, 2000.
12. L. A. Freitag and C. Ollivier-Gooch, “Tetrahedral mesh improvement using swapping and smoothing,” *International Journal for Numerical Methods in Engineering*, vol. 40, pp. 3979–4002, 1997.
13. L. A. Freitag, “On combining Laplacian and optimization-based mesh smoothing techniques,” *AMD Trends in Unstructured Mesh Generation, ASME*, vol. 220, pp. 37–43, 1997.
14. A. Oddy, J. Goldak, M. McDill, and M. Bibby, “A distortion metric for isoparametric finite elements,” *Transactions of the CSME*, vol. 12, no. 4, pp. 213–218, 1988.
15. S. A. Cannan, J. R. Tristano, and M. L. Staten, “An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes,” in *7th International Meshing Roundtable*, (Detroit, Michigan, USA), pp. 479–494, 1998.
16. P. M. Knupp, “Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II – a framework for volume mesh optimization and the condition number of the Jacobian matrix,” *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 1165–1185, 2000.
17. M.-G. Vallet, *Génération de maillages éléments finis anisotropes et adaptatifs*. PhD thesis, Université Pierre et Marie Curie, Paris VI, France, 1992.
18. P.-L. George and H. Borouchaki, *Delaunay Triangulation and Meshing. Applications to Finite Elements*. Paris: Hermès, 1998.
19. P. J. Frey and P.-L. George, *Mesh Generation. Application to Finite Elements*. Paris: Hermès, 2000.
20. P. Labbé, J. Dompierre, M.-G. Vallet, F. Guibault, and J.-Y. Trépanier, “A universal measure of the conformity of a mesh with respect to an anisotropic metric field,” *International Journal for Numerical Methods in Engineering*, vol. 61, pp. 2675–2695, 2004.
21. C. L. Bottasso, “Anisotropic mesh adaption by metric-driven optimization,” *International Journal for Numerical Methods in Engineering*, vol. 60, pp. 597–639, May 2004.
22. Y. Sirois, J. Dompierre, M.-G. Vallet, and F. Guibault, “Measuring the conformity of non-simplicial elements to an anisotropic metric field,” *International Journal for Numerical Methods in Engineering*, vol. 64, no. 14, pp. 1944–1958, 2005.
23. Y. Sirois, J. Dompierre, M.-G. Vallet, P. Labbé, and F. Guibault, “Progress on vertex relocation schemes for structured grids in a metric space,” in *8th International Conference on Numerical Grid Generation*, (Honolulu, USA), pp. 389–398, June 2002.
24. É. Seveno, *Génération automatique de maillages tridimensionnels isotropes par une méthode frontale*. PhD thesis, Université Pierre et Marie Curie, Paris VI, Mar. 1998.
25. Y. Sirois, J. Dompierre, M.-G. Vallet, and F. Guibault, “Using a Riemannian metric as a control function for generalized elliptic smoothing,” in *Proceedings of the 9th International Conference on Numerical Grid Generation in Computational Field Simulations*, (San Jose, CA), June 2005.

On Asymptotically Optimal Meshes by Coordinate Transformation

Guillermo D. Cañas and Steven J. Gortler

Harvard University
{gdiez|sjg}@cs.harvard.edu

Summary. We study the problem of constructing asymptotically optimal meshes with respect to the gradient error of a given input function. We provide simpler proofs of previously known results and show constructively that a closed-form solution exists for them. We show how the transformational method for obtaining meshes, as is, cannot produce asymptotically optimal meshes for general inputs. We also discuss possible variations of the problem definition that may allow for some forms of optimality to be proved.

1 Introduction

In this paper, we study “optimal triangular meshes for minimizing the gradient error”, as first described in the landmark paper by D’Azevedo and Simpson in [2].

In particular they consider the problem of approximating an input bivariate scalar function $f(x_1, x_2)$ by a piece-wise linear triangle mesh that interpolates f at its vertices. They are interested in approximations whose gradient errors are less than an input ϵ in an L^∞ sense, and are interested in the limit behavior of the meshing method as $\epsilon \rightarrow 0$.

D’Azevedo and Simpson [2] described the following simple approach. Given f ,

- Find a reparametrization of f (described by a mapping from the plane to the plane) that is “isotropic” in gradient error.
- Lay down a regular triangulation in the reparametrized domain.
- Map this triangulation back to the original (x_1, x_2) domain (straightening out any edges that have become curves).
- Sample f at the vertices of this triangulation to produce the piece-wise linear approximation.

D’Azevedo and Simpson then show the following

- They prove that a reparametrization that equidistributes gradient error exists, and can be calculated as the solution to a specific two-dimensional differential equation.
- They prove that if f is quadratic, then their method has at most 30% more triangles than any other triangulation that has L^∞ gradient error less than ϵ .
- They argue that the same 30% bound should hold *in the limit* for arbitrary f over any region where the determinant of H , the Hessian of f , is bounded away from 0.

This method is very appealing because it creates an approximation using a triangulation that has completely regular connectivity, a property that can be very useful for finite element applications. Moreover, the triangulation is the result of an explicit reparametrization step and so no optimization is required. Of course the reparametrization does require knowledge of f , so this may present somewhat of a bootstrapping problem, but in practice f may be approximated. Additionally, similar to some other techniques [2, 3, 6], this method only applies in regions where the determinant of H is bounded away from 0, and so for more complicated functions it may need to be used in conjunction with other global decomposition methods.

In this paper, we present the following new results

- We show that the isotropic reparametrization can be expressed in closed form simply as the gradient of f . No differential equation solving is necessary.
- We show that, in general, for a non-quadratic f the argument from [2] is not complete, and in fact, when there are no bounds on the anisotropy of H , then the amount of gradient error can be unbounded.
- We follow this with a discussion of some ideas we have explored to address the limitations of the basic algorithm.

Our closed form expression for the reparametrization greatly simplifies the method of [2]. It removes the need to solve a complicated differential equation (compare [2] equation 3.3). It also makes the existence proof of the reparametrization almost trivial (compare [2] section 7).

Unfortunately, we also show that for arbitrary f , the gradient error can be unbounded. Informally speaking, a right isosceles triangle in the equilibrating reparametrization will automatically have the “correct” aspect ratio as determined by the eigenvalues of H , and will be “correctly” stretched in the directions corresponding to the eigenvectors of H . But if the edges are not additionally aligned with the eigenvector directions, then the resulting triangulation can have large angles, arbitrarily close to 180 degrees.

For a quadratic function f , the eigenvector directions are spatially invariant, and so one can always rotate the domain to align its axes everywhere with the eigenvectors. For a non-quadratic function, this is generally not possible. These resulting triangles can thus exhibit arbitrary gradient error.

Negative results of this kind have appeared elsewhere. Reference [5] shows that it is, in general, not possible to obtain meshes through the transformational method that satisfy the orientation and equidistribution property for any adaptation function that specifies the shape and orientation of elements at every point. Although ours is somewhat of a negative result, we hope that the readers will value the clarification of the strengths and weaknesses of the method described in [2].

2 Reparametrization

Consider the problem of approximating an input scalar function $f(x_1, x_2)$ defined over the plane with a piece-wise linear triangle mesh. The mesh exactly interpolates \mathbf{f} at the vertices and is linear inside triangles. Such a mesh defines a continuous piece-wise linear approximation of \mathbf{f} . We define the point-wise gradient error at each (x_1, x_2) to be the squared L^2 norm of the difference between the gradient of \mathbf{f} and the gradient of the mesh approximation. Because the mesh represents a piece-wise linear function, it's gradient is piece-wise constant, and can be evaluated only in the interior of triangles. This definition of gradient error follows [2].

Define the gradient error of the triangulation to be the maximum point-wise gradient error over the entire domain.

At every point in the plane, define the Hessian matrix of f with respect to the coordinates x_i , with entries

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

As done in [2], we restrict ourselves to a domain in the plane where H is non-degenerate (determinant bounded away from 0).

At each point, define the (positive definite) squared Hessian matrix $Q = H^T H$. As described in [2], this matrix is related to the gradient error for very small triangles. If, as in [2], we assume that for every small triangle, there is an interior (or very close) point x_0 where the triangle's gradient exactly matches the gradient of f , then the gradient error at any other point x interior to the triangle is approximately $H(x - x_0) = Hr$, where H is the Hessian matrix of the Taylor expansion of \mathbf{f} around x_0 , and $r = x - x_0$ is a very small displacement. The squared L^2 gradient error at a point is thus $r^T H^T H r = r^T Q r$ (plus higher order terms).

Under the transformational method it is our goal to find a reparametrization of the plane such that Q becomes the identity matrix.

In [2] it is shown that such an *isotropic* transformation always exists for any smooth \mathbf{f} , and they provide a differential equation that can be numerically integrated to find the solution. As discussed in [2], finding such a reparametrization has implications in finding triangulations with low gradient error.

2.1 A Closed Form Reparametrization

Here we present a simple closed form for a reparametrization of the plane such that Q becomes the identity matrix.

We interpret the matrix $Q = H^T H$ of section 2 as the coordinates of the $(0, 2)$ error tensor

$$q_{ab} = \sum_{ij} Q_{ij} (dx^i)_a (dx^j)_b$$

where the a and b subscripts are coordinate-free tensor place holders and the $(dx^i)_a$ make up the basis of covectors at each point [1].

Assume that we have reparametrized the plane using new coordinates

$$[\hat{x}_1(x_1, x_2), \hat{x}_2(x_1, x_2)]$$

We can then re-express the original tensor q as

$$q_{ab} = \sum_{ij} \hat{Q}_{ij} (d\hat{x}^i)_a (d\hat{x}^j)_b$$

with some new appropriate matrix \hat{Q} . Define the Jacobian matrix of the reparametrization at each point as

$$J_{ij} = \frac{\partial \hat{x}_i}{\partial x_j}$$

then, following the well known “basis change rule” [1], we have

$$\hat{Q} = J^{-T} Q J^{-1} \tag{1}$$

Note that this matrix \hat{Q} is *not* the square of a matrix with entries $\frac{\partial^2 f}{\partial \hat{x}_i \partial \hat{x}_j}$. Rather it represents the coordinates of the original tensor re-expressed in the new parametrization.

Define a parametrization (\hat{x}_1, \hat{x}_2) to be isotropic if $\hat{Q} = I$.

Define the gradient parametrization as

$$\hat{x}_i = \frac{\partial f}{\partial x_i} \tag{2}$$

Theorem 1. *Given a bivariate function $f(x_i)$ in a region where its Hessian matrix is non-singular, then the gradient parametrization is an isotropic parametrization.*

Proof. The entries of the Jacobian of this parametrization are

$$J_{ij} = \frac{\partial}{\partial x_j} \left(\frac{\partial f}{\partial x_i} \right) = \frac{\partial^2 f}{\partial x_j \partial x_i} = H_{ij}$$

Under this parametrization, the matrix \hat{Q} can be computed as

$$\hat{Q} = J^{-T} Q J^{-1} = J^{-T} (H^T H) J^{-1} = H^{-T} (H^T H) H^{-1} = I$$

■

This proves that an isotropic parametrization exists, and that it can be computed in closed-form using equation (2).

Theorem 2. The isotropic parametrization of equation (2) is unique up to rigid isometry.

Proof. Consider any isotropic parametrization $\tilde{x}(x)$ that is different from the gradient parametrization $\hat{x}(x)$. Because the Jacobian of $\hat{x}(x)$ is the Hessian (H) of f , and H is everywhere non-singular, then $\hat{x}(x)$ is invertible. We can always write the new isotropic parametrization as $\tilde{x} = s \circ \hat{x}$, where $s = \tilde{x} \circ \hat{x}^{-1}$. In a certain sense, s tells us how \tilde{x} differs from \hat{x} .

If K is the Jacobian of $\tilde{x}(x)$, then $S \equiv KH^{-1}$ is the Jacobian of s . From equation (1) and the fact that \tilde{x} is an isotropic parametrization, we know that it must satisfy

$$\hat{Q} = K^{-T} Q K^{-1} = K^{-T} (H^T H) K^{-1} = I$$

or in other words

$$(KH^{-1})^T (KH^{-1}) = S^T S = I$$

Which shows that S is everywhere orthogonal. It follows that, since s is a mapping between two planar domains with an everywhere orthogonal Jacobian, s must be a rigid isometry [7]. ■

It is also a tedious but straightforward calculation to show that the gradient parametrization, in fact, satisfies the differential equation given in [2] (equations 3.3 to 3.6).

We first note that, in the above proofs, there is no explicit mention of the dimensionality of the input domain, suggesting that the above is a proof that the gradient map is the solution to any equivalent problem posed in an arbitrary-dimensional Euclidean space.

Corollary 1. Given an input function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, in a region where its Hessian is non-singular, the gradient parametrization of equation (2) is an isotropic parametrization. This isotropic parametrization is unique up to rigid isometry.

We can conclude that, given an input with a non-singular Hessian, there always exists a mapping that can transform the original domain into one in which the gradient error is isotropically distributed, and that this mapping is always the gradient map composed with some rigid isometry.

2.2 Alignment

For purposes of error analysis, it will later be important to study the relation between the orientations of edges of a triangulation in the original and reparametrized spaces. We present this analysis here to establish some notation and concepts that will be useful through later sections.

We can study the relation between the orientations of edges in the original and transformed domains for meshes that are increasingly fine by looking at simple differential properties of a reparametrization $m : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Given a triangle edge starting at \mathbf{x} and oriented in direction \mathbf{v} , it will be oriented in direction $\mathbf{w} = m(\mathbf{x} + \mathbf{v}) - m(\mathbf{x})$ when transformed by m . We can see that, in the limit as triangles become smaller and smaller, and their edges are equally shrunk, an edge with orientation \mathbf{v} will map onto an edge oriented in the direction of $\mathbf{w} = D_{\mathbf{v}}m$ (the derivative of m in the direction \mathbf{v}) in the reparametrized domain.

Given an input function f with an everywhere non-singular Hessian,

Define an *aligned isotropic* parametrization to be an isotropic parametrization that maps the eigenvector directions of H onto the coordinate directions of the isotropic domain.

From our knowledge of the properties of isotropic parametrizations, we can easily compute the form that an aligned isotropic parametrization has. If the (generally spatially varying) Hessian of the input is $H = R^T \Lambda R$, where R is orthogonal, and Λ is diagonal, then, as proved in section 2.1, any isotropic parametrization m will have a Jacobian $K = SH = SR^T \Lambda R$, where S is a constant orthogonal matrix.

This parametrization will transform the i -th eigenvector \mathbf{v}_i of H into

$$\mathbf{w}_i = D_{\mathbf{v}_i}m = K\mathbf{v}_i = SR^T \Lambda R\mathbf{v}_i = (SR^T) \cdot (\lambda_i \hat{e}_i) \quad (3)$$

where λ_i is the eigenvalue corresponding to \mathbf{v}_i and \hat{e}_i is the i -th coordinate direction. By definition, an aligned isotropic parametrization is one for which, for all i , this \mathbf{w}_i is in the direction of \hat{e}_i . From equation (3) and the fact that SR^T is an orthogonal matrix, we conclude that for m to be aligned it must be that $SR^T = I$, that is $S = R$. In conclusion, we see that an aligned isotropic parametrization is an isotropic parametrization that is the composition of the gradient parametrization with an isometry with Jacobian S , where at every point it is $S = R$.

Because, as proved in section 2.1, the orthogonal matrix S is constant, we can state that an aligned isotropic parametrization only exists for input functions f for which R is constant (e.g. a quadratic f). In conclusion, for general inputs f for which R is spatially varying, an isotropic reparametrization exists, but an aligned isotropic one doesn't. This fact will prove to be crucial in later sections.

3 Asymptotically Optimal Meshing

We are now in a good position to analyze the problem of asymptotically optimal meshing with respect to gradient error as defined above. Consider a mesh such that every triangle in the mesh has gradient error (as defined in section 1) bounded by a constant ϵ . For any given mesh and bound ϵ , we define the *efficiency* of the mesh to be the average area of its triangles. An optimal mesh is one that has the highest efficiency among all others for a given ϵ bound, because it must have the minimum number of triangles when meshing any finite domain.

Given a mesh \mathcal{T} with an error bound of ϵ and efficiency $\nu(\mathcal{T})$, its *competitiveness ratio* is defined as $\nu(\mathcal{T})/\nu_o(\epsilon)$, where $\nu_o(\epsilon)$ is the optimal efficiency for this error bound. Note that for a domain which is a finite area of the plane (x_1, x_2) , we can also measure the competitiveness ratio using the “average area” in a reparametrized domain (\hat{x}_1, \hat{x}_2) . This is true because the total area before and after the reparametrization is simply related by a constant.

A meshing algorithm \mathcal{A} that can take any ϵ and produce a mesh $\mathcal{T} = \mathcal{A}(\epsilon)$ with an error bound of ϵ is said to be asymptotically optimal to within a constant λ if $\lim_{\epsilon \rightarrow 0} \frac{\nu(\mathcal{A}(\epsilon))}{\nu_o(\epsilon)} \geq \lambda$.

3.1 Transformational Meshes

We describe here the process of obtaining a mesh from a transformation, and the subsequent steps that we take to analyze the error of its triangles. We first assume that the transformation, or reparametrization, that we apply to the original domain where the input \mathbf{f} is defined is an isotropic transformation, and therefore is the gradient map composed with some isometry. As before, we call the original domain x and the isotropic one \hat{x} .

For any given value of the error threshold ϵ , we seek to find a triangulation with squared gradient error bounded by ϵ . We first lay out a regular grid of right-isosceles triangles in the \hat{x} -domain, as in figure 1 (left). We then map the vertices of these triangles to the x -domain and connect them with straight edges using their original connectivity, as shown in figure 1 (middle). This defines the transformational mesh. The function f is then sampled at the vertices of the transformational mesh, and approximated as a piece-wise linear function.

To analyze the error of the transformational mesh’s triangles we take another further step, and transform the mesh back to the isotropic domain. Consider the highlighted triangle in the x -domain of figure 1 (middle), with vertices x_1, x_2 , and x_3 . When we map the triangle back onto the \hat{x} -domain we obtain the highlighted triangle $\Delta\hat{x}_1\hat{x}_2\hat{x}_3$ of figure 1 (right). Notice that, although the vertices of this triangles lie on a uniform grid, its edges are in general curved, since the gradient map is in general a non-linear transformation. We can compute the gradient of the triangle $\Delta x_1x_2x_3$, which is simply

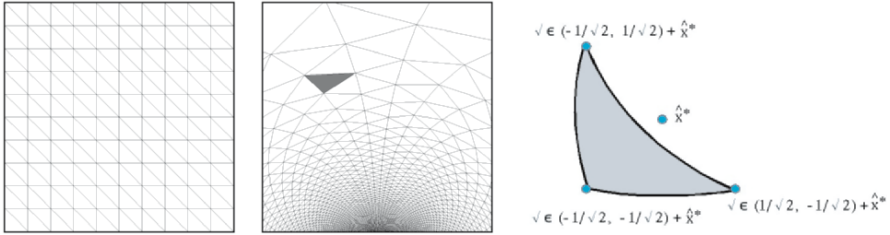


Fig. 1. A regular grid of right-isosceles triangles in the reparametrized domain (left). The corresponding set of triangles shown in the original domain (a transformational mesh) (middle). The marked triangle transformed back to the reparametrized domain (right).

the gradient of the piece-wise linear approximation formed by $\Delta x_1 x_2 x_3$. This gradient can be identified as some point \hat{x}^* in the \hat{x} -domain, as shown in figure 1 (right). This is so because the \hat{x} -domain really represents (up to isometry) the gradient space of the input function, and so the gradient \hat{x}^* will correspond to some location in the \hat{x} -domain.

Once this setup is complete, the task of measuring gradient error is simple. In the x -domain, the squared gradient error between any point in the interior of a triangle and the piece-wise linear approximating mesh is the squared distance between the gradient of \mathbf{f} at that point and the gradient of the triangle that the point lies in. In other words, we can measure this error in the \hat{x} -domain by simply taking squared euclidean distances between any point interior to a transformed triangle and its gradient \hat{x}^* . Finally, we point out that composing the gradient parametrization with an isometry does not change this reasoning since, by definition, isometries do not change distances between transformed points. Thus the above analysis applies to any isotropic parametrization.

3.2 Quadratic Input

We first analyze the behavior of the above construction when applied to a simple quadratic input \mathbf{f} . The relevance of this simpler case is that, as pointed out in [2], the behavior of an arbitrary input is very similar to that of the quadratic case when we look at meshes in the limit as they become finer and finer.

Consider an input function \mathbf{f} with constant Hessian. This Hessian H is symmetric real and can be decomposed as $H = R^T \Lambda R$, where Λ is diagonal and R is a rotation by θ radians. Let us refer to the eigenvalues of H as λ_1 and λ_2 . Consider for analysis a right-isosceles triangle in the \hat{x} domain with coordinates $\hat{x}_1 = \hat{o} + \sqrt{\frac{\epsilon}{2}}(-1, -1)$, $\hat{x}_2 = \hat{o} + \sqrt{\frac{\epsilon}{2}}(1, -1)$, and $\hat{x}_3 = \hat{o} + \sqrt{\frac{\epsilon}{2}}(-1, 1)$

(because it is expressed in generic form as a function of the coordinates of its circumcenter \hat{o} , this analysis applies to any triangle in a regular grid of right-isosceles triangles.) We use an isotropic parametrization \hat{x} in general form, that is, computed as the gradient parametrization composed by an isometry with Jacobian S , where S is a rotation by α radians. Note that, because the gradient map of a quadratic function is a linear map, both this triangle and its corresponding triangle in the x -domain have straight edges. In order to analyze the error inside this triangle, we compute its gradient ∇f^* (the gradient of its supporting plane) and the location of this gradient in the \hat{x} -domain: $\hat{x}^* = S\nabla f^*$. We can show (see appendix) that \hat{x}^* is such that

$$\|\hat{x}^* - \hat{o}\| = \sqrt{\epsilon} |\sin[2(\theta - \alpha)]\mu| \tag{4}$$

with

$$\mu = \frac{1}{2\sqrt{2}}(\lambda_1 - \lambda_2) \left[\frac{\lambda_1}{\lambda_2}(1 + \sin(2\theta)) + \frac{\lambda_2}{\lambda_1}(1 - \sin(2\theta)) \right] \tag{5}$$

From equation (4) we can conclude that if $\theta = \alpha$, that is, if $S = R$ and therefore if the isotropic reparametrization is aligned, as defined in section 2.2, then $\hat{x}^* = \hat{o}$. In this case, because the error of a triangle is the maximum distance between any of its interior points and \hat{x}^* , we can conclude that the error of this triangle is exactly ϵ . (Recall that since f is quadratic, the isotropic image of $\Delta_{x_1x_2x_3}$ is a triangle with straight edges.)

Clearly, as was argued in [2], in any optimal triangulation, the isotropic image of each triangle cannot have an area greater than that of an equilateral triangle circumscribed by a circle of radius $\sqrt{\epsilon}$ (otherwise, it could not have gradient error bounded by ϵ). Each of our right-isosceles triangles is only 23% smaller than such an equilateral triangle, so the average area (measured in the isotropic domain) of the triangles obtained by this algorithm can be no smaller than 77% of the average area of the triangles of an optimal triangulation. Therefore if the reparametrization is aligned isotropic, we conclude, in agreement with [2], that the induced transformational mesh is asymptotically optimal.

We can also compute a lower bound on the approximating error (E) for any right-isosceles triangle. In particular, for any point \hat{x} interior to the triangle we have

$$\begin{aligned} E(\hat{x}) &= \|\hat{x} - \hat{x}^*\|^2 = \|\hat{x} - \hat{o} - (\hat{x}^* - \hat{o})\|^2 \\ &\geq \max\{0, \|\hat{x}^* - \hat{o}\| - \|\hat{x} - \hat{o}\|\}^2 \\ &\geq \max\{0, \|\hat{x}^* - \hat{o}\| - \sup\{\|\hat{x} - \hat{o}\|\}\}^2 \\ &\geq \max\{0, \sqrt{\epsilon} |\sin[2(\theta - \alpha)]\mu| - \sqrt{\epsilon}\}^2 \\ &= \epsilon \cdot \max\{0, |\sin[2(\theta - \alpha)]\mu| - 1\}^2 \end{aligned} \tag{6}$$

We can notice from equation (6) that if $\theta \neq \alpha$, that is, if $S \neq R$ (the reparametrization is not aligned), then the gradient error in a triangle can

be arbitrarily large if we are not to place any restrictions on the amount of anisotropy of f .

Intuitively, the amount of gradient *variation* of f inside such a triangle is bounded by ϵ , but in the non-aligned case, the gradient of the piece-wise linear approximation can still be significantly different from the gradient of f (everywhere) in the triangle.

In figure 2 we show two triangulations resulting from isotropic parametrizations. On the left, we show the point-wise gradient error obtained when using the aligned parametrization. In this case, the triangles all have small (90° or less) angles, they have zero error at their circumcenter (at the midpoint of one of their edges), and error bounded by ϵ elsewhere. On the right, we show the result using a non-aligned parametrization. In this case, the triangles can have arbitrarily large error, they might not have zero error anywhere (they might actually not interpolate the gradient), and they often have large angles. This link between large gradient error and large angles has been pointed out before (see [4] for example).

In [2] an aligned isotropic reparametrization for the quadratic case is computed ([2] equation 2.8), and in [2] theorem 2.1, it is proved that this will result in an optimal triangulation. As we have shown, there are in fact many isotropic reparametrizations, but only the one that is aligned has guarantees that it leads, for any quadratic input, to an asymptotically optimal mesh. This distinction is not made explicit in [2]. This leads them to incorrectly conclude in their section 3 (the non-quadratic case), that the isotropic parametrization they find (which is generally not everywhere aligned), will result in an asymptotically optimal triangulation.

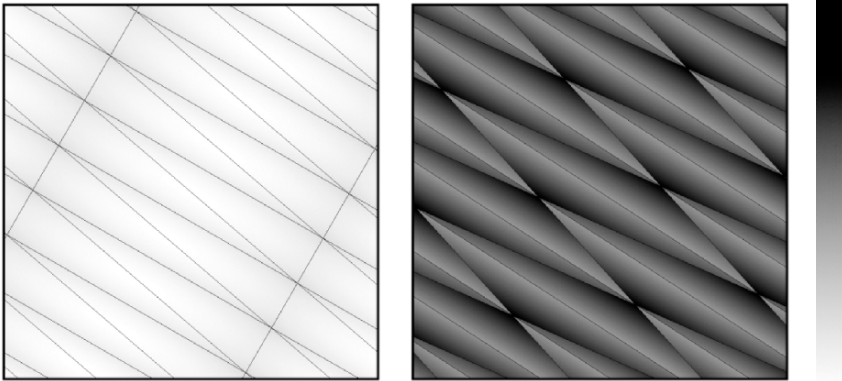


Fig. 2. Gradient approximation error graphed for a transformational mesh derived from an aligned isotropic reparametrization (left), and a non-aligned isotropic one (right). The bar on the far right shows the error scale (white is zero error). The input function is $f(x_1, x_2) = 0.156(x_1)^2 + 0.318(x_2)^2 - 0.281x_1x_2$.

3.3 Arbitrary Input

For non quadratic input, the eigenvector directions of $H = R^T \Lambda R$, and therefore R , will in general be spatially varying. As a result, for an isotropic reparametrization with Jacobian SH where S is orthogonal, (except for a curve of measure 0) we will generally have $S \neq R$, which corresponds to $\theta \neq \alpha$ in equation 6. In these regions, if $\lambda_1 \gg \lambda_2$ (f is highly anisotropic), then for any regular right-isosceles triangulation of the isotropic reparametrized domain, the gradient error will be accordingly large, and the triangles will have to be made much smaller than optimal. If no bounds are placed on the anisotropy of f , then to meet a gradient error threshold the triangles may be arbitrarily small, and an optimality bound cannot in general be met.

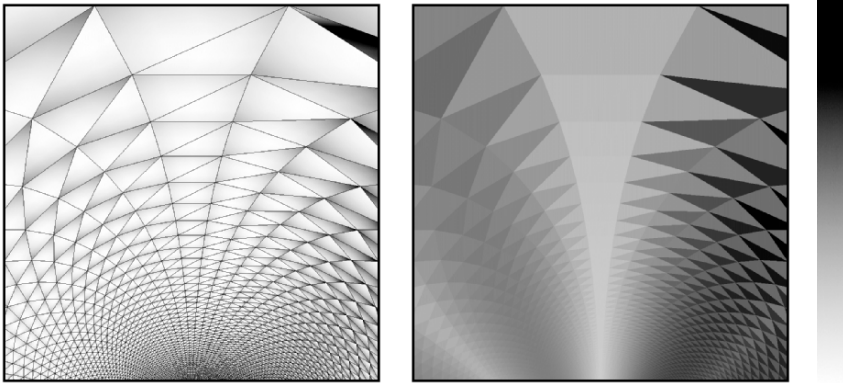


Fig. 3. Point-wise error of a transformational triangulation (left), and worst per-triangle error of the same triangulation (right). The bar on the far right shows the error scale (white is zero error). The lowest worst error is in the triangles in a vertical band around the center of each triangulation, where the eigenvectors of H are better aligned with the coordinate directions in the isotropic reparametrization. The input function is $f(x_1, x_2) = 1/\sqrt{(x_1)^2 + (x_2)^2}$. Shown is a region slightly away from the singularity.

One can of course still obtain upper bounds on the error, if one imposes an anisotropy bound on f , but this is a much weaker result than hoped for, and much less than what can be achieved in the quadratic case, where asymptotic optimality is guaranteed for an arbitrary quadratic input.

In practice we have seen that the error can grow quite large even when the amount of anisotropy is not very extreme. In figure 3 we show a transformational mesh obtained from an isotropic reparametrization. The lowest errors are found in a vertical band around the center of each image, where the alignment happens to be best (S is closest to R).

3.4 Relaxing Continuity

We can attempt to relax the problem definition in a way that might make it easier to obtain asymptotically optimal meshes. In particular, we can consider a variation of the problem where we do not sample f at vertices, but directly interpolate both f and its gradient at some interior point of a subset of the mesh's triangles.

Clearly if we relax the problem definition and allow the mesh to not have C^0 continuity, then it is possible to obtain asymptotically optimal meshes in the limit. We can lay out a uniform grid of triangles in isotropic space, and simply treat each triangle separately. We then require that each triangle exactly interpolate the function f and its gradient at its circumcenter. The error in each triangle then will be bounded by ϵ , the resulting mesh is optimally efficient in the limit. Since this method is simply interpolating f and its gradient at its center, we can just as easily use a uniform grid of equilateral triangles.

In practice a mesh that doesn't even have C^0 continuity might not be very useful. We can do slightly better by constructing the “non-conforming” mesh of figure 4, which will be guaranteed to be C^0 at the edges' midpoints. In this mesh of equilateral triangles (in isotropic space), the input f and its gradient are sampled at the circumcenter of every marked triangle. This determines the piece-wise linear approximation \bar{f} at triangles that are marked. We construct \bar{f} at all other triangles by reading the values of \bar{f} at the midpoints of edges of marked triangles. We can see that every edge in the triangulation is incident to exactly one marked triangle, and so this construction exactly determines \bar{f} without over-constraining it.

In appendix B, we show that in the quadratic case, this produces asymptotically optimal meshes when any isotropic reparametrization is used, even an unaligned one. Because in the non-quadratic case we can always compute

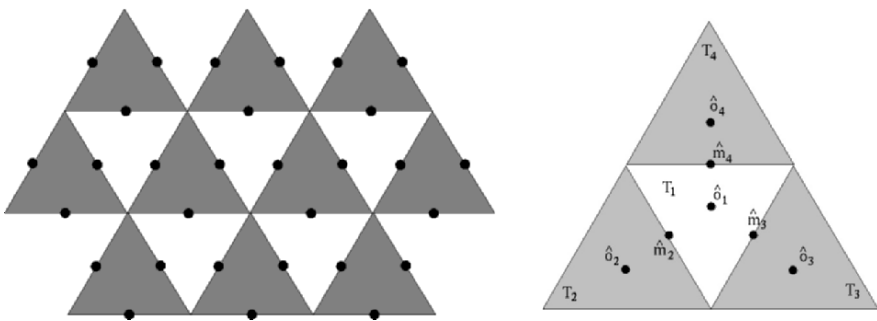


Fig. 4. A non-conforming triangle mesh that is C^0 continuous only at the edge midpoints in the isotropic domain (left). The construction used in the appendix for showing C^0 continuity at midpoints and bounded error (right).

an isotropic parametrization and alignment is not necessary, and because in the limit the approximation's behavior is dominated by its low order behavior, we conjecture that a formal proof of its optimality can be constructed.

Again, the obvious drawback is that this is a non-conforming triangle mesh that is only guaranteed to be C^0 at edge midpoints.

4 Asymptotically Optimal Triangulations

Although we have shown that a transformational mesh, as defined, cannot produce guarantees on asymptotical optimality for an arbitrary input, our analysis does provide a deeper understanding of the problem of asymptotically optimal meshing.

We have also shown how a regular grid of right-isosceles triangles in an isotropic reparametrization has error bounded by ϵ if all triangles are perfectly aligned (that is, if their edges in the original parametrization are aligned with the eigenvectors of H), and that their efficiency is ϵ : only 23% lower than the maximum attainable efficiency. Therefore such a triangulation is guaranteed to be asymptotically optimal to within at least 77%.

In practice, obtaining such a triangulation may prove too hard, and we can obviously allow some flexibility by permitting the triangulation to include non-aligned triangles that have an error bounded by ϵ , and an area possibly much smaller than the maximum attainable efficiency, so long as the proportion of these triangles to the total goes to zero when we make the triangulation increasingly fine. This approach may provide a future avenue for obtaining provably asymptotically optimal meshes. We plan to explore this possibility in future work.

References

1. Robert M. Wald (1984) *General Relativity*. University of Chicago Press, Chicago.
2. E. F. D'Azevedo and R. B (1991) Simpson, On Optimal Triangular Meshes for Minimizing the Gradient Error. *Numerische Mathematik*. 59, 321–348
3. E. F. D'Azevedo (1991) Optimal Triangular Mesh Generation by Coordinate Transformation. *SIAM Journal on Scientific and Statistical Computing*. 12, 4, 755–786.
4. J. R. Shewchuk (2002) What Is a Good Linear Finite Element? – Interpolation, Conditioning, Anisotropy, and Quality Measures. *Eleventh International Meshing Roundtable*. 115–126.
5. W. Huang (2005), Anisotropic mesh adaptation and movement. *Lecture notes for Peking Univ. Workshop on Adaptive mesh methods*.
6. L. Chen and J. Xu (2004) Optimal Delaunay Triangulations. *Journal of Computational Mathematics*. 22, 2, 299–308.
7. M. Do Carmo (1976) *Differential Geometry of Curves and Surfaces*. Prentice Hall, New Jersey.

Appendix A

Suppose that we are given an input quadratic function \mathbf{f} with Hessian $H = R^T \Lambda R$, where, as in section 3.2, R is a rotation by θ radians, and Λ has eigenvalues λ_1 and λ_2 . We consider an isotropic parametrization \hat{x} of the x -domain that is the composition of the gradient parametrization with an isometry having orthogonal Jacobian S , where S is a rotation of the plane by α radians. We first compute the expression of the gradient of a triangle that, in the \hat{x} -domain, has vertices with coordinates $\hat{x}_1 = \sqrt{\frac{\epsilon}{2}}(-1, -1)$, $\hat{x}_2 = \sqrt{\frac{\epsilon}{2}}(1, -1)$, and $\hat{x}_3 = \sqrt{\frac{\epsilon}{2}}(-1, 1)$. This triangle can be mapped to the x -domain and considered as a piece-wise linear element. As such, it will have a gradient, which we then transform to the \hat{x} -domain by applying the rotation S . The final coordinates of this transformed gradient \hat{x}^* in the \hat{x} -domain can be used to derive error bounds for the triangle. We then show that if we were to translate all the points of this triangle in the \hat{x} -domain by a vector \hat{o} we would get the same result, that is, the value of \hat{x}^* is the same as before except that it is also translated by \hat{o} .

If the input function is an arbitrary quadratic function $f(x) = c + g^T x + \frac{1}{2}x^T Hx$, where c is an arbitrary constant and g an arbitrary vector, then we can write the relation between the x and the \hat{x} domains. Because the parametrization is by definition $\hat{x} = S\nabla f(x)$, and $\nabla f(x) = g + Hx$, then we can say that

$$x = H^{-1}(S^T \hat{x} - g) \tag{7}$$

Because the triangle $\Delta x_1 x_2 x_3$ linearly approximates f in the x -domain, we can write

$$f^* - f_1 = u \cdot (f_2 - f_1) + v \cdot (f_3 - f_1)$$

where $f^*(x)$ is the linear approximation, $f_i = f(x_i)$, and u and v are (the first two) local barycentric coordinates of the triangle. The scalars (u, v) are such that at any point x we can write

$$x - x_1 = P \begin{bmatrix} u \\ v \end{bmatrix}$$

$$P = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \end{bmatrix}$$

We can write the gradient of f^* as

$$\frac{\partial f^*}{\partial x_i} = \frac{\partial f}{\partial u} \frac{\partial u}{\partial x_i} + \frac{\partial f}{\partial v} \frac{\partial v}{\partial x_i}$$

which can be rewritten as

$$\nabla f^* = P^{-1} \begin{bmatrix} f_2 - f_1 \\ f_3 - f_1 \end{bmatrix} \tag{8}$$

We now compute the two terms of the right-hand side of equation (8).

Because $x_2 - x_1 = H^{-1}S^T(\hat{x}_2 - \hat{x}_1)$ and $x_3 - x_1 = H^{-1}S^T(\hat{x}_3 - \hat{x}_1)$, and from the definition of P and of $x_1, x_2,$ and x_3 we find that $P = \sqrt{2\epsilon}H^{-1}S^T$ and

$$P^{-1} = (\sqrt{2\epsilon})^{-1}SH$$

On the other hand, the expressions for $f_2 - f_1$ and $f_3 - f_1$ simplify to

$$f_2 - f_1 = \frac{\epsilon}{4} \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}^T SH^{-1}S^T \begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix}^T SH^{-1}S^T \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right)$$

$$f_3 - f_1 = \frac{\epsilon}{4} \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix}^T SH^{-1}S^T \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix}^T SH^{-1}S^T \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right)$$

If we call $B = SHS^T$, then we know that $\det(B) = \det(H)$, and that B can be decomposed as $B = (RS^T)^T \Lambda (RS^T)$ where RS^T is a rotation of the plane by $\theta - \alpha$ radians. In particular, the off-diagonal entry of B is $B_{12} = \frac{1}{2} \sin(\theta - \alpha)(\lambda_1 - \lambda_2)$, and we can also verify that $B_{12}^{-1} = -B_{12}/\det(B) = -B_{12}/\det(H)$.

This allows us to simplify the above equations into

$$f_2 - f_1 = \epsilon B_{12}/\det(H)$$

$$f_3 - f_1 = \epsilon B_{12}/\det(H)$$

From this we can compute

$$\nabla f^* = P^{-1} \begin{bmatrix} f_2 - f_1 \\ f_3 - f_1 \end{bmatrix}$$

$$= \sqrt{\epsilon} B_{12} \frac{SH}{\det(H)} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Because $\hat{x}^* = S\nabla f^*$, then

$$\hat{x}^* = \frac{\sqrt{\epsilon}}{2} \sin[2(\theta - \alpha)(\lambda_1 - \lambda_2)] \frac{S^2 H}{\det(H)} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \tag{9}$$

We can compute the norm of \hat{x}^* by taking into account that S^2 is an orthogonal matrix and therefore

$$\|\hat{x}^*\| = \frac{\sqrt{\epsilon}}{2\sqrt{2}} |\sin[2(\theta - \alpha)(\lambda_1 - \lambda_2)]| \left[\frac{\lambda_1}{\lambda_2} (1 + \sin(2\theta)) + \frac{\lambda_2}{\lambda_1} (1 - \sin(2\theta)) \right] \tag{10}$$

We now consider the more general case of an arbitrary triangle $\mathcal{T} = \Delta \hat{x}'_1 \hat{x}'_2 \hat{x}'_3$ in the \hat{x} -domain that is the translation of the above $\Delta \hat{x}_1 \hat{x}_2 \hat{x}_3$ by an arbitrary vector \hat{o} in the \hat{x} -domain. Where in this notation it is $\hat{x}' = \hat{x} + \hat{o}$. In this case we want to compute $\hat{x}^* - \hat{o}$ and its norm, where \hat{x}^* is the gradient of triangle $\Delta x'_1 x'_2 x'_3$ transformed by S . We can write the relation between points in the \hat{x} and x -domains as

$$x = H^{-1}(S^T \hat{x}' - g) = H^{-1}(S^T \hat{x} + S^T \hat{o} - g) = H^{-1}[S^T \hat{x} - (g - S^T \hat{o})] \quad (11)$$

We now consider the problem of approximating not f but the function $\bar{f}(x) = f(x) - \hat{o}^T Sx$. Because the gradient of \bar{f} at the origin is $g - S^T \hat{o}$, we can see that equation (11) is formally the same as (7) if we are approximating \bar{f} as opposed to f . We can apply the same analysis as above and find that, because \bar{f} and f have the same Hessian, the approximated gradient of \bar{f} is

$$\nabla \bar{f}^* = \frac{\sqrt{\epsilon}}{2} \sin[2(\theta - \alpha)(\lambda_1 - \lambda_2)] \frac{SH}{\det(H)} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

But because a linear approximation that is based on interpolating a function at three points reproduces linear functions, and $f = \bar{f} + \hat{o}^T Sx$, we find that if we had been approximating f we would've obtained that $\nabla f^* = \nabla \bar{f}^* + S^T \hat{o}$. We are now interested in computing $\hat{x}^{t*} - \hat{o}$, which we can obtain by multiplying the expression for ∇f^* by S , and subtracting \hat{o}

$$\hat{x}^* - \hat{o} = \frac{\sqrt{\epsilon}}{2} \sin[2(\theta - \alpha)(\lambda_1 - \lambda_2)] \frac{S^2 H}{\det(H)} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

whose norm is

$$\|\hat{x}^{t*} - \hat{o}\| = \frac{\sqrt{\epsilon}}{2\sqrt{2}} |\sin[2(\theta - \alpha)](\lambda_1 - \lambda_2)| \left[\frac{\lambda_1}{\lambda_2} (1 + \sin(2\theta)) + \frac{\lambda_2}{\lambda_1} (1 - \sin(2\theta)) \right]$$

Which has the same form as equations (9) and (10).

Appendix B

We show here that the triangulation shown in figure 4 (left) has gradient error bounded by ϵ and has C^0 continuity at edge midpoints. Given an input function that is an arbitrary quadratic function $f(x) = c + g^T x + \frac{1}{2} x^T Hx$, where c is an arbitrary constant and g an arbitrary vector, then we can write the relation between the x and the \hat{x} domains. Because the parametrization is, by definition, $\hat{x} = S \nabla f(x)$, and $\nabla f(x) = g + Hx$, we can say that

$$x = H^{-1}(S^T \hat{x} - g) \quad (12)$$

Consider the triangles shown in figure 4 (right) in the isotropic domain \hat{x} . Their circumcenters are \hat{o}_i and the midpoint of their shared edges are $\hat{m}_i = \frac{1}{2} \hat{o}_1 + \frac{1}{2} \hat{o}_i$, $i \in \{2, 3, 4\}$ (their counterparts in the original domain are o_i and m_i). For those triangles that are “marked”, the input f and its gradient are sampled at their circumcenter \hat{o}_i , $i \in \{2, 3, 4\}$, and thus their gradient error is ϵ . This sampling completely determines the piece-wise linear approximation

f^* of f inside marked triangles. At the midpoints \hat{m}_i of edges, we sample f^* from the single marked triangle incident to the edge. Setting the values of f^* at \hat{m}_2 , \hat{m}_3 , and \hat{m}_4 then completely determines the piece-wise linear approximation at \mathcal{T}_1 . \mathcal{C}^0 continuity at the midpoints \hat{m}_i is guaranteed by construction. To prove that the gradient error at \mathcal{T}_1 is bounded by ϵ we simply show that the construction above will produce the same values of f^* at the midpoints \hat{m}_i (and therefore the same linear approximation inside \mathcal{T}_1) as if we had sampled f and ∇f at the circumcenter of \mathcal{T}_1 .

For triangle \mathcal{T}_i , $i \in \{2, 3, 4\}$, we can compute the sampled value $f^*(m_i) = f(o_i) + \nabla f(o_i)^T(m_i - o_i)$. Similarly, for \mathcal{T}_1 , we can compute the value that f^* would have at \hat{m}_i if we had obtained it by sampling f and ∇f at o_1 : $\bar{f}^*(m_i) = f(o_1) + \nabla f(o_1)^T(m_i - o_1)$. We now only have to prove that $f^*(m_i) = \bar{f}^*(m_i)$ for $i \in \{2, 3, 4\}$.

We first compute $f(o_i)$, $i \in \{1, 2, 3, 4\}$. From (12) we know that $o_i = H^{-1}(S^T \hat{o}_i - g)$, and so

$$\begin{aligned} f(o_i) &= c + g^T H^{-1}(S^T \hat{o}_i - g) + \frac{1}{2}(\hat{o}_i^T S - g^T)H^{-1}(S^T \hat{o}_i - g) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_i^T S H^{-1}S^T \hat{o}_i \end{aligned} \tag{13}$$

Because (from our definition of the reparametrization) $\nabla f(o_i) = S^T \hat{o}_i$, we can write

$$\begin{aligned} f^*(m_i) &= f(o_i) + \nabla f(o_i)^T(m_i - o_i) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_i^T S H^{-1}S^T \hat{o}_i + \hat{o}_i^T S H^{-1}S^T(\hat{m}_i - \hat{o}_i) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_i^T S H^{-1}S^T \hat{o}_i \\ &\quad + \hat{o}_i^T S H^{-1}S^T\left(\frac{1}{2}\hat{o}_i + \frac{1}{2}\hat{o}_1 - \hat{o}_i\right) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_i^T S H^{-1}S^T \hat{o}_1 \end{aligned} \tag{14}$$

While on the other hand

$$\begin{aligned} \bar{f}^*(m_i) &= f(o_1) + \nabla f(o_1)^T(m_i - o_1) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_1^T S H^{-1}S^T \hat{o}_1 + \hat{o}_1^T S H^{-1}S^T(\hat{m}_i - \hat{o}_1) \\ &= c - g^T H^{-1}g + \frac{1}{2}g^T H^{-1}g + \frac{1}{2}\hat{o}_i^T S H^{-1}S^T \hat{o}_1 \end{aligned} \tag{15}$$

Meshing Algorithms

High Quality Bi-Linear Transfinite Meshing with Interior Point Constraints

Nilanjan Mukherjee

Meshing & Abstraction Group, Digital Simulation Solutions UGS
2000 Eastman Dr, Milford, OH 45150, USA
nilanjan.mukherjee@ugs.com

Abstract. For a variety of structural finite element analyses on automotive body panels, aerospace wings and space satellite panels, high-quality, structured quadrilateral meshing is imperative. Transfinite meshing, the technique to produce such meshes is severely infringed by the presence of surface-interior point constraints. The present paper attempts to solve the inverse problem of transfinite meshing with interior point constraints. A modified Newton Raphson based solution is proposed to inverse solve Coons bi-linear blending equation. The Coons parametric coordinates are thus determined for a set of face-interior points from their global coordinates. The boundary of the surface is next seeded with “soft-points” at reflected locations and smart-discretized to result in high fidelity, high-quality transfinite meshes.

Keywords: Point constraint, mesh, transfinite, mapped, structured, Coon's Equation, Newton-Raphson

1. Introduction

Mapped meshing or transfinite meshing is an important mesh generation technique, especially with quads, used frequently in a wide gamut of finite element analysis problems. These meshes are structured and hence have a higher solution reliability. These meshes are also economical and if stress-sensitive regions of the outer surface are pre-meshed with such meshes, a lighter tetrahedral mesh is usually produced. However, interior mesh points are hard to honor for transfinite meshes. Usually the nearest node is snapped to the interior point. This depletes element quality often. In the

present paper, an attempt is made to solve the inverse problem of transfinite meshing with interior point constraints. The inverse problem is solved to evaluate Coons parametric coordinates of the interior point constraints from their global world coordinates. A conventional modified Newton- Raphson based solution is proposed to inverse solve Coons' bi-linear blending equation. The boundary of the surface is next seeded with "soft-points" at reflected locations and smart-discretized.

2. Past Research

Mapped or transfinite meshing techniques with both quadrilateral and triangular elements remain to be one of the earliest automatic mesh generation algorithms in the world of surface mesh generation. Zienkiewicz and Phillips [1] report probably one of the earliest papers in this area. They proposed a 2D automatic mesh generation scheme based on isoparametric mapping for flat and curved surfaces. Gordon and Hall [2] defined the transfinite interpolation on the rectangle two years later in 1973. In 1974, Cook [3] used it to construct C^0 continuous quadrangulations of deformed quadrangles. Cook's method induces of C^0 continuous structured meshes on C^0 continuous transfinite patches. Haber et al. [4] discuss a general purpose transfinite mapping technique for a wide range of surfaces. Alain Peronnet [5–7] did several in-depth investigations on transfinite interpolation techniques on both C^1 and G^1 continuous domains for both 2D and 3D surfaces. Mitchell [8–9] and Armstrong [10] reported approaches to automatically identify the corners of a mapped meshable domain and discuss techniques to assign intervals on surface boundaries. However, even after an exhaustive research, no research work was found on the transfinite mesh generation problem with interior point constraints.

3. Problem Statement

The present paper attempts to solve the problem of generating a transfinite mesh on a face geometry such that the grid lines pass through a set of face interior point constraints. When that is attained, a nice smooth structured mesh is produced that has high quality surface elements that are not distracted by the interior constraints. Fig. 1 shows a regular mapped mesh where the interior point constraints are ignored.

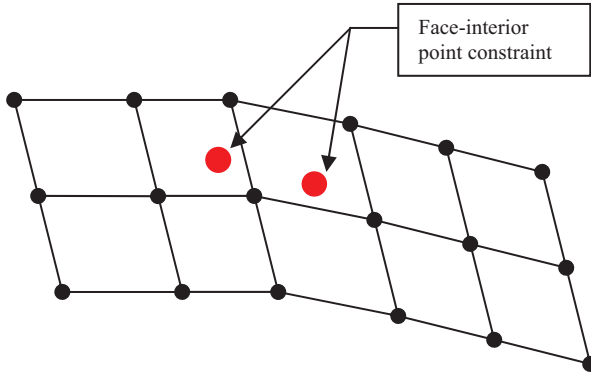


Fig. 1. A mapped mesh with interior constraints ignored

Figure 2 depicts the same geometry with the same mesh, where the interior mesh nodes are snapped to the constraints that are nearest to them. No mesh node is allowed to snap to more than one point constraint, else the topology of the mesh will collapse at that location. The boundary discretization remains unchanged.

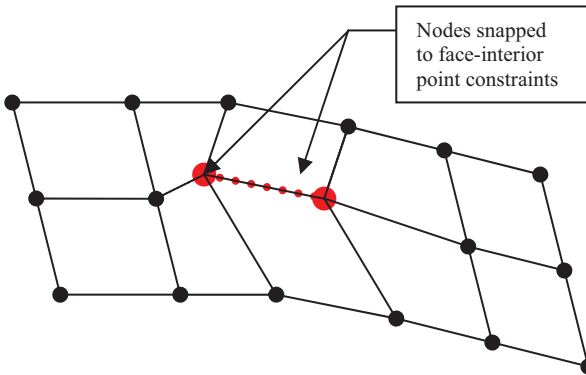


Fig. 2. Mapped mesh with mesh nodes snapped to the nearest interior constraints

When nearest nodes are snapped to the interior mesh points, the element shapes distort resulting in highly skewed elements that are unreliable for stress and dynamic structural analysis. Automobile car body panels need to model arrays of spot-weld points which represent potential high stress areas hence requiring good -quality (low skew) structured meshes connecting them. Most part of the industry accomplishes such meshes through tedious, inefficient, manual techniques.

4. Transfinite Interpolation in 2D Space

Figure 3 depicts a typical 4 sided area that needs to be transfinite meshed. The area is four-sided and require nodes to match up on each pair of “logical” sides.

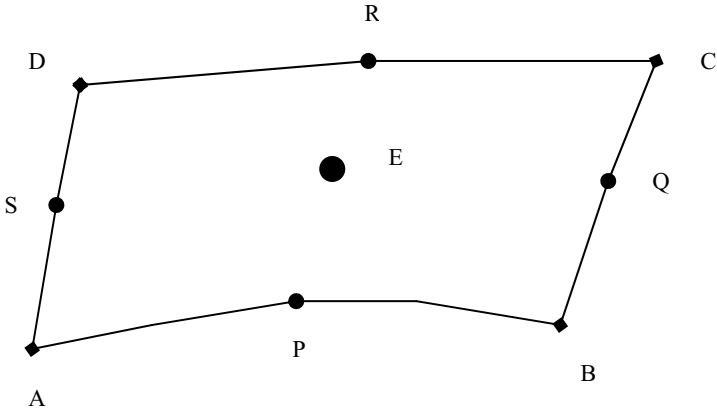


Fig. 3. A 4-sided Coons space

4.1 Coons Blending

Given an area bounded by three or four curves (B-Spline/Bezier) , a surface patch can be created by blending the boundaries using suitable blending functions [11]. The theory of patches and blending was first developed by Coons [12]. Coons blending functions are traditionally used to generate transfinite or mapped meshes on 2D and 3D representation of surfaces or mesh-domains. A 2D four-sided area bounded by four curves (B-spline, Bezier or discrete) as shown in Fig.3. Let P,Q,R,S be functions representing the boundary curves in any cartesian 2D space. Thus,

$$P \equiv Q \equiv R \equiv S \equiv f(x,y) \tag{1}$$

and the rail points are

$$P(x,y) \equiv r(u,0), \quad Q(x,y) \equiv r(u,1), \quad R(x,y) \equiv r(0,v) \text{ and } S(x,y) \equiv r(1,v) \tag{2}$$

where $r(a,b)$ is a generic parametric function that represents each boundary curve in the range of a to b . Also at any point on the boundary curves the cartesian functions can be written as

$$P(x,y) \equiv (P_x, P_y) \tag{3}$$

The corners of the area are denoted by A,B,C,D where

$$A(x,y) \equiv r(0,0), B(x,y) \equiv r(1,0), C(x,y) \equiv r(1,1) \text{ and } D(x,y) \equiv r(0,1) \quad (4)$$

Thus, for any interior node $E(x,y) \equiv r(u,v)$, Coons bilinear blending function can be written as a bullean sum.

$$\begin{aligned} E_x &= (1 - v)P_x + vQ_x + (1 - u)R_x + uS_x - [(1 - u)(1 - v)A_x \\ &\quad + (1 - v)uB_x + v(1 - u)D_x + uvC_x]; \\ E_y &= (1 - v)P_y + vQ_y + (1 - u)R_y + uS_y - [(1 - u)(1 - v)A_y \\ &\quad + (1 - v)uB_y + v(1 - u)D_y + uvC_y]; \end{aligned} \quad (5)$$

In a matrix form, equation (5) can be rewritten for the abscissa as

$$E_x = \begin{bmatrix} -(1-u)(1-v) \\ -u(1-v) \\ -uv \\ -v(1-u) \\ (1-u) \\ u \\ v \\ (1-u) \end{bmatrix} \{\phi\} = [B] \{\phi\} \quad (6)$$

where

$$\{\phi\} = \{ A_x, B_x, C_x, D_x, P_x, S_x, Q_x, R_x \}^T \quad (7)$$

A similar companion equation exists for the ordinate E_y .

4.2 Presence of Interior Point Constraint

If an interior point constraint $F(x,y) \equiv r(u',v')$ exists in the domain closest to node E, (as exhibited in Fig. 1) node E will have to be snapped to location F. The deviation of mesh node E from point constraint F can thus be expressed as

$$f(u,v) = [B] \{\phi\} - F_x \text{ and } g(u,v) = [B] \{\phi\} - F_y \quad (8)$$

As explained before, the aim of this exercise is to minimize the the functionals f,g with respect to u,v as described in eqn. (8). For n interior mesh-points, the problem can be globally described by

$$b = \text{Min}_{(u,v)} \left(\sum_i^n f_i, \sum_i^n g_i \right) \tag{9}$$

The minimization problem can further be expressed as

$$\{f_i \ g_i\}^T = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}^T \{\Delta s_i \ \Delta t_i\}^T \tag{10}$$

Although, this approach has an easy and logical extention in 3D, the 2D approach is mostly used. A 2D domain of the curved surface is developed (or parameter space used) and the mesh is generated in 2D using the improved algorithm. Once the mesh is generated in 2D, a transformation mechanism is used to get the 2D mesh on the 3D surface. This is a standard procedure for generating 2D meshes on developable surfaces and is done no differently for this case.

5. The Inverse Problem and Its Solution

The present scenario leads to an inverse problem as posed by equation (9). During transfinite meshing, Coons equation (6) is used to locate a mesh interior point in the cartesian 2D domain, when its boundary parametric ([B]) and cartesian coordinates ({φ}) are known. With the interior point constraint this problem is reversed. The parametric coordinates (u',v') need to be determined while its cartesian location F(x,y) is known.

In order to solve eqn. (10), a modified Newton-Raphson procedure may be adopted.

Using a modified Newton-Raphson, the solution is given by

$$[J] \Delta Z + F = 0 \tag{11}$$

where $[J] = \text{Jacobian} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}$ (12)

$$\Delta Z = \{ \Delta s, \Delta t \}^T; \quad F = \{f, g\}^T; \tag{13a}$$

The elements of the Jacobian can be expressed as

$$J_{11} = \delta f / \delta s; \quad J_{21} = \delta g / \delta s \tag{13b}$$

$$J_{12} = \delta f / \delta t; \quad J_{22} = \delta g / \delta t \tag{13c}$$

Finally, the change in the parametric coordinates during the i th iteration step can be written as

$$\Delta s_i = (-J_{22} f_{i-1} + J_{12} g_{i-1})/|J| \quad (14a)$$

$$\Delta t_i = (J_{21} f_{i-1} - J_{11} g_{i-1})/|J| \quad (14b)$$

Iteratively solve the following equation, till it converges

$$(s, t)_i = (s, t)_{i-1} + (\Delta s_i, \Delta t_i) \quad (15)$$

6. Solution Convergence

The solution to eqn. (9) is usually quite speedy and usually converges for an error norm $|\varepsilon_i| \leq 1e-05$. The error norm $|\varepsilon|$ is a root-mean-square of the collective differences of the evaluated coordinates across successive iterations and can be expressed as

$$|\varepsilon_i| = (s_i - s_{i-1})^2 + (t_i - t_{i-1})^2 \quad \text{for the } i\text{th iteration.} \quad (16)$$

However, the convergence of the solution depends on the geometry of the boundary. If the rail curves are represented by higher order rational splines, the solution could slow down a bit; it could slow down a little further if the face is represented by facets (implying the boundary curves are represented by poly-lines). However, for all practical purposes the solution time is insignificant compared to the mesh generation time on these surfaces.

7. Boundary Reflection

Once the inverse problem is solved, the parametric coordinates of the interior points are known in the Coons' domain. These parametric coordinates are now used to create reflected locations on the boundaries of the domain. Figure 4 shows two face interior constraints E and F whose parametric locations in the Coon's space are given by E (s_1, t_1) and F (s_2, t_2) . The solution to the inverse problem gives us the Coon's parametric locations of these points.

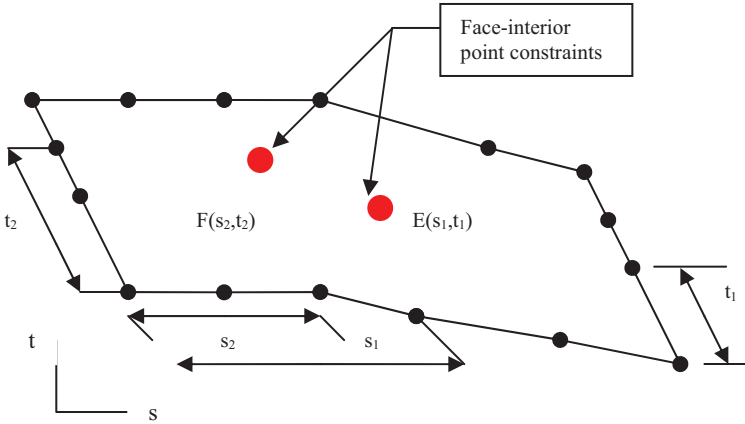


Fig. 4. Interior point constraints reflected on the boundary via “soft-points”

7.1 Soft Points and Pseudo-Edges

The Coons space parametric coordinates of interior points E & F are next used to create 4 temporary nodes on the rail curves at parametric locations s_1, s_2, t_1 and t_2 . These nodes act as “soft-points” or soft-constraints. These soft points need to be honored during boundary discretization. i.e. mesh nodes need to be created at these locations. As a result, it is ensured that the interior points are always reflected on the boundary. The resulting transfinite mesh lines thus flow through the interior point constraints. It is important to note that when 2 or more soft-point locations are close enough on a given pair of sides, they are merged into one.

For every interior point constraint, 4 such soft-points need to be created on the 4-sides of the area. Each side of the 4-sided area is called a “Pseudo-Edge”. It is important to remind here that each pseudo-edge is actually a collection of one or more CAD edges. When the pseudo-edge is discretized, the soft-point acts like a “pseudo-vertex”. A node is always created on it. This ensures, that when a pseudo-edge with a given element count is discretized, these “soft” locations are guaranteed to get a node. The resulting mesh, gets interior nodes that are very close to the face-interior constraints. These nodes are now snapped to the constraint location.

Conventional mesh relaxation methods try to solve the same problem, but they would hold the boundary nodes fixed. The present algorithm, in contrast, determines apriori suitable boundary node locations so as to

minimize the distortion of the mesh. Because there is more freedom on the boundary, with the present algorithm, the chances of producing a better structured mesh is stronger.

7.2 Boundary Discretization

When face-interior point constraints are present, boundary discretization changes in a two-fold manner. Firstly, it creates a non-uniform seeding in most cases, secondly it alters the element count on a given pair of sides. The maximum element count on a side or a Pseudo-Edge can be given by

$$m_i = \lfloor (L_i/s) , (n + 1) \rfloor_{\max} \tag{17}$$

where

m = element count on pseudo-edge i

L_i = length of pseudo-edge i

n = number of unique boundary reflections on a side

s = meshing size

A pseudo-edge is an assembly of p edges. However, when this pseudo-edge is pre-discretized with q soft-points (reflected location of face-interior constraints), the edge is assumed to be logically composed of $r = (p + q)$ sub-edges. The 3D coordinates of node j to be placed at parametric location s_j can be expressed as

$$N_j(x,y,z) = (1 - s_{ij}).P_{ls}(x,y,z) + s_{ij}.P_{le}(x,y,z) \tag{18}$$

where this node is found to lie on the l -th sub-edge; s_{ij} represent its local parametric co-ordinate on the l -th sub-edge; P_{ls} and P_{le} signify the start and end locations of the l -th sub-edge. The local parametric location is given by

$$s_{ij} = \left(s_j \cdot l_{tot} - \sum_{k=1}^{l-1} l_k \right) / l_l \quad \text{where } l_{tot} = \text{length of the } r \text{ sub-edges} \tag{19}$$

l_l = length of the l -th sub-edge that contains this node

7.3 Boundary Blending

We have already observed that presence of interior point constraints affects the boundary discretization of the face. Because interior points are

reflected on the boundary, the boundary discretization becomes non-uniform. When boundary node distribution becomes non-uniform, a boundary blended bi-linear transfinite interpolation becomes necessary to make sure that the mesh line flow is smooth and boundary effects are well reflected in the interior of the space. Coons eqn. (5) now changes to

Thus, for any interior node $E(x,y) \equiv r(u,v)$, Coons bilinear blending function can be written as a bullean sum.

$$E_x = (1 - v')P_x + v'Q_x + (1 - u')R_x + u'S_x - [(1 - u')(1 - v')A_x + (1 - v')uB_x + v'(1 - u')D_x + u'v'C_x]; \quad (20a)$$

$$E_y = (1 - v')P_y + v'Q_y + (1 - u')R_y + u'S_y - [(1 - u')(1 - v')A_y + (1 - v')u'B_y + v'(1 - u')D_y + u'v'C_y]; \quad (20b)$$

where the boundary modified parametric coordinates can be written as

$$u' = \{(1 - \eta)u_1 + \eta u_2\} / \{1 - (u_2 - u_1)(v_2 - v_1)\} \quad (21a)$$

$$v' = \{(1 - \psi)v_1 + \psi v_2\} / \{1 - (u_2 - u_1)(v_2 - v_1)\} \quad (21b)$$

where u_1, u_2 represent the parametric coordinates of the pair of guide nodes on the u -rail curves and v_1, v_2 represent the corresponding parameters on the v -rail curves. ψ and η represent the u and v -directional coordinate for this (u,v) Coons space location assuming a uniform boundary distribution.

8. Examples and Discussion

Figure 5 depicts a flat semi-annular surface with 6 interior point constraints. In automobile body panels, such point constraints usually represent spot-welds. A transfinite mesh of size 5 length units is generated on the surface. The mesh nodes nearest to the point-constraints are snapped to them. As a result, the quad element quality, especially around the spot-welds deplete. The only work-around is to reduce the element size and create a finer mesh so as to reduce element distortion.

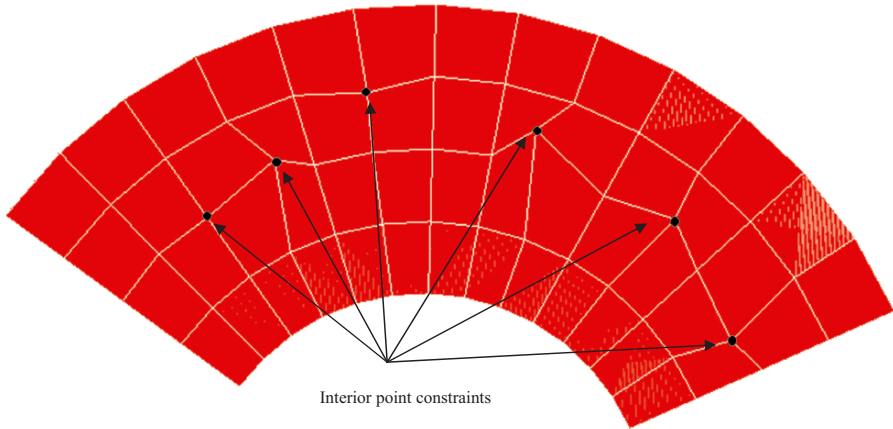


Fig. 5. Unconstrained transfinite (mapped) mesh with interior points. After the mesh is generated, nearest interior nodes are snapped to the point constraints.

Figure.6 shows an improved transfinite mesh that honors the point constraints. Although the element size is same, the interior point constraints are reflected on the boundary. Consequently, the number of elements in the t direction change (from 4 to 6). Since the number of face-interior points is less than the element count in the s direction, the final element count in the s direction does not change (13). It is interesting to note here, that although the elements produced by the algorithm in Fig. 6 are structured compared to the elements around the constraints in Fig. 5, the mesh aspect ratio becomes non-uniform. In most structural analyses, especially of such seam/spot welded body panels, the accuracy of stress computation is most sensitive to element distortion. This, distortion (D) is usually measured as a positive ratio of the minimum to the maximum Jacobian measured at the Gauss points as

$$D = | J_{\min}/J_{\max} | \quad (22)$$

The distortion D , thus, depends little on the aspect ratio of the element, as long as the element shape is rectangular. However, elements too thin (high aspect ratio) tend to have a negative impact on the assembly stiffness matrix. A 5:1 aspect ratio is usually used as a limit. Within this limit, a mesh with a better element distortion (Fig. 6) is deemed more reliable than the unstructured pattern (Fig. 5).

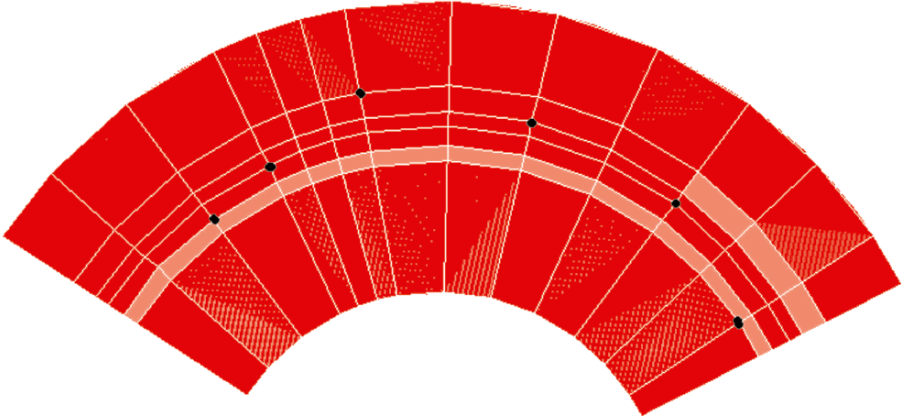


Fig. 6. Improved transfinite (mapped) mesh with interior point constraints.

The following example has 6 interior point constraints, but Fig. 6 shows only 5 soft-points in the radial (t) direction on each pseudo-edge. This is because, two soft-points per pseudo-edge in the t -direction, were merged into an average location because they were too close. As a result, those two point constraints lie on the same nodal rail-line as shown in Fig. 6. This is an example of a practical compromise that needs to be made when one or more constraints are “equi-potential”.

Figure 7 shows a quarter section of a structural bearing which is being analyzed for stress variations under dynamic loads. A swept hexahedral mesh is generated on the volume, where one of the wall faces has 3 interior point constraints. The hex mesh nodes are snapped to the point constraints. 2D transfinite meshes are first generated on all wall faces before the interior is filled. The point constraints, in this case, represent concentrated radial dynamic loads. The mesh nodes of the transfinite mesh on the wall face are snapped to the point constraints, thus resulting in bad quality hexahedral elements in the vicinity of the load application point.

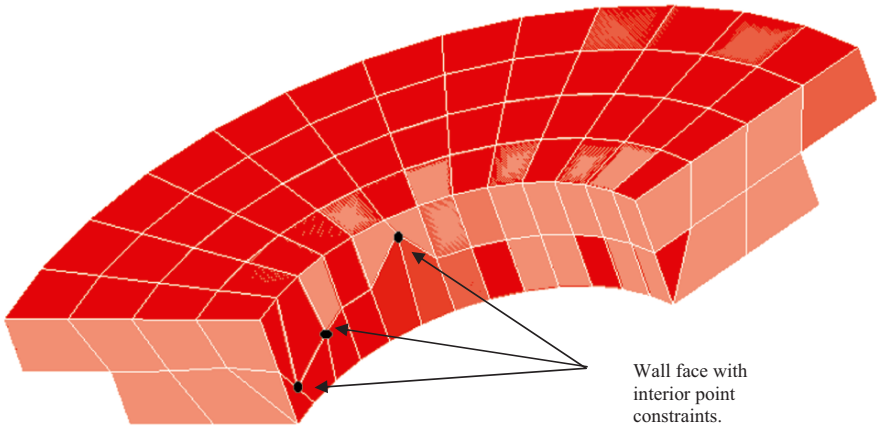


Fig. 7. Swept mesh with face interior point constraints. One wall face shows an unconstrained transfinite (mapped) mesh where the nearest interior nodes are snapped to the point constraints.

Figure 8 shows a much improved hex meshed volume, where the transfinite mesh on the wall face is perfectly structured even though it honors the point-constraints. The resulting mesh has an admirably high mesh quality compared to the mesh in Fig. 7.

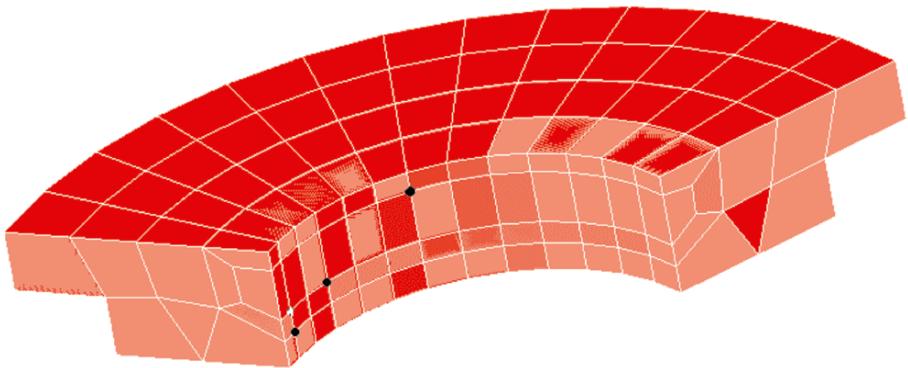


Fig. 8. Improved Swept mesh with face interior point constraints. The transfinite mesh on the wall face is immaculately structured.

9. Conclusion

Structural analyses of automotive parts and body panels frequently require high quality, high fidelity structured meshes. Many of these meshes need to honor pre-defined face-interior and boundary point constraints that represent load application points or welded or joined spots. Conventional meshing techniques snap nearest nodes to these point constraints after meshing is done thus negatively impacting the mesh quality at critical zones of interest. The present paper proposes an apriori remedial approach, where an inverse solution of Coons bi-linear blending equation is performed to determine the parametric co-ordinates of the point constraints. Once the coordinates are known, a boundary correction step is taken, where the boundary of the face is pre-seeded at these parametric locations. The number of elements to be generated along each pair of sides is also influenced by the number and location of point constraints. With the new boundary discretization, a very high quality structured mesh results as is evident from the two examples presented.

References

1. O.C., Zienkiewicz and D. V. Phillips, "An Automatic Mesh Generation Scheme For Plane and Curved Surfaces By Isoparametric Co-ordinates" *International J. Numerical Meth. Engg.* Vol 3 pp. 519-528 (1971).
2. W.J., Gordon and C.A. Hall, "Construction of curvilinear co-ordinate systems and applications to mesh generation" *International J. Numerical Meth. Engg.* Vol 7 pp. 461-477 (1973).
3. W.A., Cook, "Body oriented (natural) co-ordinates for generating three-dimensional meshes" *International J. Numerical Meth. Engg.* Vol 8 pp. 27-43 (1974).
4. Robert, Haber, Mark, S. Shepherd, John, F. Abel, Richard, H. Gallagher and Donald, P. Greenberg "A General Two-Dimensional, Graphical Finite Element Preprocessor Utilizing Discrete Transfinite Mappings" *International J. Numerical Meth. Engg.* Vol 17 pp. 1015-1044 (1981).
5. R. Pierrot, J. Vazeilles and A. Perronnet, "Une methode de generation d'un maillage 2d ou 3d a' partir d'un maillage grossier" *Annales de l'I.T.B.T.P.* Nol 372 (May 1979).
6. A. Perronnet, "Logical and physical representation of an object, modularity for the programming of f.e.m." PDE Software, Interface and Systems, Elsevier Publishers North-Holland IFIP. Soderkoping, Sweden (1984).
7. Alain, Perronnet, "Triangle, tetrahedron, pentahedron transfinite interpolations. Application to the generation of C^0 or G^1 -continuous algebraic meshes" *Proc. Int. Conf. Numerical Grid Generation in Computational Field Simulations. Greenwich, England.* Vol 7 pp. 467-476 (July6-9,1998).
8. Scott, A. Mitchell, "Choosing Corners of Rectangles for Mapped Meshing" *Proc. 13th Annual Symposium on Computational Geometry*, ACM Press, pp. 87-93, (June 1997).
9. Scott, A. Mitchell, "High Fidelity Interval Assignment" *Proceedings, 6th International Meshing Roundtable*, Sandia National Laboratories, pp. 33-44, (October 1997).

10. T.K.H.Tam and Cecil, G. Armstrong, "Finite Element mesh control by integer programming" *International J. Numerical Meth. Engg.* Vol 36, pp. 2581-2605 (1993).
11. G. Farin, *Curves and Surfaces for computer aided geometric design*. Academic Press Inc., San Diego, 1990.
12. I.D. Faux and M.J. Pratt, *Computational geometry for design and manufacture*. Ellis Horwood, Chichester, 1979.

Implementation in ALBERTA of an Automatic Tetrahedral Mesh Generator

R. Montenegro¹, J.M. Cascón², J.M. Escobar¹, E. Rodríguez¹
and G. Montero¹

¹ Institute for Intelligent Systems and Numerical Applications in Engineering,
University of Las Palmas de Gran Canaria, Campus Universitario de Tafira,
Las Palmas de G.C., Spain
`rafa@dma.ulpgc.es`, `jescobar@dsc.ulpgc.es`, `barrera@dma.ulpgc.es`,
`gustavo@dma.ulpgc.es`

² Department of Mathematics, Faculty of Sciences, University of Salamanca,
Spain
`casbar@usal.es`

This paper introduces a new automatic tetrahedral mesh generator on the adaptive finite element ALBERTA code. The procedure can be applied to 3-D domains with boundary surfaces which are projectable on faces of a cube. The generalization of the mesh generator for complex domains which can be split into cubes or hexahedra is straightforward. The domain surfaces must be given as analytical or discrete functions. Although we have worked with orthogonal and radial projections, any other one-to-one projection may be considered. The mesh generator starts from a coarse tetrahedral mesh which is automatically obtained by the subdivision of each cube into six tetrahedra. The main idea is to construct a sequence of nested meshes by refining only the tetrahedra which have a face on the cube projection faces. The virtual projection of external faces defines a triangulation on the domain boundary. The 3-D local refinement is carried out such that the approximation of domain boundary surfaces verifies a given precision. Once this objective is achieved reached, those nodes placed on the cube faces are projected on their corresponding true boundary surfaces, and inner nodes are relocated using a linear mapping. As the mesh topology is kept during node movement, poor quality or even inverted elements could appear in the resulting mesh. For this reason, a mesh optimization procedure must be applied. Finally, the efficiency of the proposed technique is shown with several applications.

1 Introduction

In finite element simulation in engineering, it is crucial to adapt automatically the three-dimensional discretization to the geometry and to the solution. Many authors have made great efforts in the past to solve this problem in different ways. A perspective on adaptive modeling and meshing can be studied in [1]. The main objective is to achieve a good approximation of the *real* solution with a minimal user intervention and a low computational cost. It is clear that as the complexity of the problem increases (domain geometry and model), the methods for approximating the solution are more complicated. ALBERTA [2, 3] is a software which can be used for solving several types of 1-D, 2-D or 3-D problems with adaptive finite elements. The local refinement and derefinement can be done by evaluating an error indicator for each element of the mesh and it is based on element bisection. To be more specific, the newest vertex bisection method is implemented for 2-D triangulations [4]. Actually, ALBERTA has implemented an efficient data structure and adaption for 3-D domains which can be decomposed into hexahedral elements as regular as possible. These elements are subdivided into tetrahedra by constructing a main diagonal and its projections on its faces for each hexahedral element. The local bisection of the resulting elements is recursively carried out by using ideas of the longest edge and the newest vertex bisection methods. Details about the local refinement technique implemented in ALBERTA for two and three dimensions can be analyzed in [5]. This strategy works very efficiently for initial meshes with a particular topology and high-quality elements (obtained by subdivision of regular quadrilateral or hexahedral elements). In these cases the degeneration of the resulting meshes after successive refinements is avoided. The restriction on the initial element shapes and mesh connectivities makes necessary to develop a particular mesh generator for ALBERTA. In this paper we summarize the main ideas introduced for this purpose. Obviously, all these techniques could be applied for generating meshes with other types of codes. Besides, these ideas could be combined with other type of local refinement algorithms for tetrahedral meshes [6, 7].

2 Automatic Mesh Generator

In this section, we present the main ideas which have been introduced in the mesh generation procedure. In section 2.1, we start with the definition of the domain and its subdivision in an initial 3-D triangulation that verifies the restrictions imposed in ALBERTA. In section 2.2, we continue with the presentation of different strategies to obtain an adapted mesh which can approximate the surface boundaries of the domain within a given precision. We construct the mesh of the domain by projecting the boundary nodes from a plane face to the true boundary surface and by relocating the inner nodes. These two steps are summarized in sections 2.3 and 2.4, respectively. Finally, in section 2.5 we present a procedure to optimize the resulting mesh.

2.1 Initial Coarse Mesh

In order to understand the idea of the proposed mesh generator, it is convenient to first consider a domain of which the boundary can be projected on the faces of a cube. A second case is to consider a parallelepiped instead of a cube. In this last case, an automatic decomposition of the parallelepiped into cubes can be carried out. At present, we have implemented in ALBERTA these two cases. Nevertheless, in the input data we could define an object outline with connected cubes and/or parallelepiped, such that the boundary of the domain is obtained by a one-to-one projection from the boundary faces of the object outline to the true boundary surface. Once the decomposition in cubes is done, we build an initial coarse tetrahedral mesh by splitting all cubes into six tetrahedra [5]. For this purpose, it is necessary to define a main diagonal on each cube and the projections on its faces, see Figure 4(a). In order to get a conforming tetrahedral mesh, all cubes are subdivided in the same way maintaining compatibility between the diagonal of their faces. The resulting initial mesh τ_1 can be introduced in ALBERTA since it verifies the imposed restrictions about topology and structure. The user can introduce in the code the number of recursive global bisections [5] which is necessary to fix a uniform element size in the whole initial mesh.

The same technique can be applied by considering a decomposition of the object outline into hexahedra instead of cubes. In this case, the recursive local refinement technique [5] introduced in ALBERTA may produce poor quality elements and, consequently, degenerate meshes. In this paper, as a first approach, we have used a decomposition of the object outline into cubes.

2.2 Local Refined Mesh

The next step of the mesh generator includes a recursive adaptive local refinement strategy of those tetrahedra with a face placed on a boundary face of the initial mesh. The refinement process is done in such a way that the true surfaces are approximated with a linear piece-wise interpolation within a given precision. That is, we look for an adaptive triangulation on the boundary faces of cubes, such that the resulting triangulation after node projection on the true boundary surface is a good approximation of this boundary surface of the domain. The user has to introduce as input data a parameter ε that defines the maximum separation allowed between the linear piece-wise interpolation and the true surface [8]. We remark that the true surface may be given by an analytical or a discrete function, such that each point of a cube face corresponds only to one point on the true surface. We propose two different strategies for reaching our objective.

The first one consists on a simple method. We construct a sequence of tetrahedral nested meshes by recursive bisection of all tetrahedra which contain a face located on a boundary face of cubes. The number of bisections is determined by the user as a function of the desired resolution of the true

surface. So, we have a uniform distribution of nodes on these cube faces; see for example Figure 4(b). Once all these nodes are *virtually* projected on the true surface, the application of the derefinement criterion developed in [8], with a given derefinement parameter ε , defines different adaptive triangulations for each face of the cube. We remark that the derefinement criterion fixes which nodes, placed on cube faces, can not be eliminated in the derefinement process in order to obtain a good approach of the true surface. Specifically, a node can not be eliminated if the distance between its virtual position on the true surface and the middle point of its *surrounding edge* is greater than ε . Then, for conformity reasons other nodes of the 3-D triangulation can not be removed. Besides, all node belonging to the coarse initial mesh continues in all the levels of the derefined sequence of nested meshes. As the derefinement criterion in ALBERTA is associated to elements, we mark for derefinement all tetrahedra containing a node which can be eliminated. In particular, we make a loop on tetrahedra during the derefinement process from the penultimate level of the sequence to the coarse initial mesh. We analyze elements with two sons and if the node, that was introduced by their father's bisection, verifies the derefinement condition, then we mark its two sons for derefinement.

The second strategy only works with a local refinement algorithm. In this case, the idea is to apply a recursive refinement on all tetrahedra containing a face placed on a boundary cube face and, at the same time, verifying that the distance between the points of the virtual triangle defined by the projection of its nodes on the true surface and the corresponding points on the true surface is greater than ε .

The first strategy is simpler, but it could lead to problems with memory requirements if the number of tetrahedra is very high before applying the derefinement algorithm. For example, this situation can occur when we have surfaces defined by a discrete function with a very high resolution. Nevertheless, the user can control the number of recursive bisections.

On the other hand, the problem of the second strategy is to determine for each tetrahedron face, placed on a boundary face of cubes, if it must be subdivided attending to the approximation of the true surface. This analysis must be done every time that a face is subdivided into its son faces. Suppose, for example that true surface is given by a discrete function. Then, the subdivision criterion stops for a particular face when all the surface discretization points on this face have been analyzed and all of them verify the approximation criterion.

2.3 Projection on Boundary Surfaces

Although ALBERTA has already implemented a node projection on a given boundary surface during the bisection process, it has two important restrictions: nodes belonging to the initial mesh are not projected, and inverted elements could appear in the projection of new nodes on complex surfaces.

In this last case, the code does not work properly, as it is only prepared to manage *valid* meshes.

For this reason, a new strategy must be developed in the mesh generator. The projection is really done only when we have defined the local refined mesh by using one of the methods proposed in the previous section. Then, the nodes placed on the cube faces are projected on their corresponding boundary surfaces, maintaining the position of the inner nodes of the domain. We have remarked that any one-to-one projection can be defined: orthogonal, spherical, cylindrical, etc.

After this process, we obtain a valid triangulation of the domain boundary, but it could appear a tangled tetrahedral mesh. Inner nodes of the domain could be located now even outside of it. So, an optimization of the mesh is necessary. Although the final optimized mesh does not depend on the inner nodes initial position, it is better for the optimization algorithm to start from a mesh with a quality as good as possible. Then we propose to relocate in a reasonable position the inner nodes of the domain before the mesh optimization.

2.4 Relocation of Inner Nodes

There would be several strategies for defining a new position for each inner node of the domain. An acceptable procedure is to modify their relative position as a function of the distance between boundary surfaces before and after their projections. This relocation is done attending to proportional criteria along the corresponding projection line. Although this node movement does not solve the tangle mesh problem, it normally makes it decrease. That is, the number of resulting inverted elements is less and the mean quality of valid elements is greater.

2.5 Mesh Optimization: Untangling and Smoothing

An efficient procedure is necessary to optimize the pre-existing mesh. This process must be able to smooth and untangle the mesh and it is crucial in the proposed mesh generator.

The most usual techniques to improve the quality of a *valid* mesh, that is, one that does not have inverted elements, are based upon local smoothing. In short, these techniques consist of finding the new positions that the mesh nodes must hold, in such a way that they optimize an objective function. Such a function is based on a certain measurement of the quality of the *local sub-mesh*, $N(v)$, formed by the set of tetrahedra connected to the *free node* v . As it is a local optimization process, we can not guarantee that the final mesh is globally optimum. Nevertheless, after repeating this process several times for all the nodes of the current mesh, quite satisfactory results can be achieved. Usually, objective functions are appropriate to improve the quality of a valid mesh, but they do not work properly when there are inverted elements. This

is because they present singularities (barriers) when any tetrahedron of $N(v)$ changes the sign of its Jacobian determinant. To avoid this problem we can proceed as Freitag et al in [9, 10], where an optimization method consisting of two stages is proposed. In the first one, the possible inverted elements are untangled by an algorithm that maximises their negative Jacobian determinants [10]; in the second, the resulting mesh from the first stage is smoothed using another objective function based on a quality metric of the tetrahedra of $N(v)$ [9]. After the untangling procedure, the mesh has a very poor quality because the technique has no motivation to create good-quality elements. As remarked in [9], it is not possible to apply a gradient-based algorithm to optimize the objective function because it is not continuous all over \mathbb{R}^3 , making it necessary to use other non-standard approaches.

We have proposed an alternative to this procedure [11], so the untangling and smoothing are carried out in the same stage. For this purpose, we use a suitable modification of the objective function such that it is regular all over \mathbb{R}^3 . When a feasible region (subset of \mathbb{R}^3 where v could be placed, being $N(v)$ a valid submesh) exists, the minima of the original and modified objective functions are very close and, when this region does not exist, the minimum of the modified objective function is located in such a way that it tends to untangle $N(v)$. The latter occurs, for example, when the fixed boundary of $N(v)$ is tangled. With this approach, we can use any standard and efficient unconstrained optimization method to find the minimum of the modified objective function, see for example [12].

In this work we have applied, for simultaneous smoothing and untangling of the mesh by moving their inner nodes, the proposed modification [11] to one objective function derived from an *algebraic mesh quality metric* studied in [13], but it would also be possible to apply it to other objective functions which have barriers like those presented in [14].

Besides, a smoothing of the boundary surface triangulation could be applied before the movement of inner nodes of the domain by using the new procedure presented in [15] and [16]. This surface triangulation smoothing technique is also based on a vertex repositioning defined by the minimization of a suitable objective function. The original problem on the surface is transformed into a two-dimensional one on the *parametric space*. In our case, the parametric space is a plane, chosen in terms of the local mesh, in such a way that this mesh can be optimally projected performing a *valid* mesh, that is, without *inverted* elements.

3 Applications

The performance of our new mesh generator is shown in the following three applications. The first corresponds to a domain defined over a complex terrain, the second to a sphere and the third to a cube with all faces deformed by Gaussian functions.

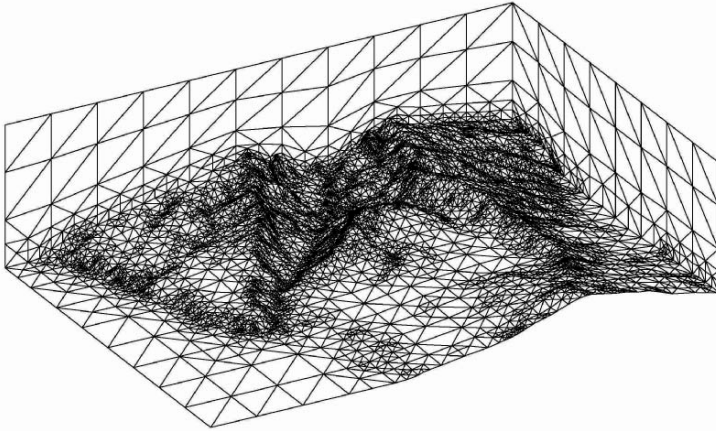
3.1 Domain Over Complex Terrain

In the last few years, we have developed a tetrahedral mesh generator that approximates the orography of complex terrains with a given precision [17, 18]. To do so, we only have digital terrain information. Our domain is limited on its lower part by the terrain and on its upper part by a horizontal plane placed at a height at which the magnitudes under study may be considered steady. The lateral walls are formed by four vertical planes. The generated mesh could be used for numerical simulation of environmental phenomena, such as wind field adjustment [19], fire propagation or atmospheric pollution [20]. The following procedures are mainly involved in this automatic mesh generation: a Delaunay triangulation method [21, 22], a 2-D refinement/derefinement algorithm [8] and a simultaneous untangling and smoothing algorithm [11]. Besides, we have recently developed a new method for quality improvement of surface triangulations, by using optimal local projections [15, 16], which can be introduced in the mesh generator.

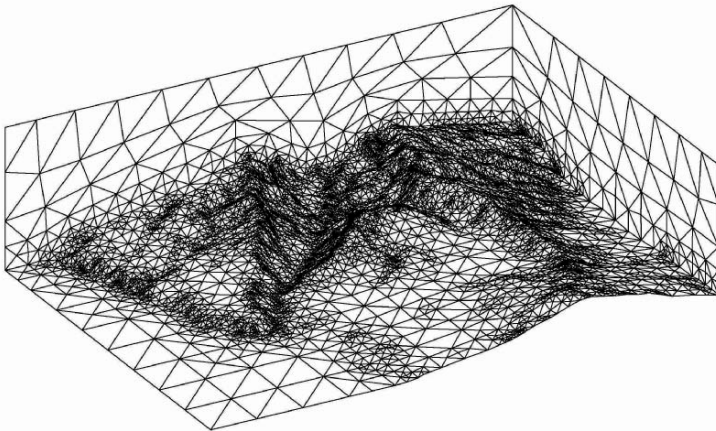
Alternatively to this strategy, the automatic mesh generator proposed in this paper can be used for the same purpose. As a practical application we have considered a rectangular area in *Isla de La Palma* (Canary Islands) of 22×16 km. The upper boundary of the domain has been placed at $h = 6$ km. To define the topography we use a digitalization of the area where heights are defined over a uniform grid with a spacing step of 200 m in directions x and y . We start from a parallelepiped of $22 \times 16 \times 6$ km initially subdivided into $11 \times 8 \times 3$ cubes with edge sizes of 2 km. Each cube is subdivided into six tetrahedra by using the subdivision proposed in [5], see Figure 4(a). This discretization is used to define the uniform initial triangulation τ_1 of the parallelepiped. We refine it 18 times by constructing a recursive bisection of all tetrahedra which contain a face placed on the lower face of the parallelepiped. If we applied 6 global refinements by using the 4-T Rivara's algorithm [23] instead of previous recursive bisections, the resultant 2-D triangulation on the lower face of the parallelepiped would be the same.

Once the orography is virtually interpolated on this local refined mesh, the derefinement condition, introduced in [8], is applied with a derefinement parameter of $\varepsilon = 25$ m. Then, we make an orthogonal projection on the terrain of the adaptive triangulation obtained on the lower face of the parallelepiped. Besides, we relocate the other nodes vertically by using a proportional criterion. The adapted mesh has 65370 tetrahedra and 15263 nodes, see Figure 1(a), and it nears the terrain surface with an error less than $\varepsilon = 25$ m.

This mesh has 115 inverted tetrahedra, its average quality measure is $\bar{q}_\kappa = 0.68$ and its minimum quality is 0.091, see reference [11] and Figure 2. The node distribution is hardly modified after five steps of the optimization process by using our modified objective function. We remark that we have not relocated those nodes placed on the terrain during this optimization process.



(a)



(b)

Fig. 1. Detail of *Isla de La Palma* (Canary Island): (a) initial mesh and (b) resulting mesh after five steps of the optimization process

The evolution of the mesh quality during the optimization process is represented in Figure 2. This measure tends to stagnate quickly. The quality curves corresponding to the second and fifth optimization steps are close. The average quality measure increases to $\bar{q}_\kappa = 0.75$. After this optimization process, the worst quality measure of the optimized mesh tetrahedra is 0.34. Finally, we remark that the number of parameters necessary to define the resulting mesh is quite low, as well as the computational cost. The total CPU time for the initial mesh and its optimization is less than 1 minute on an Intel Pentium M processor, 2.26 GHz and 2 Gb RAM memory. In particular, the computational cost of five iterations of the simultaneous untangling and smoothing

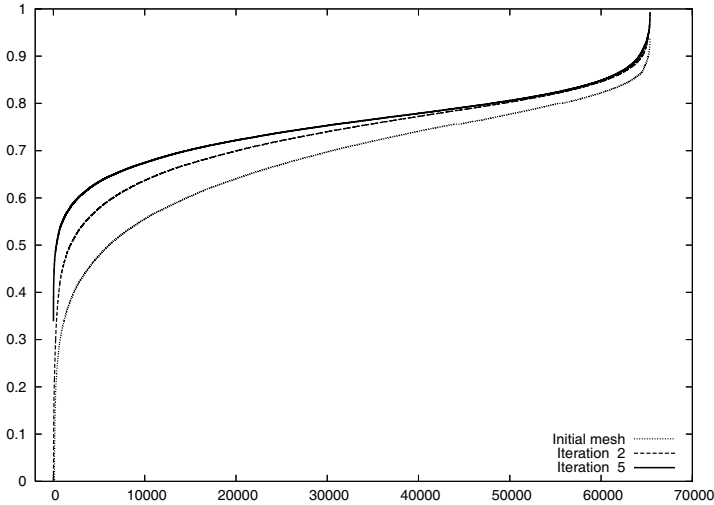


Fig. 2. Quality curves for the initial and optimized meshes after two and five iterations for the domain defined in *Isla de La Palma* (Canary Island)

procedure is about half a minute. At the first iteration of this optimization process the mesh is untangled.

3.2 Sphere

We consider now a 3-D spherical domain as a first application where all cube faces are radial projected on a curve surface. We start from a cube divided into six tetrahedra, see Figure 4(a), and we refine 9 times constructing a recursive bisection of all tetrahedra which contain a face placed on a face of the cube. The resulting mesh contains 577 nodes and 2016 tetrahedra. Then, we make a radial projection on the spherical surface of triangulations defined on the cube faces. Besides, we relocate the inner nodes radially by using a proportional criterion. A view of the resulting mesh can be seen in Figure 3(a).

No inverted elements appear in this process and high quality elements are produced. Its average quality measure is $\bar{q}_\kappa = 0.71$ and its minimum quality is 0.48. If we use the tetrahedral mesh optimization presented in [11] by only relocating inner nodes of the domain, the mesh quality is improved with a minimum value of 0.55 and an average $\bar{q}_\kappa = 0.73$. We remark that the improvement is not so significant after ten iterations, since the initial mesh has good quality. The CPU time for constructing the initial mesh is approximately 0.2 seconds and for its smoothing process is 0.5 seconds on a Intel Pentium M processor, 2.26 GHz and 2 Gb RAM memory.

In this application we also show the smoothing possibility of the surface triangulation. So, we use the technique proposed in [15, 16] to improve the quality of the spherical surface triangulation presented in Figure 3(a). This

surface mesh contains 386 nodes and 640 triangles with an average quality of 0.85 and a minimum one of 0.65. After five iterations of surface smoothing process we obtain the mesh shown in Figure 3(b) which has an average quality of surface triangles of 0.86 and a minimum one of 0.78. If we keep the boundary nodes in their new positions and we optimize again the mesh by only moving inner nodes, then the tetrahedral mesh results with an average quality measure $\bar{q}_\kappa = 0.72$ and its minimum quality is 0.52. It can be observed that these values are between those obtained for the initial mesh and its optimization. Nevertheless, the difference is not significant due to the regularity of the initial mesh. The procedure proposed in this paragraph could be interesting when we start from meshes with poor quality surface triangulations.

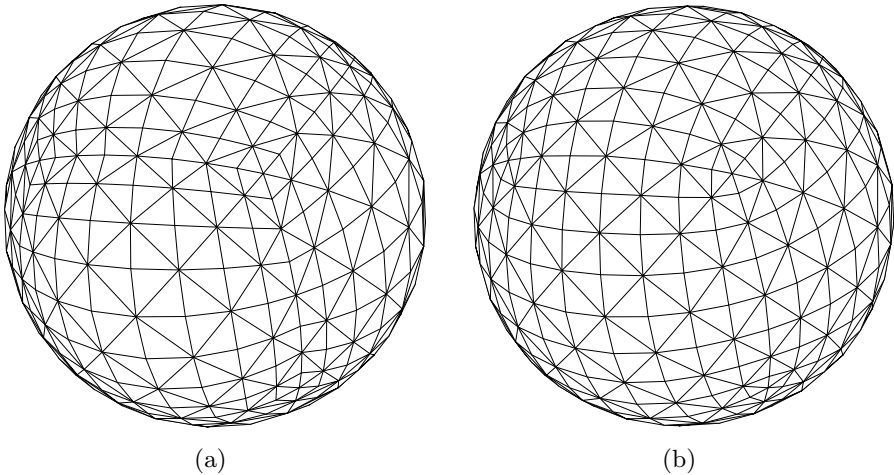


Fig. 3. Surface triangulation of the sphere: (a) initial mesh and (b) resulting mesh after five steps of the surface optimization process

3.3 Domain with Gaussian Surfaces

As a last application we present a unit cube domain of which faces are orthogonal projected on different surfaces defined by Gaussian functions. We start from an initial coarse triangulation composed by 8 nodes and 6 tetrahedra. This subdivision is proposed in [5], see Figure 4(a). We refine this triangulation τ_1 by 18 recursive bisections of all the tetrahedra which contain a face placed on a cube face, resulting a mesh with 48703 nodes and 198672 tetrahedra, see Figure 4(b). Once the boundary surface information was virtually interpolated on this local refined mesh, the derefinement condition [8] was applied with a derefinement parameter $\varepsilon = 0.00001$.

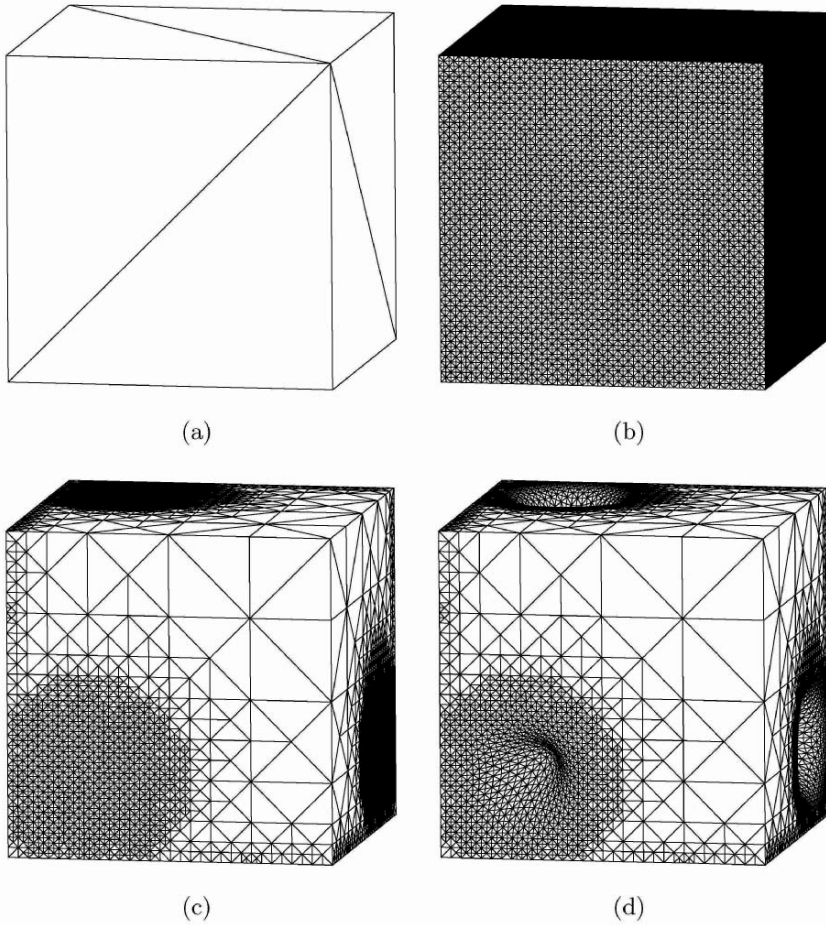


Fig. 4. Main stages of the mesh generator for the domain with Gaussian surfaces on all its faces: (a) initial mesh, (b) global face refinement, (c) local face refinement after applying the derefinement procedure and (d) projection on boundary surfaces

In Figure 4(c) the corresponding mesh is presented and it contains 23520 nodes and 60672 tetrahedra. Then, we make an orthogonal projection on the domain boundary surface of the adaptive triangulation obtained previously over the cube faces. Besides, we relocate the inner nodes by using a proportional criterion along the Cartesian directions. The resulting mesh is shown in Figure 4(d) and it initially has 240 inverted tetrahedra with average quality measure $\bar{q}_\kappa = 0.64$, see Figure 6. An inner view of the surface triangulation may be observed in Figure 5. The evolution of the mesh quality during the optimization process, by applying the simultaneous untangling and smoothing procedure [11] to the inner nodes of the domain, is represented in Figure 6. The quality curves corresponding to the second and tenth optimization steps

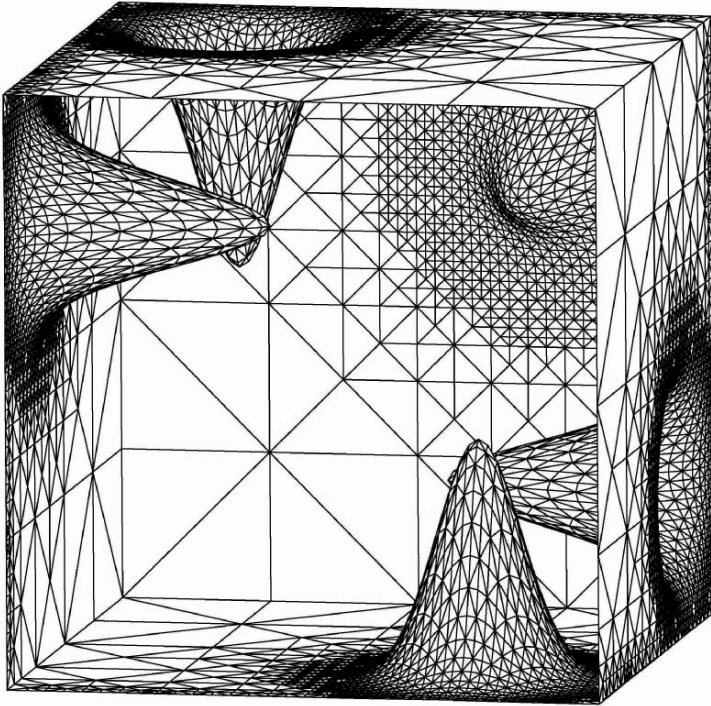


Fig. 5. Inner view of the surface triangulation for the domain with gaussian surfaces

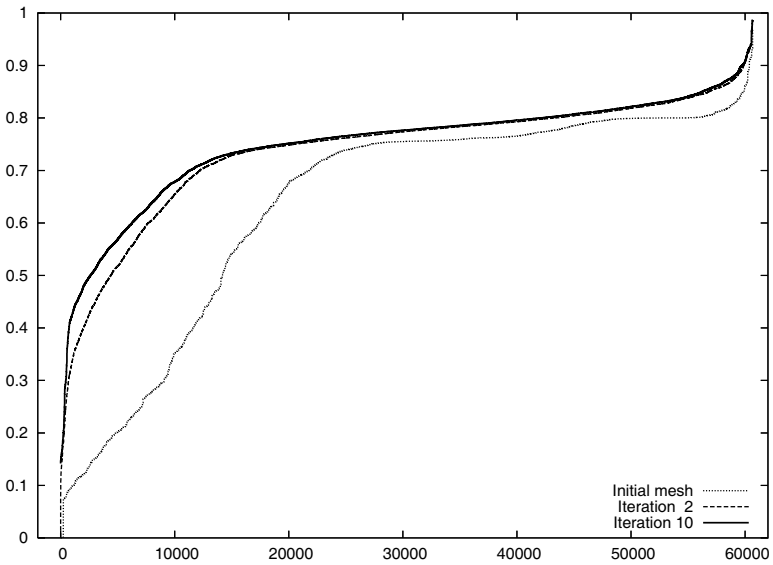


Fig. 6. Quality curves for the initial and optimized meshes after two and ten iterations for the domain with gaussian surfaces

are very close. The average quality measure increases to $\bar{q}_\kappa = 0.75$ and the minimum value improves to 0.14. Finally, we remark that the final mesh is generated in less than one minute on a Intel Pentium M processor, 2.26 GHz and 2 Gb RAM memory. In particular, the computational cost of five iterations of the simultaneous untangling and smoothing procedure is about half a minute. We observe that at the second iteration of this optimization process the mesh is untangled.

4 Conclusions and Future Research

The proposed mesh generator is an efficient method for creating tetrahedral meshes on domains with boundary faces projectable on faces of cubes and it is used as pre-processor for ALBERTA. We remark that it requires a minimum user intervention and has a low computational cost. The main ideas presented in this paper for automatic mesh generation could be used for different codes which work with other tetrahedral or hexahedral local refinement/derefinement algorithms. With these ideas, more complex domains could be meshed by decomposing its *outline* into a set of connected cubes or hexahedra. Although this procedure is at present limited in applicability for high complex geometries, it results in a very efficient approach for the problems that fall within the mentioned class. The mesh generation technique is based on sub-processes (subdivision, projection, optimization) which are not in themselves new, but the overall integration using a simple shape as starting point is an original contribution of this paper and it has some obvious performance advantages.

Acknowledgments

This work has been supported by the Spanish Government and FEDER, grant contracts: CGL2004-06171-C03-03/CLI and CGL2004-06171-C03-02/CLI (see <http://www.dca.iusiani.ulpgc.es/proyecto0507>). Besides, we thanks to the authors of ALBERTA [3] for the code availability in internet [2] and for their suggestions.

References

1. Carey GF (2006) *Comp Meth Appl Mech Eng* 195:214–235
2. ALBERTA – An Adaptive Hierarchical Finite Element Toolbox, <http://www.alberta-fem.de/>
3. Schmidt A, Siebert KG (2005) *Design of Adaptive Finite Element Software: The Finite Element Toolbox ALBERTA*. Lect N Comp Sci Eng 42. Springer, Berlin Heidelberg New York

4. Mitchell WF (1989) *ACM Trans Math Soft* 15:326–347
5. Kossaczky I (1994) *J Comp Appl Math* 55:275–288
6. Löhner R, Baum JD (1992) *Int J Num Meth Fluids* 14:1407–1419
7. González-Yuste JM, Montenegro R, Escobar JM, Montero G, Rodríguez E (2004) *Adv Eng Soft* 35:693–702
8. Ferragut L, Montenegro R, Plaza A (1994) *Comm Num Meth Eng* 10:403–412
9. Freitag LA, Knupp PM (2002) *Int J Num Meth Eng* 53:1377–1391
10. Freitag LA, Plassmann P (2000) *Int J Num Meth Eng* 49:109–125
11. Escobar JM, Rodríguez E, Montenegro R, Montero G, González-Yuste JM (2003) *Comp Meth Appl Mech Eng* 192:2775–2787
12. Bazaraa MS, Sherali HD, Shetty CM (1993) *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, Inc. New York.
13. Knupp PM (2001) *SIAM J Sci Comp* 23:193–218
14. Knupp PM (2000) *Int J Num Meth Eng* 48:1165–1185
15. Escobar JM, Montero G, Montenegro R, Rodríguez E (2006) *Int J Num Meth Eng* 66:740–760
16. Montenegro R, Escobar JM, Montero G, Rodríguez E (2005) Quality improvement of surface triangulations. In: Hanks BW (ed) *Proceedings of the 14th International Meshing Roundtable*, 469–484. Springer, Berlin Heidelberg New York
17. Montenegro R, Montero G, Escobar JM, Rodríguez E, González-Yuste JM (2002) *Lect N Comp Sci* 2329:335–344
18. Montenegro R, Montero G, Escobar JM, Rodríguez E (2002) *Neural, Parallel & Scientific Computation* 10:57–76
19. Montero G, Rodríguez E, Montenegro R, Escobar JM, González-Yuste JM (2005) *Adv Eng Soft* 36:3–10
20. Montero G, Montenegro R, Escobar JM, Rodríguez E, González-Yuste JM (2004) *Lect N Comp Sci* 3037:642–645
21. George PL, Hecht F, Saltel E (1991) *Comp Meth Appl Mech Eng* 92:269–288
22. Escobar JM, Montenegro R (1996) *Adv Eng Soft* 27:27–39
23. Rivara MC (1987) *Int J Num Meth Eng* 24:1343–1354

Sparse Voronoi Refinement^{*}

Benoît Hudson, Gary Miller, and Todd Phillips

Computer Science Department
Carnegie Mellon University

Summary. We present a new algorithm, Sparse Voronoi Refinement, that produces a conformal Delaunay mesh in arbitrary dimension with guaranteed mesh size and quality. Our algorithm runs in output-sensitive time $O(n \log(L/s) + m)$, with constants depending only on dimension and on prescribed element shape quality bounds. For a large class of inputs, including integer coordinates, this matches the optimal time bound of $\Theta(n \log n + m)$. Our new technique uses interleaving: we maintain a sparse mesh as we mix the recovery of input features with the addition of Steiner vertices for quality improvement.

1 Introduction

One of the main missing components in current tetrahedral meshing algorithms research is the existence of refinement algorithms with good run time analysis. Runtime analysis for some methods (notably those involving quad-trees or octrees) has been straightforward [HPU05, MV00] due to the very structured spatial decomposition. However, there are many practical meshing algorithms with very poor run time guarantees.

The goal in designing Sparse Voronoi Refinement (SVR) was to create a meshing algorithm that was similar in implementation and style to many widely used meshing algorithms, but with the added benefit of very strong worst-case bounds on the runtime complexity and space usage. An additional achievement of SVR is that the algorithm can work in any fixed dimension d . The most practical implementations of SVR will probably take advantage of fixing $d = 3$, but higher dimensional meshing (spacetime methods, etc.) is a growing area of future research to which SVR can contribute.

The three important aspects of a mesh that are addressed by SVR are that the mesh resolve all the input features (conforming), that mesh elements

^{*} This work was supported in part by the National Science Foundation under grants CCR-9902091, CCR-9706572, ACI 0086093, CCR-0085982 and CCR-0122581.

be well-shaped (quality), and that the number of elements be small (size-guarantee).

Quality: In this work, we will refer to a **quality** mesh as one in which the radius-edge ratio is bounded by a constant for any mesh element. In three dimensions, this metric is somewhat lacking, as it can admit a family of poorly-shaped elements known as *slivers*, although it is known how to post-process a radius-edge mesh and eliminate slivers. On the other hand, this criterion allows better proof generalization and yields strong bounds on the aspect ratio of the *Voronoi Diagram*, the dual of the Delaunay triangulation. Thus most of the SVR operations and metrics are defined on the Voronoi diagram. This vastly simplifies most of the proofs, especially for $d \geq 3$. An actual implementation of SVR would likely only use the Delaunay, but the conceptual framework of the Voronoi is essential to understanding the algorithmic design of SVR. We further discuss mesh quality, Voronoi quality, and the elimination of slivers in Section 2.1.

Conforming: Sparse Voronoi Refinement produces a Voronoi diagram such that the dual Delaunay triangulation *conforms* to the input. That is, every input feature is represented in the output Delaunay mesh by one or several segments, triangles, and higher-dimensional facets.

Size Guarantee: We would like to say that our algorithm generates a mesh whose size is at most a constant fraction larger than the smallest radius-edge quality mesh. There are two problems with this goal. First, finding such a mesh seems difficult. Two, it is not obvious that this is the correct goal for meshing applications. To see the second concern consider an input of just two edges of the same length forming a cross. Assume the edges do not intersect but come arbitrarily close. An optimal radius-edge mesh would simply form a sliver out of these two edges. While, a good aspect-ratio mesh would introduce very small tetrahedra near where the two edges are close. Thus, it seems that we would like to return a mesh such that the local feature size at every point is bound both from above and below by a constant times the diameter of the simplex containing it. In this paper we will only show that our algorithm will return a mesh such that no tetrahedra is too small. That the distance between any two vertices is bounded below by the local feature size of the input.

Time and Space Usage: Given a Piecewise Linear Complex (PLC) as input [MTTW95], we will use n to denote the total number of input features (vertices, segments, triangles and larger facets, etc). We will use L/s to denote the the ratio of the diameter of the PLC to the smallest pairwise distance between two disjoint features of the PLC. The notation L/s is historic, and the concept appears in many works under many names ([Eri01] contains a long list of references).

Sparse Voronoi Refinement has worst case runtime bounded by $O(n \log L/s + m)$, where m is the number of output vertices. This runtime bound is a vast improvement over prior meshing algorithms for three and higher dimensions. For almost all interesting inputs, this bound is equivalent to $O(n \log n + m)$,

which is optimal (using a sorting lower bound). SVR also has optimal output-sensitive space usage $O(m)$.

Most versions of Delaunay refinement algorithms (Section 2 contains a short survey of such methods) first construct the Delaunay triangulation of the input points as a preprocess. Unfortunately, in dimension higher than two, the complexity of the Delaunay triangulation (that is, the number of edges, triangles, tetrahedra, *etc.*) can be super-linear in the number of input vertices: $\Omega(n^{\lceil d/2 \rceil})$ in the worst case. More concretely, in three dimensions this implies that any such algorithm has worst case space and runtime $\Omega(n^2)$, which is impractical for large inputs – furthermore, some of the classic worst-case examples look very much like engineered surfaces one might want to mesh.

However, it is well known that a quality Delaunay mesh with m vertices has only $O(m)$ elements [MTTW95, Tal97], implying that it should be possible to break the $O(n^{\lceil d/2 \rceil})$ runtime barrier by avoiding the preprocessing step.

Sparse Voronoi Refinement accomplishes this by interleaving the traditional preprocess with the main body of the algorithm. We maintain a Delaunay mesh throughout the algorithm, but enforce that it is *always* a quality radius-edge mesh. The input point set and features are gradually recovered over the life of the algorithm, *rather than at initialization*.

By maintaining the quality of the mesh, SVR ensures that it is *sparse*: the degree of any vertex is at most constant. This allows all the mesh modifications to be performed in constant time, so that the runtime work for the refinement process is only $O(m)$ (Section 8.1), plus bookkeeping of $O(n \log(L/s))$ term arising from point location costs (Section 8.2).

SVR expands the capabilities of existing analyzed meshing algorithms in terms of practical output size and generalization to higher dimension. SVR also represents the first analyzed 3D Delaunay refinement algorithm to achieve the near-optimal bounds we claim. Nonetheless, SVR does not yet completely solve the meshing problem because it has overly strict restrictions on the geometry of the input. Future work in this regard is discussed in Section 9.

Section 2 discusses some related work, along with some of the design decisions of SVR.

Some notation needed to understand the algorithms, proofs, and input restrictions is presented in Section 3.

Section 4 presents a simplified, toy version of SVR that operates on an input point set without input features or boundary concerns. Important new algorithmic and proof techniques are discussed in this section. The goal is to present the salient novel features of SVR, so that the reader can better understand the general method behind SVR and the geometric intuitions behind the proofs.

Section 5 presents the full SVR algorithm, with the added complications for handling boundaries and input features. The approaches taken by SVR to address these complexities are highly similar to previous works [MPW02]. For brevity we refer to earlier results for most of the proofs and only highlight the

novel portions; a technical report version of this paper presents the proofs in full detail [HMP06].

The results of Section 6 in particular provide fairly strong structural results about quality Voronoi diagrams that may be applicable to other meshing algorithm analyses; Sections 7 and 8 use the structural results to prove our algorithm is correct, size-guaranteed, and fast.

2 Related Work

2.1 Mesh Quality and Well-Spaced Points

It has long been known that the Finite Element Method converges to a solution if there are no large (almost 180°) angles in the mesh. The closely related problem of ensuring there are no small (almost 0°) angles in the mesh has proven to be more tractable using the Delaunay triangulation, which has well-understood geometric and topological properties. In general, the smallest angle in the Delaunay triangulation can be still very small, though it is maximal over all possible triangulations; therefore, to ensure a quality mesh, we must add Steiner vertices. When all the angles in a simplex are large, the simplex has good **aspect ratio**, which is variously described as the ratio between the radius (or volume) of the minimum circumscribing ball to the maximum inscribed ball, or the ratio between the volume of the simplex to the length of its shortest edge.

In generalizing to higher dimension, it is much more convenient to use a different definition of mesh quality than the aspect ratio. The **radius-edge ratio** of a simplex is the ratio of the circumradius to the length of the smallest edge of the simplex, where the circumradius of a simplex is the radius of the d -dimensional circumball that passes through its vertices. It was observed more than ten years ago that meshes with good radius-edge have the property that the points from the mesh are well spaced in a precise technical sense. In particular, the Voronoi diagram of the point set has the property that each Voronoi polytope has good aspect ratio (since the Voronoi region may be unbounded one has to carefully define its aspect ratio; see Definition 1). This work has motivated research into efficient algorithms to generate these good radius-edges meshes. Indeed, most methods related to Ruppert's Delaunay refinement algorithm generate meshes that explicitly satisfy the radius-edge condition rather than the aspect ratio condition.

In two dimensions, the radius-edge ratio corresponds exactly with the aspect ratio. In three and higher dimension, this is not true: there are simplices with good (small, near one) radius-edge ratio that have bad (large, near infinite) aspect ratio. Because of their shape in 3D, these types of elements are known as **slivers**. Many algorithms have been proposed that take as input a good radius-edge 3D mesh, and refine it into a sliver-free, good aspect-ratio mesh [Che97, ELM⁺00, LT01].

In particular, using the results and techniques of our analysis, a very simple timing analysis of the Li-Teng algorithm for sliver removal ([LT01], extended by Li to higher dimension [Li03]) shows that it will work with linear time and space requirements on point sets. Such an algorithm can easily be run as a post-process to SVR in order to generate a sliver-free mesh. Therefore, we henceforth ignore the issue of slivers.

2.2 Delaunay Refinement Algorithms

There have been several different approaches to the meshing problem. The idea of generating a mesh whose size is within a constant factor of optimal was first considered by Bern, Epstein, and Gilbert [BEG94] using a quadtree approach. A 3D extension was given by Mitchel and Vavasis [Mit93].

Chew introduced a 2D Delaunay refinement algorithm [Che89] and showed termination. Chew added the circumcenter of poor quality triangles (Steiner nodes).

Ruppert [Rup95] extended this idea of adding circum-centers for 2D meshing to produce a mesh that was within a constant factor in size from the optimal and also handled line segments as input features. The extension of this algorithm to 3D has been ongoing research. Some methods assume that that Ruppert's local feature size function is given [MTT⁺96]. Others refine a bad aspect ratio mesh directly [She98, MPW02]. These methods by themselves do not give constant approximation size meshes since they may include slivers. But they do produce meshes that have size bounded below by a constant times the local feature size.

Finding refinement algorithms that have provably good run times has also been of interest. Spielman, Teng, and Üngör [STÜ02] proved that Ruppert's and Shewchuk's algorithms can be made to run in $O(\lg^2 L/s)$ parallel steps. They did not, however, prove a work bound. Miller [Mil04] provided the first sub-quadratic time bound in 2D with a sequential work bound of $O((n \lg \Gamma + m) \lg m)$, in which Γ is a localized version of L/s (in particular, $\Gamma \leq L/s$). The extra $\lg m$ factor is related to a priority queue that was required for the runtime proof in the presence of input segments. Har-Peled and Ungor removed the $\lg m$ factor using a quadtree data structure [HPU05].

Many algorithms for Delaunay refinement in 3D have been proposed [She02, CP03, PW04, CD03], but so far have eluded nontrivial runtime analysis. Simple examples can usually give bad worst-case performance for naive implementations of these algorithms. As mentioned, they will all suffer from intermediate size $\Omega(n^2)$ in the worst case.

3 Preliminaries

Throughout this paper, unless explicitly stated otherwise, all objects are in \mathbb{E}^d . Given a set S , the **convex closure** of S , denoted $CC(S)$, is the smallest convex set containing S , while the **convex hull** is the boundary of $CC(S)$.

Throughout, all **balls** will refer to topologically open balls. We say that a point x **encroaches** on a ball B if $x \in B$; that is, if x is interior to the ball.

A *polytope* is the convex combination of finite set of points P . The dimension of the polytope is the dimension of the affine subspace generated by P . The boundaries as well as the domain we consider are a collection of polytopes. We formally use the formal definition of a Piecewise Linear Complex (PLC) from [MTTW95].

For this paper, we place additional requirements on the input; we expect that most if not all of these can be lifted with additional work drawing on existing techniques. First, we make the usual Draconian requirement that the angle between any two intersecting polytopes, when one is not contained in the other, is at least 90° . We also require that polytopes have a convex hull that is defined by $O(1)$ vertices (although we place no restriction on the number of interior vertices); and furthermore that the hull vertices are well-spaced. Finally, the definition of a PLC requires that all input facets are convex. This would be a minor restriction were input angles unrestricted, because one can always decompose the PLC facets as needed to fulfill this condition. In the Future Work Section 9 we discuss how to lift some of these restrictions.

We define a mesh M as a set of vertices in \mathbb{E}^d and the Voronoi diagram of the vertices. The Voronoi cell of a mesh vertex v is denoted $V_M(v)$. We will call the set of nodes of the Voronoi diagram the Voronoi nodes (these are *not* the vertices of M). We define the **outradius** $R_M(v)$ to be the distance from v to the farthest Voronoi node of its cell. We define the **inradius** $r_M(v)$ to be the distance from v to the point of closest approach of the boundary of the Voronoi cell. When the mesh in question is clear, we may write R_v or r_v for brevity.

Given a set of points P , let $CC(P)$ denote the convex closure of P , i.e., the smallest convex set containing P .

Definition 1. *Given a mesh M and a mesh vertex v , the **aspect ratio** of the Voronoi cell $V_M(v)$ is $R_M(v)/r_M(v)$.*

We say M has aspect ratio τ if for each vertex $v \in \text{vertices}(M)$ the aspect ratio of $V(p)$ is at most τ . In this case, we will refer to M as a τ -quality Voronoi diagram.

We will assume throughout there are no degeneracies, that is, no sphere of dimension $k \leq d$ containing $k + 2$ points. Algorithms to address degeneracies have been considered; incorporating these method into our framework will be addressed in later work following Miller, Pav, and Walkington [MPW02].

A crucial definition is that of **local feature size**, as defined by Rupert [Rup95]. Let $\text{lfs}(x)$ be the minimum distance from x to two disjoint polytopes of the input PLC. We also define a closely related function, the **current feature size** $\text{cfs}_M(x)$, which is the distance from x to the second-nearest mesh vertex of M . We will simply write cfs when it is clear that only one mesh M is involved. This notion of cfs has been written elsewhere as lfs_M , lfs_0 , or $\text{lfs}_{0,M}$, however, in this work we will use cfs to strongly disambiguate

the two notions. The lfs never changes and counts distance to any polytope. The cfs decreases between intermediate meshes as refinement proceeds, and only counts distance to vertices.

Given any point p in the convex closure of G we consider the lowest dimensional cell containing p . We call this the **containing dimension of p** , denoted $\text{CD}(p)$.

We will need a spacing function defined everywhere for our runtime bounds. We use the *gap size* definition originally defined for mesh coarsening [MTT99, Tal97].

Definition 2. *Let P be a point set in \mathbb{E}^d . A **gap-ball** B of P is any d -ball meeting the following two criteria:*

- B is not encroached by any point in P .
- The center of B lies inside the convex closure $CC(P)$.

Definition 3. *Let P be a point set in \mathbb{E}^d . Let x be a point in $CC(P)$. The **gap size** $G_P(x)$ is the radius of the largest gap-ball B of P such that x lies on the surface of B .*

For brevity, we define that $G_M(x) \equiv G_{\text{vertices}(M)}(x)$.

Clearly, the gap size is a monotone decreasing function as we add vertices to the mesh: the shape of the convex closure does not change, but it gets harder to satisfy the non-encroachment requirement.

Definition 4. *Let a mesh M and a point $x \in CC(M)$ be given. We define the **grading** of M at x as:*

$$\Gamma_M(x) = \frac{G_M(x)}{\text{cfs}_M(x)}$$

This notion of grading is useful for capturing the relative quality of a mesh, with the advantage that Γ is defined everywhere in the convex closure, rather than just at vertices like many mesh metrics. This notion and nearly-equivalent notions for grading have used before [MTT99, Tal97, Mil04, HPU05].

4 Simplified Algorithm

In this section, we describe a simplified version of our algorithm without any input feature complications or boundary concerns. For the simple algorithm, the input is a set of points in the infinite plane \mathbb{E}^d where the hypercube $[0, L^{1/d}]^d$ repeats. While this is not a particularly realistic model of computation, it does allow us to develop the intuition we use for the full algorithm, while avoiding considerable distractions. Furthermore, the periodic point set

SIMPLIFIED-SVR(P : d -dim point set, τ : mesh quality constant, k : $0 < k \leq 1$)

- 1: Assume an initial Voronoi mesh M exists.
- 2: Let U be the set of uninserted input vertices: $U = P - V(M)$
- 3: **while** U is non-empty **do**
- 4: Perform a *break* move
- 5: **while** M has aspect ratio worse than τ **do**
- 6: Perform a *clean*
- 7: **end while**
- 8: **end while**

TRYINSERT(p : d -dim point, M d -dim mesh, U : set of uninserted d -dim points)

- 9: **if** $\exists u \in U$ s.t. $|pu| \leq kNN_M(p)$ **then**
- 10: *Yield*: Insert u into M , updating U and the Voronoi diagram.
- 11: **else**
- 12: Insert p into M , updating the Voronoi diagram.
- 13: **end if**

Fig. 1. The simplified SVR algorithm on a periodic point set. The break and clean moves are described in the text.

model can be simulated by embedding the input hypercube within a larger bounding box.

For the purposes of this simplified exposition, we will not discuss initialization, except to note that it is only linear work, assigning uninserted vertices to one of a constant number Voronoi cells. At a basic level, SVR operates incrementally. At any point in time, we have a Voronoi diagram. Some of the input points are Voronoi vertices, some of the input points are not yet recovered. The Voronoi diagram also has Steiner vertices. Every input point that is not yet recovered is contained in some Voronoi cell.

The algorithm iteratively plays one of three moves until none of the moves apply:

- *clean* move: Pick a Voronoi cell of bad aspect ratio. Take one of its farthest Voronoi nodes, and try to insert it using TRYINSERT.
- *break* move on an input: Pick an input vertex that has been recovered in the mesh and whose Voronoi cell includes an input point. Take its farthest Voronoi node, and try to insert it using TRYINSERT.
- *break* move on a Steiner vertex: Pick a mesh vertex that is not an input (it is a Steiner point) and whose Voronoi cell includes an input point. Unconditionally insert one of the input points.

Assuming this algorithm terminates, at that point there will be no cells of bad aspect ratio, and all the input vertices will have been recovered, so the algorithm will have produced a quality conforming mesh.

We add the additional constraint that clean moves take precedence over break moves. One can then think of the algorithm as operating in phases. Each phase, SVR starts with a quality mesh, “breaks” it with a break move; then “cleans” using only clean moves, until quality is re-established.

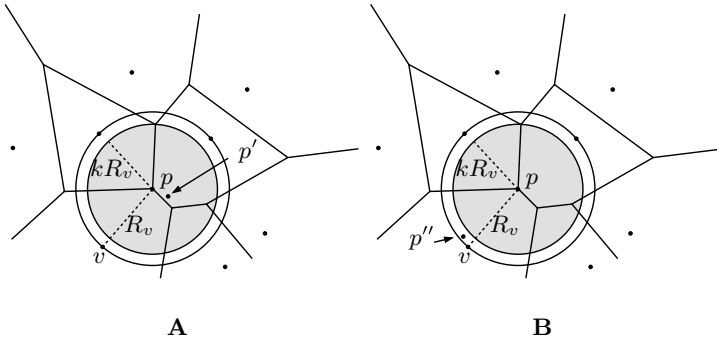


Fig. 2. The cell of v is being destroyed by SVR (either because it is skinny, or because p'' is inside); SVR tries to add the farthest Voronoi node p . The outer circle around p is of radius R_v ; the inner disc, in grey, is of radius kR_v . **(A)** Because the uninserted input point p' is near p , SVR must yield – otherwise, an artificially small feature will have been created, violating our size guarantees. **(B)** The uninserted input point p'' is too far from p , so SVR does not yield. Our size guarantees would be maintained were SVR to yield to p'' , but the time bounds would be violated because p'' may be arbitrarily close to v .

When the algorithm considers adding a Steiner point, it may instead *yield* to an input vertex if there is one nearby. This ensures that no Steiner point ever is inserted too close to an input vertex, which is critical for guaranteed mesh size. Suppose SVR is cleaning a skinny Voronoi cell $V(v)$ and considers the addition of Steiner p . There is an empty ball around p of radius R_v . This provides a natural definition of “nearby,” for yielding purposes. For reasons related to runtime analysis, we must reduce the radius of this neighborhood by a constant factor k , slightly smaller than 1. Expressed concisely, we will yield to an unrecovered input point p' if $|pp'| < kR_v$ (see Figure 2).

Why does the runtime argument need $k < 1$? If we relaxed to $k = 1$, p' might be arbitrarily close to v (or some other vertex). Such a situation would cause an arbitrarily bad break in the quality of the mesh. This would cause us to lose sparsity guarantees, which would in turn destroy runtime guarantees. For this reason, the constants in the runtime analysis will depend on $1/(1-k)$, greatly increasing as k approaches 1. On the other hand, guarantees about spacing and mesh size work best when we are most aggressive about yielding to input vertices rather than adding more Steiner vertices: smaller k makes for larger output. There is a tradeoff here, which we expect argues for setting k close to 1.

The proof that the simplified SVR terminates with guaranteed mesh size is almost verbatim from Ruppert [Rup95], albeit adapted to higher dimension and with an additional case for yielding to input vertices. What is novel are the proofs that bound the runtime and intermediate mesh quality.

Under appropriate settings of the user-given parameters τ and k (namely, $k\tau > 2\sqrt{2}$), we can guarantee that M never has aspect ratio worse than some constant τ' that depends only on τ , k , and d – even during the clean and break phases. It is well known [MTTW95, Tal97] that if the mesh vertices are well-spaced, then the Delaunay and its dual Voronoi have size linear in the number of vertices. In particular, every vertex has constant degree (its Voronoi cell has a constant number of facets). This further implies that inserting a vertex into such a mesh will take only constant time using an incremental algorithm; this gives us the $O(m)$ bound for refinement work.

The only charge left, then, is the point location: determining whether any unrecovered vertices are included in the k -ball around p . The point location structure we use is almost trivial: the algorithm simply stores the list of unrecovered input vertices that each cell includes. When a new mesh vertex is inserted, the insertion algorithm also recomputes the lists of affected Voronoi cells. Then, to determine whether any unrecovered vertices are in the k -ball, the algorithm merely queries each neighboring Voronoi cell in turn.

We use an amortized analysis for the point location charges that is reminiscent of the two-dimensional analysis of Miller [Mil04]: we charge the point location not to the Voronoi node that is prompting the test, but to the unrecovered vertex being tested. There are two sub-charges: the relocation charge for updating point location information, and the charge for point location queries. Both charges are computed similarly, so in this section we will merely sketch out the former case.

Consider an unrecovered vertex u . Whenever the Voronoi cell that contains u changes, a new vertex p was inserted nearby – at a distance related to (within a constant factor of) $\text{cfs}(u)$. Furthermore, when p is inserted, SVR ensures via the yielding rules that the nearest neighbor of p is no closer than a distance related to $\text{cfs}(p)$. Finally, because u and p are close, $\text{cfs}(p)$ is related to $\text{cfs}(u)$. Thus, for every p whose insertion affects u , p is both close (distance $O(\text{cfs}(u))$) and has a large (radius $\Omega(\text{cfs}(u))$) empty ball around it. Therefore, an adversary can only pack a constant number of vertices around u before the feature size at u falls by half.

Later, we will prove that $\text{cfs}(u)$ is at most the diameter L of the periodic region, and never goes below $\Omega(s)$. Thus, the adversary can force the algorithm to halve the feature size at most $\log(L/s)$ times around u . So the number of times that u is relocated will be at most $\log(L/s)$. Precise statements of the lemmas are found in Section 8.2, while rigorous proofs appear in the tech report [HMP06].

As we develop the full algorithm, the important techniques of this section will remain the same: as long as SVR ensures that no new vertex is ever inserted too close to an existing mesh vertex, the algorithm always has a quality mesh (not too broken). In turn, this implies that the insertion is fast and that the gap size around unrecovered features falls exponentially, so that the point location costs are also small.


```

TRYADD( $p$ :  $i$ -dim Voronoi point,  $r$ : radius,  $M$ :  $i$ -dim mesh)
1: if  $\exists q \in U(M)$  such that  $|pq| \leq kr$  then
2:   FORCEADD( $q, M$ )
3: else if  $\exists B \in Balls(M)$  such that  $p \in B$  then
4:   SPLIT( $B$ ) for all such  $B$ 
5:   TRYADD( $p, r, M$ )
6: else
7:   FORCEADD( $p, M$ )
8: end if

SPLIT( $B$ : protecting ball from a mesh  $M$ )
1: TRYADD(center( $B$ ), radius( $B$ ),  $M$ )
2: while  $M$  has a skinny cell  $V(v)$  do
3:   TRYADD(far-node( $v$ ),  $R_M(v)$ ,  $M$ )
4: end while

FORCEADD( $p$ :  $i$ -dim point,  $M$ :  $i$ -dim mesh)
1: Run Bowyer-Watson or Edelsbrunner-Shah to insert  $p$  into the mesh  $M$ .
2: Reassign vertices in  $U(M)$  and protecting balls in  $Balls(M)$  to their new Voronoi cells as needed.
3: for every higher-dimensional mesh  $M^+$  that contains  $M$  as a sub-feature do
4:   Remove from  $Balls(M^+)$  all the balls associated with Voronoi nodes that were destroyed.
5:   Add to  $Balls(M^+)$  a ball for every Voronoi node that was created.
6:   Add  $p$  to  $U(M^+)$  if not already present.
7: end for
8: {The following only occurs if  $p$  was a vertex in  $P(M)$ .}
9: if  $cd(p) < i$  then remove  $p$  from  $U(M)$ 
10: while  $\exists B \in F(M)$  such that  $p \in B$  do
11:   if  $p$  is the second encroachment on  $B$ , SPLIT( $B$ )
12: end while

```

Fig. 3. The subroutines used in the SVR algorithm.

5 Full Algorithm

The input is described as a PLC, and must meet the criteria described in Section 3.

As scratch data structures, we maintain, *for every input feature* including the complete domain:

- A mesh M . We will prove that each mesh always has good quality (good Voronoi aspect ratio).
- A mapping U from each Voronoi cell in M to the list of uninserted vertices in that cell of containment dimension $< i$.
- A mapping $Balls$ from each Voronoi cell in M to the list of balls intersect the cell (these balls are used to protect lower-dimensional features).

A protecting ball is a ball $B(p, NN_{M^-}(p))$ centered at a Voronoi node of a lower-dimensional mesh M^- . This corresponds exactly to the circumball of a lower-dimensional subfacet.

In the bootstrapping phase of the algorithm, we initialize the data structures for each feature in order of increasing dimension. We generate the Voronoi diagram of the convex hull of each feature F , then partition each sub-feature's protecting balls among the cells of $M(F)$ to create the S mapping. We also partition the vertex set similarly.

As in the point-set case, the algorithm will now iteratively perform *break* and *clean* moves until it reaches a quality conformal mesh. The *clean* move is identical to before (except for an updated version of TRYINSERT). The *break* move is now expanded to try to break encroached balls using the new SPLIT routine. Encroached balls correspond to lower-dimensional input features that has not yet been recovered.

However, we must sequence the moves correctly across dimensions. In particular, we maintain the invariant that every mesh is always of good quality. We first refine for quality (clean moves) from the bottom (lowest dimension) up. Next, we refine for input (break moves) from top down. This approach is critical to establishing the runtime bound, though it is irrelevant to the correctness proof.

When we consider adding a Steiner point, as before, we may have to yield to an input vertex. However, we may also have to yield to lower-dimensional input features; this is a common technique in mesh refinement. Unlike in standard Delaunay mesh refinement techniques, for runtime reasons we are required to occasionally allow input protective balls to be encroached by mesh vertices; however, we only allow a single encroachment, and only by vertices that have containment dimension less than $\dim(M)$. The need for this is based on the fact that while we can charge according to the spread (L/s) of the input for the point location of features the user input, we want the charge on sub-features the algorithm creates to be only linear in the size of the output. Allowing singly-encroached input features allows the algorithm to ensure it never performs point location on any non-input protective balls until the gap size around the protective ball is related to its diameter.

6 Structural Properties of Quality Voronoi Diagrams

In this section we present several structural lemmas about good radius-edge meshes. These lemmas are crucial for our analysis and may be well suited for timing analysis of future algorithms. For brevity, proofs are omitted, but can be found in [HMP06].

Let M be a τ -quality Voronoi diagram in \mathbb{E}^d . Our first main goal in this section is to show that the cfs of any point in a gap-ball is bounded below by a constant times the ball's radius.

Lemma 1. *Suppose that M is a τ -quality Voronoi diagram, and suppose that B is a gap-ball of $V(M)$ with center c and radius r . If $x \in B \cap CC(M)$ then*

$$\text{cfs}(x) \geq c_1 r$$

where c_1 depends only on τ and is independent of dimension.

We now state a lemma that is functionally equivalent to Lemma 1, but allows the gap-ball to have one vertex encroaching it.

Lemma 2. *Suppose M is a τ -quality Voronoi diagram and p is a Voronoi vertex. Define M' as the Voronoi diagram of $V(M) \setminus \{p\}$. Suppose B is a gap-ball of radius r in M' , then for all $x \in B$:*

$$\text{cfs}_M(x) \geq c_2 r$$

We now state two lemmas that relate the grading Γ of a Voronoi diagram to the quality τ . These lemmas are not really new, but we include more simple formulations than prior work.

Lemma 3. (Quality Gives Bounded Grading)

Suppose M is a τ -quality Voronoi diagram. Then there exists a constant c_3 depending only on τ such that $\Gamma_M(x) \leq c_3$ for any $x \in CC(M)$.

Lemma 4. (Bounded Grading Gives Quality)

Suppose we have a Voronoi diagram M , and suppose that $\Gamma_M(p) \leq \tau$ at every vertex $p \in M$. Suppose further that every Voronoi node of M is contained in $CC(M)$. Then M is a 2τ -quality Voronoi diagram.

We note that the hypothesis for Lemma 4 is satisfied when we have a τ -quality mesh such that the diametral ball of every mesh simplex on the convex hull is unencroached.

7 Spacing

The key lemma that Ruppert used in his paper stated that the algorithm will never insert two points more than a constant factor closer to each other than what is dictated by the lfs function over the input – in fact, it will never even *consider* inserting a point too close to a neighbor. By controlling the spacing of output vertices, Ruppert could then prove that his algorithm terminates with a mesh of guaranteed size. We will do the same; and we also need to control the spacing in order to achieve our time bounds.

The statement of the theorem is as follows:

Theorem 1. *For any point p considered for insertion into a mesh M , whether or not p is eventually inserted, we have that $\text{lfs}(p) \leq CNN(p)$*

From this we get a corollary which allows us to bound the nearest neighbor of a mesh vertex at any time – in particular, at the end of the algorithm:

Corollary 1. *In any mesh M produced during the run of the algorithm, for any mesh vertex $v \in M$, we have that $\text{lfs}(v) \leq (1 + C)NN_M(v)$*

The proofs herein are somewhat simplified from those presented by many prior authors, even as they are generalized to higher dimension. As in most such analyses, we split the single constant c into a constant C_i for every dimension $1 \dots d$. In our algorithm, unlike in most, we need an additional constant C_0 for the spacing of input vertices when they are inserted.

Lemma 5. *Let p be an input vertex being inserted by SVR. Then:*

$$\text{lfs}(p) \leq \left(1 + \frac{C_1}{k}\right)NN_M(p)$$

Proof. Let v be the vertex in whose cell p lies. If v has containment dimension 0, then the lemma is obvious. Otherwise, we know $\text{lfs}(v) \leq C_1NN_{M_v}(v)$ by the usual inductive argument. Furthermore, we know that v did not yield to input vertex p when v was inserted, which means that $|vp| \geq kNN_{M_v}(v)$. Finally, we use the Lipschitz condition and some algebra to prove the result. This shows that $C_0 \geq 1 + C_1/k$.

The bounds on C_i for $i > 0$ are exactly analogous to prior work but with a $1/k$ factor. This gives a linear program which we can solve for C_0 , which shows that the sizing theorem holds whenever $\tau k^d > 2^{d-1/2}$. In other words, for any $\tau > 2^{d-1/2}$, there is a k close enough to 1 that allows the proof to go through. Setting $k = 1$ gives a correct algorithm, but we will see that the algorithm will run in time proportional to $(1 - k)^{-1}$ which suggests a tradeoff between solution quality and runtime.

7.1 Size Optimality

A proof due to Ruppert shows that, so long as the sizing condition is guaranteed – as it is for SVR, according to Theorem 1 –, then on point set input, the mesh is within a constant factor of the smallest mesh that respects the input vertices. As shown by Shewchuk [She98], this proof fails in the presence of input features (even just segments) in dimension at least 3: by using slivers, on certain inputs, we can produce an arbitrarily smaller mesh than what is dictated by the local feature size. Given that slivers are undesirable, it is unclear exactly what the correct definition of size optimality is, when in the presence of features.

8 Timing Analysis Overview

8.1 Mesh Update Work

We claim that the entire refinement process modifies only $O(m)$ Voronoi cell boundaries over the life of the algorithm. Hence, the cost of maintaining the mesh is $O(m)$.

Proof (see [HMP06]) proceeds by first showing that the mesh is always a quality mesh at any point during the algorithm. It is then shown that the mesh is always sparse. Finally, a charging argument to count the total number of edges.

Lemma 6 (Always Quality). *At any point during Sparse Voronoi Refinement, the intermediate mesh is a τ' -quality mesh.*

Theorem 2 (Sparse Mesh). *Any intermediate mesh during the lifetime of Sparse Voronoi Refinement is sparse, i.e. there is a constant depending only on τ and k that bounds the degree of every vertex.*

Corollary 2 (Mesh Update Charge). *The number of Voronoi facets that are ever created is $O(m)$, so that the overall time spent updating the mesh is $O(m)$.*

Proof. Any time a Voronoi facet f is created, it is adjacent to a freshly inserted vertex v . We will charge the cost of creating f to the insertion of v . Consider the intermediate mesh immediately after the insertion of v . By Theorem 2, v is of constant degree, hence the number of facets charged to v is constant, so the total number of Voronoi facets that are ever created is linear in the total number of vertices ever inserted, hence $O(m)$.

8.2 Accounting for Point Location and Relocation

We sketch a basic overview of the timing analysis for point location work. A rigorous analysis is omitted for brevity, but can be found in [HMP06].

Consider a feature and the sequence of meshes of that feature that the algorithm produces: $M_0, M_1, \dots, M_i, \dots, M_j, \dots, M_m$. We claim that through this sequence, no more than $O(1)$ point location events affect any ball B with center c before $\text{cfs}(c)$ falls by half, i.e.:

Lemma 7. *If $\text{cfs}_{M_j}(c) \geq \text{cfs}_{M_i}(c)/2$ then B was relocated at most $O(1)$ times during the insertion of points p_i through p_j .*

Lemma 8. *If $\text{cfs}_{M_j}(c) \geq \text{cfs}_{M_i}(c)/2$ then B was searched or tested for encroachment at most $O(1)$ times during the insertion of points p_i through p_j .*

Together, these lemmas tell us that point location events must be in some sense “packed” around the points being located. It follows that the number of point location events around a point (or ball) to be inserted is bounded by the log of the ratio between the farthest and the nearest event.

For input points and features, this bounds the number of events to $\log(L/s)$. For created balls and queued points, it can be shown that the number of events is only a constant. Together this gives a bound on point location costs of $O(n \log(L/s) + m)$.

9 Conclusions

We have shown how to produce, in near-optimal time, a conformal mesh of the input domain, in arbitrary dimension. While this is a first, there are many remaining questions.

Firstly, we allow the user to demand a value of $\tau \geq 2^{d-1/2}$, equivalent to a radius-edge ratio of $\rho \geq 2^{d-3/2}$. With some proof work (without changing the algorithm at all), we know how to improve this slightly in $d > 2$, but not substantially. In two dimensions, our bound matches Ruppert’s original bound of 20.7° , or $\rho \geq 2\sqrt{2}$ – and indeed it should, since our proof is based on the same precepts. In the decade since the publication of Ruppert’s result, his proof has been improved to allow the user to demand angles of more than 25° ; and it is believed that the correct answer is that the user should be able to demand almost 30° on any input (of course, on some inputs, the user can demand even larger angles). It is less clear how the bound changes according to dimension, but we believe our stated bound is much too conservative.

Most egregiously, we have demanded that the user must give us a PLC with all angles orthogonal or obtuse. Several recent papers have begun addressing the issue of removing the 90° angle restriction [CP03, PW04]. These techniques seek to create, at initialization, a protective region around small angles, requiring an oracle that will provide the algorithm with the lfs at many points. This requirement leads to very poor runtime bounds with any naive analysis. To incorporate these methods into SVR, the natural method would be to create some protective region that could adapt as SVR recovers more input features. Indeed, the recent work of Pav and Walkington [PW04] appears to be moving in this direction, as they attempt to reduce the oracular requirements. We believe that as algorithms for three dimensional meshing with arbitrary domains continue to mature, they can be incorporated into SVR to achieve better runtime guarantees.

Certainly, future work will include the parallelization of SVR, which is important for any modern large-scale meshing algorithm. All of the mesh modifications in SVR are local, so basic shared memory algorithmic techniques involving a conflict graph will most likely suffice. This analysis is in progress, and we do not foresee any major difficulties.

The last possibility is the reduction of the point location costs from $O(n \log(L/s))$ down to $O(n \log n)$ for inputs with pathological spread. This is mainly of theoretical concern. One possibility would be to cluster together several small input features and point locate them as a conglomerate, until the mesh is refined down to a relatively polynomial spread. It is quite unclear how to properly define such a clustering strategy, but vague intuitions suggest that something akin to well-separated pair decompositions [CK92] might suffice.

References

- [BEG94] Marshall Bern, David Eppstein, and John R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.
- [CD03] Siu-Wing Cheng and Tamal K. Dey. Quality meshing with weighted delaunay refinement. *SIAM J. Comput.*, 33(1):69–93, 2003.
- [Che89] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.
- [Che97] L. Paul Chew. Guaranteed-Quality Delaunay Meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393, Nice, France, June 1997. Association for Computing Machinery.
- [CK92] P. Callahan and S. Kosaraju. A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields, 1992.
- [CP03] Siu-Wing Cheng and Sheung-Hung Poon. Graded Conforming Delaunay Tetrahedralization with Bounded Radius-Edge Ratio. In *Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms*, pages 295–304, Baltimore, Maryland, January 2003. Society for Industrial and Applied Mathematics.
- [ELM⁺00] Herbert Edelsbrunner, Xiang-Yang Li, Gary L. Miller, Andreas Stathopoulos, Dafna Talmor, Shang-Hua Teng, Alper Üngör, and Noel Walkington. Smoothing and cleaning up slivers. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, pages 273–277, Portland, Oregon, 2000.
- [Eri01] Jeff Erickson. Nice point sets can have nasty delaunay triangulations. In *Symposium on Computational Geometry*, pages 96–105, 2001.
- [HMP06] Benoit Hudson, Gary Miller, and Todd Phillips. Sparse Voronoi Refinement. Technical Report CMU-CS-06-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 2006.
- [HPU05] Sariel Har-Peled and Alper Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. In *Symposium on Computational Geometry*, 2005.
- [Li03] Xiang-Yang Li. Generating well-shaped d -dimensional Delaunay meshes. *Theor. Comput. Sci.*, 296(1):145–165, 2003.
- [LT01] Xiang-Yang Li and Shang-Hua Teng. Generating well-shaped Delaunay meshed in 3D. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–37. ACM Press, 2001.

- [Mil04] Gary L. Miller. A time-efficient Delaunay refinement algorithm. In *Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 400–409, New Orleans, 2004.
- [Mit93] Scott A. Mitchell. Refining a Triangulation of a Planar Straight-Line Graph to Eliminate Large Angles. In *34th Annual Symposium on Foundations of Computer Science*, pages 583–591. IEEE Computer Society Press, 1993.
- [MPW02] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. Fully Incremental 3D Delaunay Refinement Mesh Generation. In *Eleventh International Meshing Roundtable*, pages 75–86, Ithaca, New York, September 2002. Sandia National Laboratories.
- [MTT⁺96] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. Control Volume Meshes Using Sphere Packing: Generation, Refinement and Coarsening. In *Fifth International Meshing Roundtable*, pages 47–61, Pittsburgh, Pennsylvania, October 1996.
- [MTT99] Gary L. Miller, Dafna Talmor, and Shang-Hua Teng. Optimal coarsening of unstructured meshes. *Journal of Algorithms*, 31(1):29–65, Apr 1999.
- [MTTW95] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 683–692, Las Vegas, May 1995. ACM.
- [MV00] Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370 (electronic), 2000.
- [PW04] Steven E. Pav and Noel J. Walkington. Robust Three Dimensional Delaunay Refinement. In *Thirteenth International Meshing Roundtable*, pages 145–156, Williamsburg, Virginia, September 2004. Sandia National Laboratories.
- [Rup95] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Austin, TX, 1993).
- [She98] Jonathan Richard Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry (Minneapolis, Minnesota)*, pages 86–95. ACM, June 1998.
- [She02] Jonathan Richard Shewchuk. Constrained delaunay tetrahedralizations and provably good boundary recovery. In *Eleventh International Meshing Roundtable*, pages 193–204, Ithaca, New York, September 2002. Sandia National Laboratories.
- [STÜ02] Daniel Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings, 11th International Meshing Roundtable*, pages 205–218. Sandia National Laboratories, September 15-18 2002.
- [Tal97] Dafna Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997. CMU CS Tech Report CMU-CS-97-164.

Session 4

Geometry

Volume and Feature Preservation in Surface Mesh Optimization

Xiangmin Jiao

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332

Summary. Mesh optimization is critical in numerical simulations involving complex or evolving geometry. Because of the geometric constraints, such as preservation of sharp features and conservation of volume, optimizing a surface mesh poses significant challenges, especially when a CAD model is unavailable. In this paper, we introduce a formulation of volume conservation in a local sense for surface meshes under smoothing or other types of mesh motion, and propose a simple and efficient technique to solve it. We also present a simple and robust feature detection technique to enhance the effectiveness of local volume conservation and mesh optimization. We present the theoretical foundation of our techniques and experimental study to demonstrate their effectiveness.

Key words: Mesh optimization; mesh smoothing; surface meshes; volume conservation; feature detection

1 Introduction

Mesh optimization is important in mesh generation for numerical simulations [8] and in many simulations with moving boundaries [3]. For its potential high efficiency and simplicity, a commonly used optimization strategy is mesh smoothing, which redistributes the vertices without changing the connectivity of a mesh. Although it has been widely used in optimizing 2-D and 3-D meshes [2, 4, 5, 12, 17], smoothing a surface mesh has some significant challenges due to additional geometric constraints. Two critical and nontrivial constraints are the preservation of sharp features and conservation of volume. Although some sophisticated techniques were developed and used in stand-alone meshing and remeshing tools, they are hard to implement and ill-suited for numerical simulation codes, especially on parallel computers. The lack of simple and effective surface mesh smoothing techniques significantly limits the capabilities or efficiencies of numerical simulations involving complex geometry under significant motion, such as in multiphase flows [25].

Conservation of volume (or mass) is a fundamental issue for accurate and stable numerical simulations, especially the simulations of dynamic systems over a long period of time [19]. It is well-known that a naive procedure, such as Laplacian smoothing, may shrink the volume of a domain substantially [24]. Volume conservation has attracted significant attention in recent years in mesh optimization and surface fairing [9, 19, 23, 27]. In the context of mesh optimization, most methods primarily focus on limiting volume errors during mesh generation or remeshing by projecting the vertices onto a continuous or discrete surface [6, 7, 9, 18]. These methods involve point location procedures, which are potentially expensive and difficult to implement especially on a parallel computer. In addition, if the mesh-optimization procedure must be called repeatedly as required in many numerical simulations, severe volume errors may still occur due to accumulation of errors. The method in [19] enforces volume conservation up to machine precision, but it may incur large local errors near singularities or sharp features.

Another issue intimately related to volume conservation is feature detection and preservation. Feature detection is a critical issue also in its own right in mesh generation, mesh optimization, and numerical simulations [1, 11, 16, 20, 26]. Without proper treatment of features or singularities, large errors may occur during a meshing or remeshing process, and significant undershoots or overshoots may occur near singularities in high-order approximations to a surface. Within numerical simulations, improper treatment of features may undermine the accuracy or stability. Although a number of feature detection techniques have been proposed in the literature, the most robust techniques are also difficult to implement and typically lack a consistent theoretical foundation. Although they may be sufficient for stand-alone meshing tools and interactive environments, simple and robust feature detection techniques are still needed for many other computational applications.

This paper aims at developing simple and efficient techniques for surface mesh optimization that can be easily integrated into numerical simulations based on sound mathematical foundations. The main contributions of the paper are twofold. First, we formulate volume conservation in a *local* sense for meshes under smoothing or other types of mesh motion, and propose a simple and efficient technique to achieve volume conservation. Second, we propose a robust feature detection technique that is relatively easy to implement and to integrate into meshing processes or numerical simulations. Both of these techniques are based on an eigenvalue analysis of a metric tensor at each vertex of the mesh, which combines the asymptotic analysis for fine discretizations of smooth surfaces and the singularity analysis for coarse meshes or sharp features in a fine mesh. This analysis establishes a new theoretical foundation for feature detection and volume conservation in mesh optimization and leads to simple and robust algorithms for them.

The remainder of the paper is organized as follows. Section 2 presents the basic framework for our surface mesh smoothing, referred to as *null-space smoothing*. Section 3 presents a formulation and numerical solution to enforce

volume conservation in a local sense. Section 4 describes a simple and robust feature detection technique that reuses the computational tools present in earlier sections and improves the effectiveness of mesh smoothing and local volume conservation. Section 5 demonstrates the effectiveness of our new technique in surface mesh smoothing. Section 6 concludes the paper with a discussion.

2 Null-Space Smoothing

2.1 Formulation

We review our basic framework for surface mesh smoothing, referred to as *null-space smoothing*, proposed previously in [15]. This technique smooths a surface mesh by moving each vertex within a *null space*, which in general is a plane, a line, or the empty set, tangential to the surface at the vertex. Let \mathbf{T} denote a matrix whose column vectors are the bases of the null space, and let \mathbf{c} denote the vector from v to the centroid (or a weighted average) of its neighborhood. The null-space smoothing moves v toward the centroid within the null space for a displacement \mathbf{t} , i.e.,

$$\mathbf{t} = \mathbf{T}\mathbf{T}^T\mathbf{c}. \quad (1)$$

We define the *null space* using an eigenvalue analysis of a metric tensor. At each vertex v , suppose v is the origin of a local coordinate frame, and m be the number of the faces incident on v . Let \mathbf{N} be an $m \times 3$ matrix whose i th row vector is the unit outward normal to the i th incident face of v , and \mathbf{W} be an $m \times m$ diagonal matrix with W_{ii} equal to the weight (such as the face area) associated with the i th face. Let \mathbf{A} denote $\mathbf{N}^T\mathbf{W}\mathbf{N}$, which we refer to as the *quadric metric tensor*, for its use in the well-known quadric error metric [14]. \mathbf{A} is symmetric positive semi-definite (i.e., $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for any vector \mathbf{x}), and its eigenvalue decomposition [10] is

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T, \quad (2)$$

where its eigenvalues $\lambda_i = A_{ii}$ are all real and nonnegative, and its corresponding eigenvectors \mathbf{e}_i are the column vectors of \mathbf{V} . We refer to the vector space spanned by the eigenvectors corresponding to relatively large eigenvalues of \mathbf{A} as its *primary space* and the complementary space as its *null space*.

In general, the null space at a vertex is tangent to the surface. For sufficiently fine meshes, the null space has dimensions 2, 1, or 0 at a smooth, ridge, or corner vertex, respectively, so it is essentially the tangent space of the surface at a vertex. For relatively coarse meshes, however, it is in general a subspace of the tangent space (e.g., it may be a line along the direction of minimum curvature in the tangent plane). Null-space smoothing moves a vertex within the tangent plane or along the direction of minimum curvature,

and therefore it tends to preserve sharp features and areas with large curvatures. In addition, it stops changing a surface after the mesh has converged tangentially. Therefore, null-space smoothing is expected to work well and typically introduce negligible perturbation to a surface in practice.

2.2 Analysis of Null-Space Smoothing

We now analyze the error in null-space smoothing more formally in terms of volume change. Consider moving a vertex v by a displacement \mathbf{t} . Let a_i and \mathbf{n}_i be the area and unit normal of the i th incident face of v , respectively, and assume $W_{ii} = a_i$. The volume change is $\delta V = \sum_{i=1}^m a_i \mathbf{n}_i^T \mathbf{t} / 3 = \|\mathbf{W}\mathbf{N}\mathbf{t}\|_1 / 3$. From the definition of \mathbf{A} , we have the singular value decomposition [10]

$$\sqrt{\mathbf{W}}\mathbf{N} = \mathbf{U}\sqrt{\mathbf{\Lambda}}\mathbf{V}^T, \tag{3}$$

where \mathbf{U} is an $m \times 3$ matrix, and $\sqrt{\mathbf{W}}$ is the diagonal matrix whose i th diagonal entry is the square root of W_{ii} (and similarly for $\sqrt{\mathbf{\Lambda}}$). Let $\mathbf{s}^T = \|\sqrt{\mathbf{W}}\mathbf{U}\|_1$, and then,

$$3\delta V = \|\mathbf{W}\mathbf{N}\mathbf{t}\|_1 = \|\sqrt{\mathbf{W}}\mathbf{U}\sqrt{\mathbf{\Lambda}}\mathbf{V}^T\mathbf{t}\|_1 = \sum_{i=1}^3 s_i \sqrt{\lambda_i} (\mathbf{t}^T \mathbf{e}_i). \tag{4}$$

If the mesh is coarse, then s_i (and similarly for $\mathbf{t}^T \mathbf{e}_i$) may have comparable sizes for different i , so null-space smoothing introduces a relatively small volume change proportional to the square roots of the smallest eigenvalues. Assume the mesh is relatively uniform and let h be a measure of the average edge length. Following an analysis similar to that in [14], it can be shown that as h approaches 0, the largest eigenvalues are $O(h^2)$ but the smallest eigenvalues corresponding to the null space of \mathbf{A} are $O(h^4)$. In addition, $\mathbf{t}^T \mathbf{e}_i$ is $O(h^2)$, so δV is $O(h^4)$ in null-space smoothing with a positional error of $O(h^2)$ even near singularities. If \mathbf{t} contained a component corresponding to larger eigenvalues (such as in Laplacian smoothing), then δV would be $O(h^3)$ with a positional perturbation of $O(h)$ near singularities. Therefore, null-space smoothing works significantly better than Laplacian smoothing.

3 Volume Conservation

Moving one vertex in null-space smoothing preserves the potential second-order accuracy of a surface triangulation. However, moving the vertices for many iterations may still degrade the order of accuracy. We propose an extension of null-space smoothing to reduce the volume error further. Our basic idea is to add a small component within the primary space at the vertex to correct the volume while preserving the singularities of the surface. If we move

each vertex one by one (i.e., updating in a Gauss-Seidel style), then the problem is relatively simple: we just need to determine a direction \mathbf{d} in the primary space and then a length α , such that $\alpha\|\mathbf{W}\mathbf{N}\mathbf{d}\|_1 = -\|\mathbf{W}\mathbf{N}\mathbf{t}\|_1$, because from (4) it is obvious that moving the vertex by $\alpha\mathbf{d} + \mathbf{t}$ leads to no volume change. If all vertices are moved concurrently (i.e., updating in a Jacobi style, which is advantageous for its preservation of symmetry and ease of parallelization), we may determine the direction for each vertex independently but the distances of their movement must be solved concurrently. In the following, we will first describe how to determine the directions that are applicable to both Gauss-Seidel and Jacobi styles, and then present how to compute the motion in the Jacobi style.

3.1 Estimation of Directions

Let us first consider the problem of estimating a direction at each vertex. The key requirement of this direction is that it must be in the primary space. Intuitively, one might determine a weighted average of face normals and project it onto the primary space, but a naive implementation may be sensitive to weights and hence prone to artifacts near singularities. It is desirable to find a direction that is well behaved near singularities so that it does not vary abruptly between two neighboring ridge vertices.

We compute the directions using a *mean normal* based on an extension of the preceding eigenvalues analysis. Suppose all the faces are offset outwards for a unit distance, and the intersection of the planes passing through the offset faces incident on a vertex v is then the solution to an $m \times 3$ linear system

$$\mathbf{N}\mathbf{x} \approx \mathbf{1}. \tag{5}$$

Since \mathbf{N} may be over- or under-determined, we reformulate it in a least squares sense and obtain a 3×3 linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{6}$$

where $\mathbf{b} = \mathbf{N}^T\mathbf{W}\mathbf{1}$. Let k denote the dimension of the primary space, and assume $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Generally speaking, k is one, two, and three at a smooth, ridge, and corner vertex, respectively, but may be smaller for extremely shape ridge or corner vertices where λ_2 or λ_3 is too small compared to λ_1 (e.g., $\leq \varepsilon\lambda_1$ for $\varepsilon \approx 0.003$). By restricting \mathbf{x} to be within the primary space, the solution to (5) is then

$$\mathbf{x} \approx \sum_{i=1}^k \mathbf{e}_i^T \mathbf{b} \mathbf{e}_i / \lambda_i, \tag{7}$$

which is numerically more stable by avoiding division by very small numbers. If $k = 1$ (i.e., the surface is smooth at the vertex), then \mathbf{x} reduces to the first eigenvector, which converges to the outward surface normal. If $k > 1$ (i.e., the surface is singular at the vertex), then \mathbf{x} approximately points to the medial

axis of the surface and in turn approximately bisects the tangent planes at the singularity. For these reasons, \mathbf{x} provides a good estimation of normals and is well behaved at singularities, and therefore we use $\mathbf{d} = \mathbf{x}/\|\mathbf{x}\|$ as the direction at each vertex for volume correction.

3.2 Concurrent Vertex Motion

After obtaining the direction at each vertex, we must then solve for the distance α that each vertex moves. In a Gauss-Seidel-style iteration, α is simply $-(\mathbf{b}^T \mathbf{t})/(\mathbf{b}^T \mathbf{d})$ at a vertex, which is similar to the single-node relaxation method in [19]. For Jacobi iterations, however, the problem is more difficult because the volume swept by each face is in general nonlinear in the displacements of the vertices. Although a naive technique such as rescaling the domain or making α uniform for all vertices may restore the total volume in a global sense, it has no physical meaning and may undermine accuracy. We propose a new approach that formulates volume conservation in a local sense to obtain a system of equations and then solves the equation efficiently using a simple iterative procedure. Although our focus is mesh smoothing, we present our formulation in a general form so that it can be easily adapted to other settings.

Suppose a volume flux (or mass flux) ϵ is given over a surface (where the flux may be due to the tangential motion in the null-space smoothing or other types of surface motion). We define a numerical flux f at each face to be the gain or loss of volume per unit area, i.e.,

$$f = \left(\int_{e'} \mathbf{x}^T \mathbf{n}' \, d\mathbf{x} - \int_e \mathbf{x}^T \mathbf{n} \, d\mathbf{x} \right) / \int_e 1 \, d\mathbf{x}, \tag{8}$$

where e and e' denote the face before and after adding the normal motion $\alpha \mathbf{d}$ at the vertices, and \mathbf{n} and \mathbf{n}' denote their unit normals, respectively. Our objective is to make the numerical flux f as close to the prescribed flux ϵ as possible, i.e., $f \approx \epsilon$. Using a weighted-residual method, we obtain a weak form by requiring the error $f - \epsilon$ project orthogonally onto the function space of the basis functions of the mesh, i.e.,

$$\int (f - \epsilon) \omega_i \, d\Gamma = 0 \tag{9}$$

for each shape function ω_i of the mesh. From the summation property $\sum_i \omega_i = 1$, this weak form of local volume conservation enforces global volume conservation strictly, i.e., $\int f \, d\Gamma = \int \epsilon \, d\Gamma$.

Given a triangle $e = \mathbf{p}_0 \mathbf{p}_1 \mathbf{p}_2$, let $\tilde{\mathbf{d}}_i = \alpha_i \mathbf{d}_i$ and $\mathbf{q}_i = \mathbf{p}_i + \tilde{\mathbf{d}}_i$. Let $\mathbf{t}_1 = \mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{t}_2 = \mathbf{p}_2 - \mathbf{p}_0$, and $\mathbf{n} = \mathbf{t}_1 \times \mathbf{t}_2$. Given the displacements, we can evaluate the integral in (9) up to machine precision using numerical quadrature. To analyze f , we decompose the prism between e and e' into three tetrahedra, $\mathbf{p}_0 \mathbf{p}_1 \mathbf{p}_2 \mathbf{q}_1$, $\mathbf{p}_0 \mathbf{q}_1 \mathbf{p}_2 \mathbf{q}_2$, and $\mathbf{p}_0 \mathbf{q}_1 \mathbf{q}_2 \mathbf{q}_0$, as illustrated in Fig. 1. If \mathbf{d}_i has the same direction for all vertices, the swept volume V of e is then

$$\begin{aligned}
 V &= \frac{1}{6} \left(\mathbf{n}^T \tilde{\mathbf{d}}_1 + (\mathbf{t}_1 + \tilde{\mathbf{d}}_1) \times (\mathbf{t}_2 + \tilde{\mathbf{d}}_2) \cdot (\tilde{\mathbf{d}}_0 + \tilde{\mathbf{d}}_2) \right) \\
 &= \frac{1}{6} \left(\mathbf{n}^T (\tilde{\mathbf{d}}_0 + \tilde{\mathbf{d}}_1 + \tilde{\mathbf{d}}_2) + (\tilde{\mathbf{d}}_1 \times \mathbf{t}_2 + \mathbf{t}_1 \times \tilde{\mathbf{d}}_2 + \tilde{\mathbf{d}}_1 \times \tilde{\mathbf{d}}_2)^T (\tilde{\mathbf{d}}_0 + \tilde{\mathbf{d}}_2) \right),
 \end{aligned}$$

which is a cubic function in α_i . If \mathbf{d}_i has different directions at different vertices, each quadrilateral of the prism would be bilinear, but it still holds that $V = \mathbf{n}^T (\tilde{\mathbf{d}}_0 + \tilde{\mathbf{d}}_1 + \tilde{\mathbf{d}}_2)/6 + O(\alpha^2)$. Therefore, (9) leads to a nonlinear system of equations in α_i .

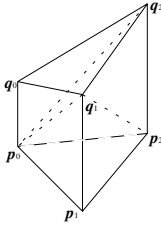


Fig. 1. Decomposition of prism swept by face $\mathbf{p}_0\mathbf{p}_1\mathbf{p}_2$ into three tetrahedra

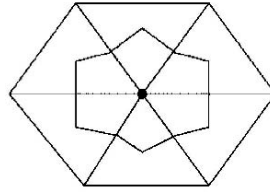


Fig. 2. Control volume for approximating numerical flux at vertex

To solve (9) efficiently, we use an efficient quasi-Newton method as follows. Let us define the control volume of each vertex to be composed of one third of each of its incident faces, as illustrated in Fig. 2. Let $g = \sum_i \alpha_i \mathbf{d}_i^T \mathbf{n} H_i$, where \mathbf{n} denote the normal field over the surface, and H_i is a step function, which is 1 in the control volume of the vertex and 0 elsewhere. Then $\int f \omega_i d\Gamma = \int g \omega_i d\Gamma + O(\alpha^2)$. Let \mathbf{D} be the diagonal matrix, where

$$D_{ij} = \frac{\partial \int g \omega_i d\Gamma}{\partial \alpha_j} = \begin{cases} \frac{1}{3} \mathbf{d}_i^T \boldsymbol{\beta} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \tag{10}$$

where $\boldsymbol{\beta} = \sum_{k=1}^m a_k \mathbf{n}_k$ (similar to \mathbf{b} in (2) but may be integrated over a different reference geometry). \mathbf{D} approximates the derivative of $\int f \omega_i d\Gamma$ with respect to α_j . Using Broyden’s method [13], $\boldsymbol{\alpha}$ can be solved iteratively using the secant equation

$$\mathbf{D} \left(\boldsymbol{\alpha}^{(k+1)} - \boldsymbol{\alpha}^{(k)} \right) = \mathbf{r}^{(k)}, \tag{11}$$

where $\mathbf{r}^{(k)}$ is the residual of the k th iteration, i.e.,

$$r_i^{(k)} = - \int (f(\boldsymbol{\alpha}^{(k)}) - \epsilon) \omega_i d\Gamma,$$

with $r_i^{(1)} = \int \epsilon \omega_i d\Gamma$. Because \mathbf{D} is a diagonal matrix, this equation can be solved very conveniently. In general, the convergence rate of Broyden’s method is superlinear and hence this method is very efficient.

We now plug in the above formulation into conservative mesh smoothing. Let Γ denote the original surface mesh and $\tilde{\Gamma}$ denote the surface mesh after tangential motion \mathbf{t} . We use $\tilde{\Gamma}$ as the reference for the numerical integration. Let $V_\gamma(\mathbf{u})$ denote the swept volume of a surface γ due to nodal displacements \mathbf{u} . The above nonlinear system would then enforce that $V_{\tilde{\Gamma}}(\tilde{\mathbf{d}}) = V_{\tilde{\Gamma}}(-\mathbf{t}) = -V_\Gamma(\mathbf{t})$, and hence

$$V_\Gamma(\mathbf{t} + \tilde{\mathbf{d}}) = V_\Gamma(\mathbf{t}) + V_{\tilde{\Gamma}}(\tilde{\mathbf{d}}) = V_\Gamma(\mathbf{t}) + V_{\tilde{\Gamma}}(-\mathbf{t}) = V_\Gamma(\mathbf{t}) - V_\Gamma(\mathbf{t}) = 0. \quad (12)$$

In summary, this conservative algorithm proceeds as following:

1. obtain intermediate surface mesh by moving vertices by \mathbf{t} , and set $\tilde{\mathbf{d}} = 0$;
2. for each vertex v , compute

$$\tilde{\mathbf{d}}_v = \tilde{\mathbf{d}}_v - \frac{\sum_e V_e(\tilde{\mathbf{d}}) - \sum_e V_e(-\mathbf{t})}{\sum_e a_e \mathbf{n}_e^T \mathbf{d}_v} \mathbf{d}_v; \quad (13)$$

3. repeat step 2 until convergence.

On line 2, the volumes, areas, and normals are computed on the intermediate surface mesh $\tilde{\Gamma}$, and the summations are over the incident faces of v . This quasi-Newton method in practice converges to nearly machine precision for only very few iterations. This method does not require the quadric metric tensor in (6) to be weighted by area, so alternative weighting schemes may be used. Note that (13) may be unstable if $\mathbf{n}_e^T \mathbf{d}_i \approx 0$. This case is unlikely to occur for our choice of \mathbf{d} except at cusps, which are inherently unstable. So vertices on cusps should in general be skipped in this procedure for robustness.

4 Feature Detection

In our preceding analysis, it is obvious that detection of geometric features is critical in preserving high-order accuracy in mesh smoothing. In addition, it is important to identify regions with relatively large curvature to reduce errors for coarse meshes. Besides surface smoothing, feature detection also plays an important role in many other geometric and numerical computations involving surfaces, such as mesh generation and remeshing [1, 26], solution transfer across different meshes [16], etc..

A number of feature detection techniques have been proposed in the literature, but most of the robust ones are relatively difficult to implement or to integrate into application codes such as numerical simulations. We present a robust feature-detection technique based on a singularity analysis of the quadric metric tensor, which is an enhancement of our preliminary results in [15]. This method is also based on the eigenvalue analysis and hence is particularly well-suited in our setting. In addition, it can be implemented using only an element connectivity table and hence is easy to be integrated into application codes even on parallel machines.

4.1 Identifying Features

The relative sizes of the eigenvalues λ_i of \mathbf{A} in (6) are closely related to the local flatness at a vertex, as illustrated in Fig. 3, where the axes of the ellipsoids are aligned with the directions of the eigenvectors and the semiaxes are proportional to the eigenvalues. In general, \mathbf{A} has three large eigenvalues at a corner, two large ones at a ridge, and one large one at a smooth point. It therefore seems natural to compare λ_3/λ_1 and λ_2/λ_1 against some thresholds to identify corners and ridges. Such a process has been used previously in processing image or meshes (such as “tensor voting” [21] and its variations [22]). However, because the metric tensor \mathbf{A} is unsigned, feature detection based on eigenvalues alone cannot distinguish a near cusp (i.e., very acute features) from a flat surface and hence is unreliable for surface meshes with very sharp features.

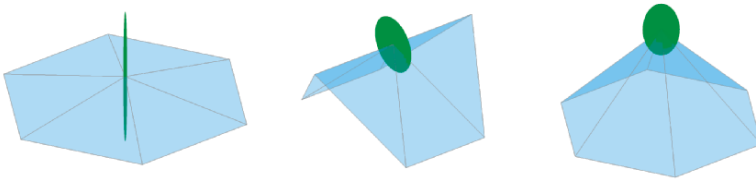


Fig. 3. Correlation of eigenvalues and local flatness

We safeguard sharp features as follows. An acute corner can be safeguarded by *angle defect*, denoted by θ_a , which is the difference between 2π and the sum of the angles at the vertex in its incident faces. To safeguard acute ridges, because the first eigenvector points toward the dominant direction of normals, its projection onto the face normals would vary in signs. Let \mathbf{e}_1 point toward the positive side of \mathbf{b} (i.e., $\mathbf{e}_1^T \mathbf{b} > 0$). We classify a vertex v as follows:

1. if $\lambda_3/\lambda_1 \geq \chi_c$ or $|\theta_a| \geq \pi/2$, then v is at corner;
2. if $\lambda_2/\lambda_1 \geq \chi_r$ or $\mathbf{e}_1 \mathbf{n} \leq 0$ in incident face, then v is on a ridge.

A tiny (close to zero) χ_c would classify all vertices as corners, and a large (close to one) χ_c would classify no corners; similarly for χ_r and the classification of ridges. Note that if the mesh has mesh folding or cusps, we can extend the second step to report a cusp if $\lambda_2/\lambda_1 \ll \chi_r$ and $\mathbf{e}_1 \mathbf{n} \leq 0$ in any face.

To obtain meaningful and intuitive values for χ_c and χ_r , we connect them with the *dihedral angle* and *open angle*. For a ridge with dihedral angle $\theta \leq \pi/2$ (i.e., the arc-cosine of the inner product of the face normals), assuming the weights in \mathbf{W} are balanced along different sides of a singular point, the eigenvalues satisfy $\lambda_2/\lambda_1 \approx \tan^2(\theta/2)$ and $\lambda_3 \approx 0$. For a cone with an opening angle $\pi - \vartheta$ (i.e., the vertex angle in the cross section through the apex and center of the base), the eigenvalues satisfy $\lambda_3/\lambda_1 \approx \lambda_2/\lambda_1 \approx 2 \tan^2(\vartheta/2)$. This

analysis provides a convenient way to choose the thresholds. Specifically, given a user-specified dihedral-angle threshold ϕ_r (on θ) and open-angle threshold ϕ_c (on ϑ), we then have $\chi_c = 2 \tan^2(\phi_c/2)$ and $\chi_r = \tan^2(\phi_r/2)$. For example, if $\phi_r = 15^\circ$ and $\phi_c = 45^\circ$, then $\chi_r \approx 0.03$ and $\chi_c \approx 0.2$. Note that these thresholds assume that the weights in \mathbf{A} are well balanced. For well-graded meshes, which are typically used in finite element analysis, the area weighting would suffice. For nonuniform meshes, as sometimes used in rapid prototyping, the angle weighting delivers more balanced weights (i.e., setting W_{ii} in (2) to be the angle at vertex v in its i th incident face).

4.2 Filtering Noise

The above feature-detection technique may be sensitive to noise, as it considers the eigenvalues at different vertices independently. It is therefore important to have additional rules to filter out false features and patch missing ones. We achieve this goal by identifying the ridge edges and using the connectivity among the ridge edges and feature vertices. We consider an edge as a *candidate ridge edge* if its dihedral angle is not too small (e.g., $> \phi_r$) and consider a ridge vertex to be *strong* if it is connected to another ridge or corner vertex by a candidate ridge edge. If the dihedral angle of a candidate edge is large (e.g., $> \phi_c$), we immediately accept it as a ridge edge. Otherwise, we accept a candidate edge if it is incident on a strong ridge vertex v and its direction is the closest to the third eigenvector at v (i.e., the inner product of the tangential direction of this edge with the eigenvector is either the positive maximum or the negative minimum among all the candidate edges incident on v).

After identifying ridge edges, we then employ them to filter out false feature vertices, based on the observation that a ridge edge usually points toward a ridge or corner vertex. The filtration proceeds as follows:

1. for each vertex, count the number k of incident ridge edges;
2. upgrade a vertex to a corner if $k > 2$;
3. upgrade a smooth vertex to a ridge if $k = 2$;
4. downgrade a ridge vertex to smooth vertex if $k < 2$ except for acute ridge vertices (i.e., $\mathbf{e}_1^T \mathbf{n} \leq 0$);
5. after reclassifying all vertices, downgrade a ridge edge if neither of its incident vertices is a ridge or corner vertex;

This procedure tends to identify the systematic patterns of ridge curves in the mesh and filter out isolated false features. If the surface is very noisy, we can reduce the sensitivity of the eigenvalues by computing the metric tensor \mathbf{A} as the sum of the tensors of itself and its neighboring ridge vertices and then repeat the classification and filtration procedure. Note that this whole detection procedure can be implemented easily by iterating through the faces or vertices. It does not require advanced data structures (such as the half-edge data structure) other than the standard element connectivity tables, so it is

particularly convenient to be integrated into numerical simulations even on parallel computers.

5 Experimental Tests

In this section, we report some experimental tests of our feature detection and conservative mesh smoothing methods. Our formulations are not limited to specific schemes for computing the centroid at each vertex. For simplicity, in our tests we use Laplacian smoothing, which computes the centroid as the average of the neighboring vertices at each vertex, but more sophisticated schemes may be used instead. Figure 4 shows the results of smoothing a surface mesh with a relatively simple geometry with sharp features to study the convergence of our method. During the process, the sharp ridges and corners are automatically identified using our feature detection technique. Four meshes of different resolutions are used and smoothed for 1000 iterations each with two volume-correction steps at each iteration. The left image in Fig. 4 shows the meshes before and after conservative smoothing drawn on top of each other. It is obvious that many vertices moved for a nontrivial distance at smooth regions and along ridges, but the surface remained on top of each other. The right image in Fig. 4 shows the convergence of volume errors using null-space smoothing and conservative smoothing with one, two or three volume-correction steps, where the x -axis corresponds to the four different meshes. The convergence rate of null-space smoothing is roughly linear with respect to grid refinement, while conservative smoothing converges faster than fourth-order and its error decreases rapidly as the number of volume-correction steps increases. Figure 5 shows the results of another example using a triple-torus, in which the features are less salient. We show the input meshes and the close-up views of the meshes before and after smoothing near a joint between the tori. The relative volume error was 2.59×10^{-8} after 500 steps of conservative smoothing with three iterations of correction steps. The geometry was clearly preserved after smoothing even with substantial tangential motion, owing to the weighted-residual formulation to minimize local errors.

To demonstrate the effectiveness of our feature detection technique, Fig. 6 shows the results of feature detection for three mechanical parts with fine features, obtained from <http://www-c.inria.fr/Eric.Saltel/download/>. The left images of Fig. 6 show the input meshes and the right images show in translucency the features detected by our method. In all the examples ϕ_r was chosen to be 15° . The first two examples (referred to as “thepart” and “fan1” in the mesh collection) are fairly representative for the coarse meshes commonly used in rapid prototyping or stereolithography. Our method accurately identified all the salient features. The third example uses a typical finite-element mesh of the “fandisk,” and remarkably some very fine features were identified by our method without any artifacts, while the commonly-used angle-based methods may have difficulties.

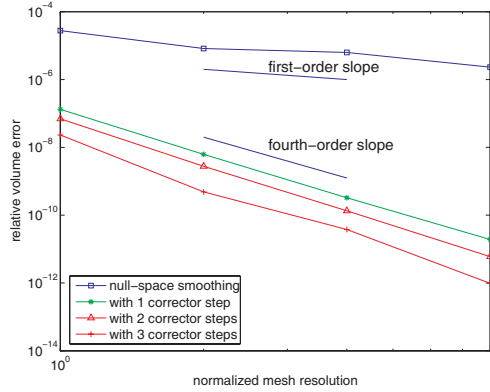
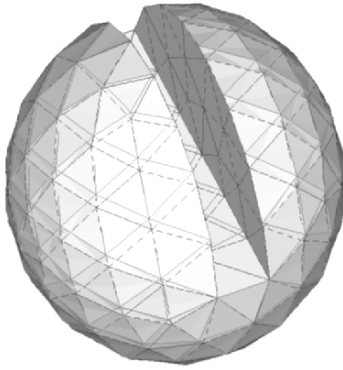


Fig. 4. Conservative smoothing of notched sphere and convergence study. In left image, dashed blue edges correspond to input mesh and solid green edges correspond to smoothed mesh

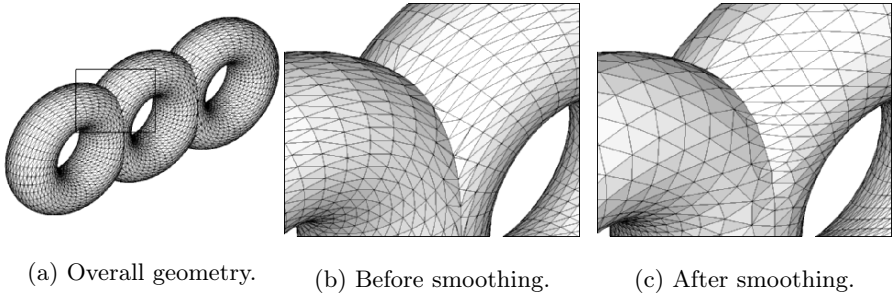


Fig. 5. Sample result of conservative smoothing of triple torus

6 Conclusion

In this paper, we presented novel techniques to conserve volume and preserve features in mesh optimization. Our techniques are based on an eigenvalue analysis of the quadric metric tensor. Due to their unified theoretical foundation, the analysis of our technique is relatively simple and coherent. More importantly, our techniques are easy to implement and does not required sophisticated data structures (such as the half-edge data structure) and expensive geometric algorithms (such as high-order surface reconstruction and point location), so they are particularly suitable to be integrated into numerical simulations. The proposed volume-conservation technique is also promising to generalize to other moving surfaces with a source term. In this paper, we only reported experimental results using Laplacian smoothing to compute the centroids around each vertex, but more sophisticated schemes can be used and are currently being investigated.

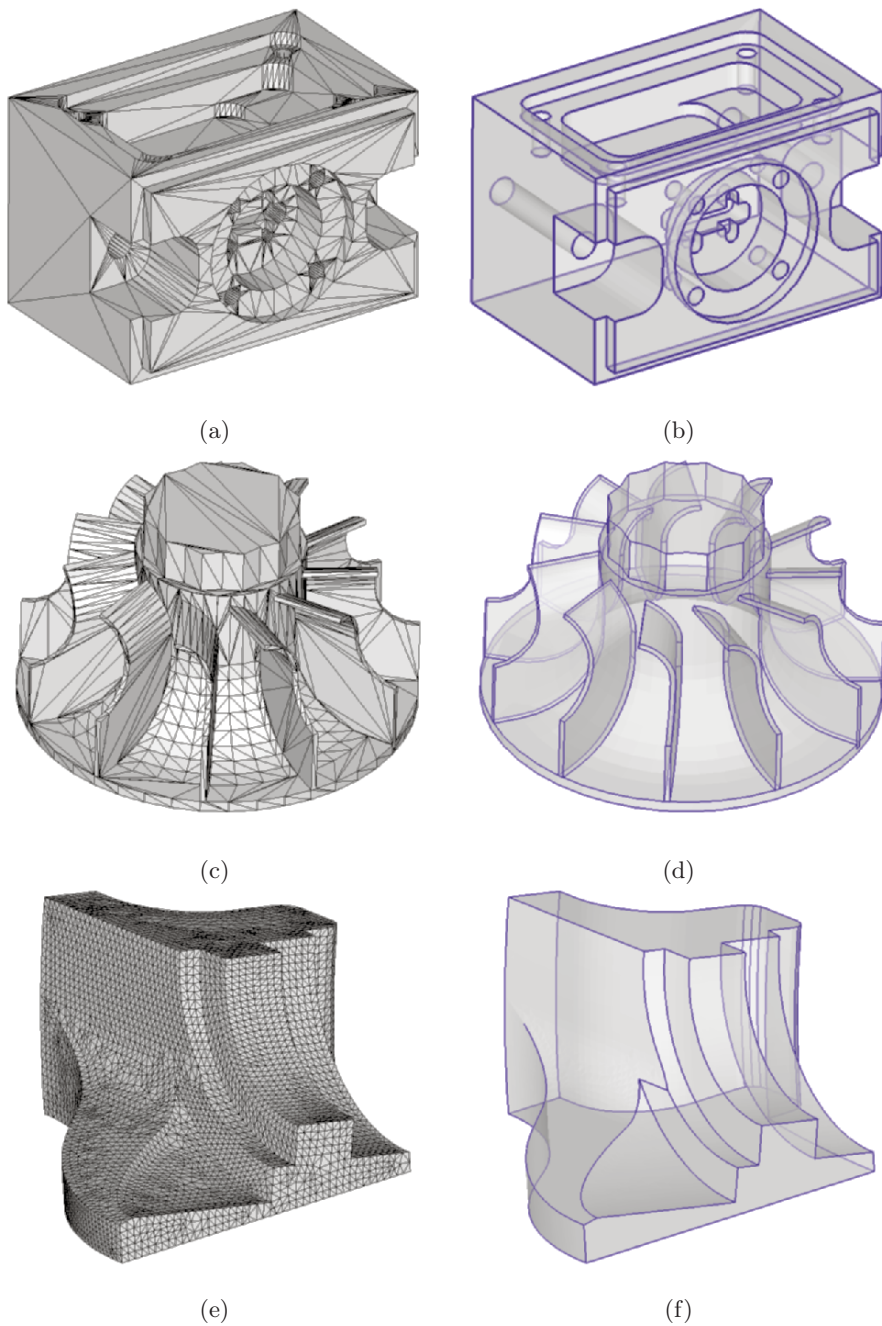


Fig. 6. Sample results of detecting features in mechanical parts

Acknowledgement

This work was supported by the U.S. Department of Energy through the University of California under subcontract B523819 with the University of Illinois at Urbana-Champaign, and by NSF and DARPA under CARGO grant #0310446.

References

- [1] T. Baker. Identification and preservation of surface features. In *Proc. 13th Int. Meshing Roundtable*, pages 299–310, 2004.
- [2] M. Brewer, L. F. Diachin, P. Knupp, T. Leurent, and D. Melander. The Mesquite mesh quality improvement toolkit. In *Proc. 12th Int. Meshing Roundtable*, pages 239–250, 2003.
- [3] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodriguez-Ferran. Arbitrary Lagrangian-Eulerian methods. In E. Stein, R. de Borst, and T. J. Hughes, editors, *Encyclopedia of Computational Mechanics*, chapter 14. Wiley, 2004.
- [4] L. A. Freitag and P. M. Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *Int. J. Numer. Meth. Engr.*, 53:1377–1391, 2002.
- [5] L. A. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *Int. J. Numer. Meth. Engr.*, 49:109–125, 2000.
- [6] P. Frey and H. Borouchaki. Geometric surface mesh optimization. *Computing and Visualization in Science*, pages 113–121, 1998.
- [7] P. J. Frey. About surface remeshing. In *Proc. 9th Int. Meshing Roundtable*, pages 123–136, 2000.
- [8] P. J. Frey and P. George. *Mesh Generation: Application to finite elements*. Hermes, 2000.
- [9] R. V. Garimella, M. J. Shashkov, and P. M. Knupp. Surface mesh quality optimization using local parametrization. In *Proc. 11th Int. Meshing Roundtable*, pages 41–52, 2002.
- [10] G. H. Golub and C. F. Van Loan. *Matrix Computation*. Johns Hopkins University Press, 3rd edition, 1996.
- [11] S. Gumhold, X. Wang, and R. Macleod. Feature extraction from point clouds. In *Proc. 10th Int. Meshing Roundtable*, pages 293–305, 2001.
- [12] G. A. Hansen, R. W. Douglass, and A. Zardecki. *Mesh Enhancement: Selected Elliptic Methods, Foundations And Applications*. Imperial College, 2005.
- [13] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, New York, 2nd edition, 2002.
- [14] P. S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Comput. Geom.*, pages 49–65, 1999.
- [15] X. Jiao and P. Alexander. Parallel feature-preserving mesh smoothing. In *Proc. Int. Conf. Comput. Sci. Appl.*, pages 1180–1189, 2005.
- [16] X. Jiao and M. T. Heath. Overlaying surface meshes, part ii: Topology preservation and feature detection. *Int. J. Comput. Geom. Appl.*, 14:403–419, 2004.
- [17] P. Knupp and S. Steinberg. *Fundamentals of Grid Generation*. CRC Press, 1994.

- [18] P. M. Knupp. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. part i—a framework for surface mesh optimization. *Int. J. Numer. Meth. Engr.*, 48:401–420, 2000.
- [19] A. Kuprat, A. Khamayseh, D. George, and L. Larkey. Volume conserving smoothing for piecewise linear curves, surfaces, and triple lines. *J. Comput. Phys.*, 172:99–118, 2001.
- [20] A. Lopez, F. Lumbreras, J. Serrat, and J. Villanueva. Evaluation of methods for ridge and valley detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21:327–335, 1999.
- [21] G. Medioni, M.-S. Lee, and C.-K. Tang. *A computational framework for segmentation and grouping*. Elsevier, 2000.
- [22] D. L. Page, A. F. Koschan, Y. Sun, J. K. Paik, and M. A. Abidi. Robust crease detection and curvature estimation of piecewise smooth surfaces from triangle mesh approximations using normal voting. In *Proc. Intl. Conf. on Computer Vision*, volume 1, pages 162–167, 2001.
- [23] I. B. Semenova, V. V. Savchenko, and I. Hagiwara. Two techniques to improve mesh quality and preserve surface characteristics. In *Proc. 13th Int. Meshing Roundtable*, pages 277–288, 2004.
- [24] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. of Int. Conf. on Computer Vision*, pages 902–907, 1995.
- [25] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for the computations of multiphase flow. *J. Comput. Phys.*, 169:708–759, 2001.
- [26] S. Yamakawa and K. Shimada. Polygon crawling: Feature-edge extraction from a general polygonal surface for mesh generation. In *Proc. 14th Int. Meshing Roundtable*, pages 257–274, 2005.
- [27] Y. Zhang, C. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In *Proc. 13th Int. Meshing Roundtable*, 2005.

On the Use of Loop Subdivision Surfaces for Surrogate Geometry

Per-Olof Persson¹, Michael J. Aftosmis², and Robert Haines³

¹ Massachusetts Institute of Technology persson@mit.edu

² NASA Ames Research Center aftosmis@nas.nasa.gov

³ Massachusetts Institute of Technology haines@mit.edu

Abstract. This work examines the use of Loop subdivision surfaces as a surrogate for CAD or analytic-based geometry. The modeler begins by constructing a subdivision surface from a full-resolution imported surface triangulation, and then queries this surface to return information about the model. Evaluations on the surface are performed using a simplified implementation of Stam’s exact evaluation procedure for Loop subdivision surfaces. The paper presents details of this simplified approach and shows how it can be used to provide surface coordinates, derivatives, and curvatures for evaluations at arbitrary parameter values. The implementation also provides the ability to tag *hard-edges* (and implicitly *hard-vertices*) in the imported geometry to preserve creases and points using a one-dimensional cubic-spline scheme which preserves C^2 continuity along hard-edges. Away from hard-edges and vertices, the Loop surface is C^2 continuous everywhere except in the immediate vicinity of irregular vertices in the control-net where it still retains C^1 continuity. Examples are presented using control-nets built from a variety of legacy triangulations with widely varying complexity. To demonstrate the modeler, we use a simplified meshing application which queries arbitrary locations on the surface to support either uniform or curvature-adaptive triangle refinement. The simple evaluation rules for surface coordinates and derivatives make the scheme extremely fast and robust. Since the input triangulation becomes the control-net of the subdivision surface, it is not necessarily an interpolant for the input data. Various approaches for making the surface interpolating are discussed, and this area remains one of active research.

1 Introduction

With increasing computing power pushing simulation technologies toward ever-higher fidelity, modeling applications in a broad spectrum of disciplines have sought ever-tighter integration with the underlying geometry. CAD-based modeling techniques have appeared in fields as diverse as Earthmoving,

Aerodynamic Vehicle Design, and Endoscopic Surgery. In the most tightly-coupled approaches, applications communicate with the underlying CAD-kernel either through vendor-provided “direct” CAD interfaces or vendor-neutral programming interfaces. In many situations, these CAD-based approaches are very attractive since they avoid translation errors and provide the modeling application with direct access to the latest revision of the fully detailed geometry. This provides important geometric consistency, since the same CAD models are used for creation, design, analysis and manufacturing.

Nevertheless, CAD-based modeling is not always either feasible or desirable. “Legacy geometry” is one example of an area which can cause difficulties for CAD-based simulations. Most simulation systems have to cope with models that date from pre-CAD eras, or even old CAD models that are somehow inconsistent when imported into current CAD systems. An aircraft geometry from two decades ago may only have been defined as a series of loftings, a faceted polyhedral mesh, or a surface triangulation that is considered coarse by the standards of modern simulations. A modern finite element solver may require position and curvature information *within* the triangles of a surface triangulation to achieve higher-order accuracy. CAD definitions for legacy models get lost, they get damaged by translation, or often they simply never existed.

In other situations, CAD-based modeling is simply not desirable. Using the CAD system to perform geometry queries and manipulation means that this system must be running and available to serve your application. This consumes CAD licenses that are typically both expensive and limited in number. Moreover, it requires the CAD system to be running either on the same platform as the simulation code or in direct communication with the platform running the simulation code. When running simulations on thousands of processors of massively parallel hardware, license consumption alone is an issue of real concern. And in addition, such high-powered computing hardware is typically protected by firewalls and other perimeter defenses that restrict communication, and accessing a non-local CAD-engine may be difficult.

All of these situations require a CAD-free surrogate for the queries that the CAD system usually fields in CAD-based simulations. In this work we investigate the use of Loop subdivision surfaces as an underlying surface model. Loop subdivision surfaces [3] have been extensively studied over the past two decades in the fields of computer graphics, animation [10] and scattered data surface reconstruction [4]. They can be used to represent globally smooth locally manifold surfaces of arbitrary global topology. The surface is defined by a control-mesh which is a watertight triangulation (simplicial polytope) constructed using some existing discrete representation of a model. While subdivision surfaces have simple rules for updating existing node locations

and performing new insertions, our use of them for geometric modeling is enabled by Stam's exact evaluation procedure for Loop subdivision surfaces [6, 7].

Any mesh generation or mesh adaptation application uses geometry evaluation and inverse evaluation to place points on the entity of interest. These applications read and hold the geometry (usually in the form of a Boundary Representation – BRep) and may support a large number of curve and surface primitives. Some applications depend on a geometry kernel (or Direct CAD interface) to deal with the complexities of the individual entities. These kernels directly provide evaluation functions that have one degree of freedom for curves (t) and two for surfaces (u, v). In either case, vertices are placed directly on geometry by direct evaluation or “snaps” (inverse evaluation) which can be performed by the use of derivatives of the forward evaluation. When using a subdivision surface as the underlying geometry, Stam's evaluation procedure answers the same type of requests by providing geometry for arbitrary parameter values. In the theoretical development, we describe a simplified implementation of this procedure and algorithms for determining the first and second derivatives of the surface. The capability to construct surface geometry and its derivatives at arbitrary parameter values provides a very fast and lightweight modeler and gives a simple and uniform view of the surface (i.e. one type – triangles) that is locally manifold. Moreover, unlike classical tensor-product spline or BRep surfaces, this approach is independent of the global complexity of the surface and is not constrained to rectangular patches.

An approximating subdivision surface does not necessarily pass through the nodes of the control-mesh (it is not an interpolant of this node set). The literature in computer graphics and scattered data reconstruction highlights several approaches to make the surface interpolating, and displacement [5], multi-resolution [9] and adaptation [4] are addressed in our discussion.

2 Surface Parametrization by Subdivision

2.1 Subdivision Surfaces

The main idea behind subdivision surfaces is to represent a smooth surface by a *control-mesh*. This mesh can be enhanced by various refinement schemes, and in the limit of infinitely many refinements it approaches the “true” surface. In practice these refinements are repeated until the surface is sufficiently fine. This is the approach used in applications from computer graphics and visualization where subdivision surfaces are most commonly employed.

A distinction is made between *interpolating* and *approximating* subdivision schemes. In the interpolating schemes the nodes of the control-mesh stay fixed during the refinement. The resulting surface is then an interpolant of the

nodes in the control-mesh, which is an attractive property. However the surfaces generated with these schemes are sometimes not sufficiently smooth, and the convergence to the “true” surface is relatively slow. The approximating schemes, on the other hand, generally do not produce surfaces which interpolate the control-mesh, but they do result in smooth surfaces with continuous second derivatives everywhere except for at certain isolated points.

When considering the functions essential to a geometry kernel (when used for mesh construction and adaptation) there are two basic requirements. The first is the ability to evaluate, that is, given parameters within the geometric support produce 3D coordinates of the point on the entity. In the case of subdivision surfaces, the support is the control-mesh triangles and the parameters are the barycentric coordinates within each triangle. The second required function is the ability to find the nearest point on the geometry to a given set of coordinates. This “snap” (inverse evaluation) is usually cast as a minimization problem and is efficiently solved by the use of an iterative Newton solver. In order to construct this solution the forward evaluation is required along with both first and second derivatives at the evaluated point. The resultant position may not be in the current control-mesh triangle, which will be reflected in the barycentric coordinates (being outside the valid range). The neighboring control triangle opposite the vertex with the largest negative weight is set as the current support and the Newton solver is restarted within this triangle. This converges quickly and is robust requiring little additional intervention (but can be sensitive to starting locations when the geometry is concave).

In this work we are concerned about second derivatives, both for computing curvatures and for applying the Newton solver (as described above). Therefore our current focus is on approximating schemes. In Section 3.3 we discuss various alternatives for obtaining an interpolating model.

2.2 Spline Curves

For one-dimensional curves the subdivision schemes are particularly simple. The control-mesh is a polygon and the limiting procedure gives a smooth curve, see Fig. 1 for an example. At each refinement step, the polygon edges are divided in two. In this example we use an approximating scheme where the inserted midpoint at level $j + 1$ is simply the average of the two neighboring nodes at level j ,

$$\mathbf{x}_{2i+1}^{j+1} = (\mathbf{x}_i^j + \mathbf{x}_{i+1}^j)/2. \quad (1)$$

The original nodes are moved to a linear combination of their previous locations as well as their neighbors’,

$$\mathbf{x}_{2i}^{j+1} = (\mathbf{x}_{i-1}^j + 6\mathbf{x}_i^j + \mathbf{x}_{i+1}^j)/8. \quad (2)$$

The limiting curve is a cubic spline and it is C^2 continuous. We note that the subdivision rules (1) and (2) are local, and therefore a node in the

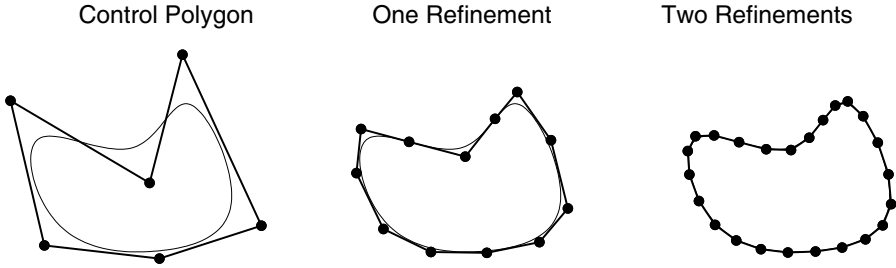


Fig. 1. Subdivision of a polygon using an approximate scheme. The limiting curve, which is the “true” smooth curve being represented, is shown with thin line.

control-polygon only affects the curve in a neighborhood around the node. As we pointed out above, the refined polygons will approach the “true” curve in the limit, but it is also possible to compute the limiting positions of the nodes at any level by the simple expression

$$\mathbf{x}_i^{j,\infty} = (\mathbf{x}_{i-1}^j + 4\mathbf{x}_i^j + \mathbf{x}_{i+1}^j)/6. \tag{3}$$

where $\mathbf{x}_i^{j,\infty}$ denotes the limiting position of point i at refinement level j .

2.3 The Loop Subdivision Scheme

For surfaces, various types of elements can be used for the control-mesh. While the original approximating subdivision schemes of Catmull and Clark [1] were based on quadrilateral meshes, we have worked exclusively with the Loop scheme [3] for triangular meshes. The control-mesh is defined by a set of nodes as well as a triangulation. In a refinement step, each triangle is split into four, and we need rules for the location of the new midpoint of the edges as well as the new location of existing nodes, see Fig. 2. The number of neighbors of existing nodes is k (the valance), and the value we use for the weight ω was proposed by Loop [3], see Warren [8] for a simpler alternative.

Again we can compute the limiting location for the nodes at any level of the refinement. The formula is the same as the one to advance one level (right plot in Fig. 2), but with ω replaced by $1/(k + 3\omega/8)$, see Fig. 3. We can also compute the tangent vectors (and from them the surface normal) using the expressions

$$\mathbf{t}_1 = \sum_{i=0}^{k-1} \cos \frac{2\pi i}{k} \mathbf{x}_i, \quad \mathbf{t}_2 = \sum_{i=0}^{k-1} \sin \frac{2\pi i}{k} \mathbf{x}_i \tag{4}$$

where \mathbf{x}_i is the i th neighbor of the considered node.

Figure 4 shows an example of a control-mesh with the limiting Loop subdivision surface. Note again that this is an approximate scheme and the nodes of the control-mesh are not located on the surface. The surface is C^2 continuous

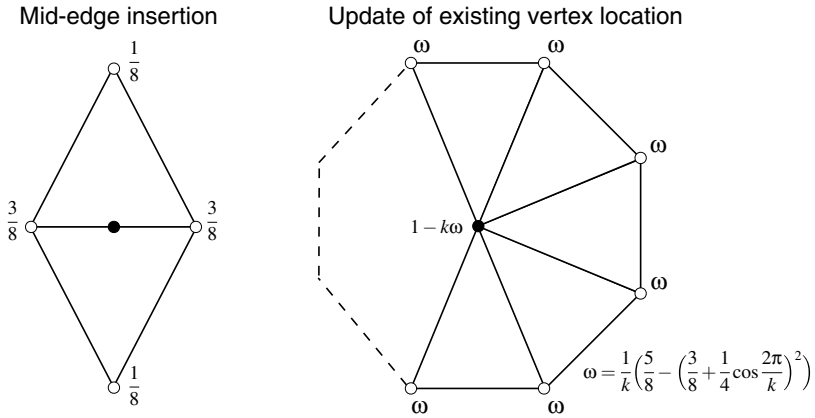


Fig. 2. The Loop subdivision scheme. The figures show the weights used to compute a new mid-edge point (left) and how to update existing nodes (right).

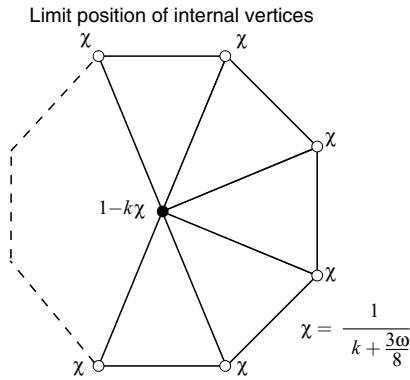


Fig. 3. The node positions in the limit of infinitely many refinements using the Loop scheme. This effectively moves the nodes of the control-mesh to the “true” surface.

everywhere except at irregular nodes (nodes that do not have six neighbors). The subdivision process converges fast and for visualization purposes a few refinements are sufficient to obtain a good approximation of the surface.

It should be noted that the subdivision scheme defines a hierarchy of “control-nets” each having the same limiting result. The left part of Fig. 4 results in the middle part after applying one subdivision operation. This set of vertices can be considered a new control-polygon and then results in the right part of Fig. 4 after another application of the operator.

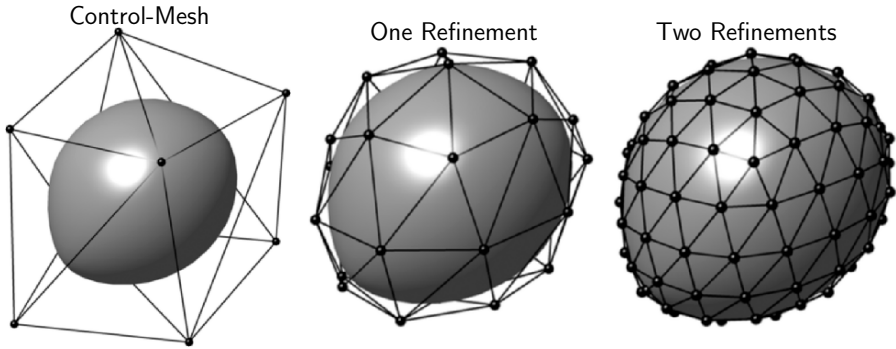


Fig. 4. Subdivision using the Loop subdivision scheme. The figures show the limiting surface together with the initial control-mesh, and two refined meshes.

2.4 Hard vs. Soft Edges

The Loop scheme produces smooth surfaces everywhere, including at the boundaries of the mesh (Fig. 5, left). Sometimes it is desired to have sharp boundaries (“creases”), which requires special treatment in the subdivision process. We tag the edges of the control-mesh corresponding to the sharp boundaries as *hard edges*. In the refinement of these nodes we use the one-dimensional spline scheme (1)-(3) instead of the Loop scheme. This will represent a smooth boundary curve with a jump in the tangent plane on the two sides of the curve (Fig. 5, middle).

In a similar way, the vertices of the boundaries can be tagged as *hard vertices* to produce a jump in the tangent along the boundary curve. Since a vertex can not be subdivided, the scheme for hard vertices is simply to leave them fixed at their original positions (Fig. 5, right).

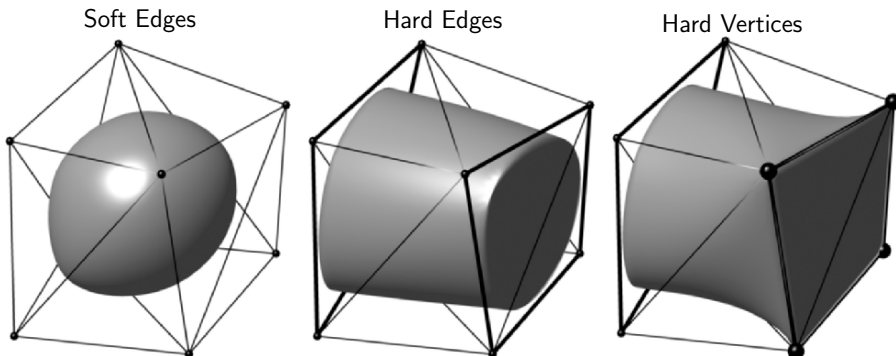


Fig. 5. Hard and soft edges and vertices. The hard edges are shown in thick lines and the hard vertices are shown with large spheres.

3 Parametrization and Evaluation

Using the subdivision scheme and the limiting expressions, we can evaluate the surface properties at the control nodes and at the nodes of any refined mesh. However, in order to be useful in a general geometry setting we need a parameterization of the surface and the ability to evaluate at arbitrary locations. There is not much literature on the parameterization of subdivision surfaces, presumably because the refinement process fulfills most all the needs in computer graphics and animation. Stam showed how to evaluate the Catmull-Clark surfaces for arbitrary parameter values [7], and later extended this analysis to Loop surfaces [6]. We use a simplified form of these methods, and we describe the parameterization for regular triangles in this section, while the next section discusses arbitrary triangulations.

The parameterization of spline curves are well-known, and for a “smooth” edge segment (no hard vertices or end points) we number the four consecutive polygon nodes $\mathbf{x}_{i-1}, \dots, \mathbf{x}_{i+2}$ and introduce a parameter value $t \in [0, 1]$ between node i and $i + 1$. The explicit expression for the spline curve is then:

$$\mathbf{x}(t) = [t^3 \ t^2 \ t \ 1] \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{i-1} \\ \mathbf{x}_i \\ \mathbf{x}_{i+1} \\ \mathbf{x}_{i+2} \end{bmatrix}$$

Similar expressions are available for edge segments neighboring hard vertices (interpolating control points), see de Boor [2] for details.

3.1 Parametrization – Regular Triangles

In a regular triangle, the subdivision surface reduces to the common box splines for which analytical expressions are available. Each node has exactly six neighbors and the total number of nodes in the triangle or adjacent to it is 12 (Fig. 6). A natural parameterization is given by the local barycentric coordinates in the triangle (right plot). Since $u + v + w = 1$ we can eliminate one of these coordinates, and we choose to parameterize by v, w .

For evaluation at the local coordinates v, w , we compute all monomials and collect in a column vector:

$$\mathbf{c}(v, w) = (1, v, w, v^2, vw, w^2, v^3, v^2w, vw^2, w^3, v^4, v^3w, v^2w^2, vw^3, w^4)^T \quad (5)$$

These basis functions are mapped to a Lagrangian basis by multiplication by the matrix ϕ below. We obtained this matrix from the expressions in [6] after substituting $u = 1 - v - w$, renumbering, identifying coefficients, and writing in matrix form.

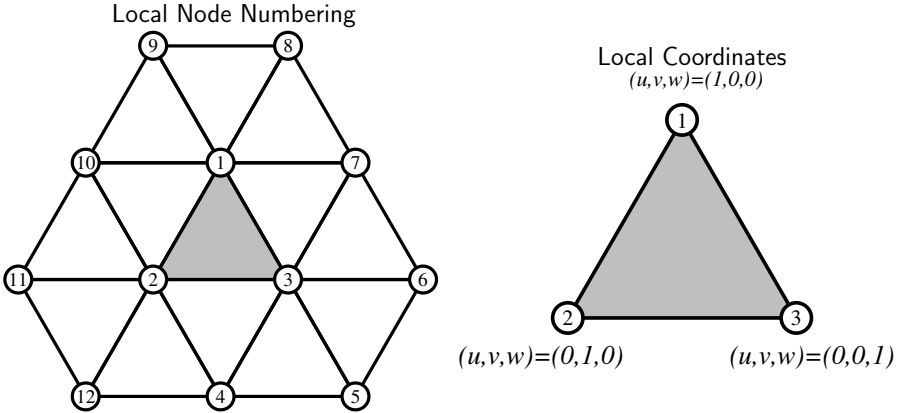


Fig. 6. The local node numbering for regular triangles (left) and the local barycentric coordinates u, v, w where $u + v + w = 1$ (right).

$$\phi = \frac{1}{12} \begin{bmatrix} 6 & 0 & 0 & -12 & -12 & -12 & 8 & 12 & 12 & 8 & -1 & -2 & 0 & -2 & -1 \\ 1 & 4 & 2 & 6 & 6 & 0 & -4 & -6 & -12 & -4 & -1 & -2 & 0 & 4 & 2 \\ 1 & 2 & 4 & 0 & 6 & 6 & -4 & -12 & -6 & -4 & 2 & 4 & 0 & -2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6 & 6 & 2 & -1 & -2 & 0 & -2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -2 & -1 \\ 1 & -2 & 2 & 0 & -6 & 0 & 2 & 6 & 0 & -4 & -1 & -2 & 0 & 4 & 2 \\ 1 & -4 & -2 & 6 & 6 & 0 & -4 & -6 & 0 & 2 & 1 & 2 & 0 & -2 & -1 \\ 1 & -2 & -4 & 0 & 6 & 6 & 2 & 0 & -6 & -4 & -1 & -2 & 0 & 2 & 1 \\ 1 & 2 & -2 & 0 & -6 & 0 & -4 & 0 & 6 & 2 & 2 & 4 & 0 & -2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \end{bmatrix} \quad (6)$$

Finally we create a node array with the 12 local nodes according to the numbering in Fig. 6 (a 3-by-12 matrix):

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}) \quad (7)$$

Using these expressions the surface location is a product of these matrices and the vector $\mathbf{c}(v, w)$:

$$\mathbf{x}(v, w) = \mathbf{x}\phi\mathbf{c}(v, w) \quad (8)$$

The first and second derivatives are obtained in a similar way by differentiating the simple monomials in $\mathbf{c}(v, w)$.

3.2 Parameterization – Irregular Triangles

The method Stam suggested for irregular triangles is based on the fact that the subdivision process introduces new triangles with regular neighborhoods

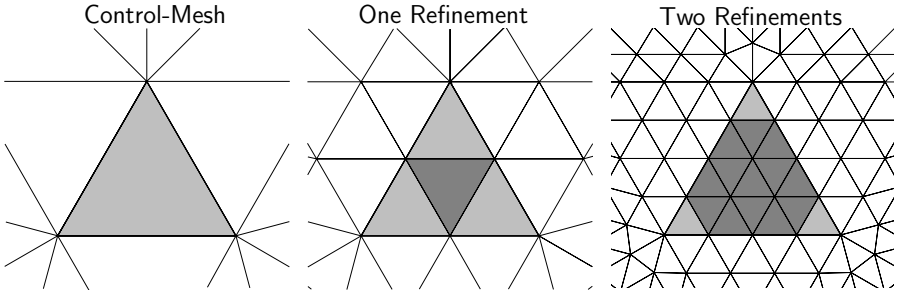


Fig. 7. Refinement of a triangle with irregular nodes. The triangle under consideration is shown in light gray, and the triangles with regular neighborhoods are shown in dark gray.

that can be evaluated using the box spline expressions (see Fig. 7). Clearly, after a sufficient number of refinements any point in the triangle can be covered by a regular triangle and evaluated using (8). However, for points close to an irregular node a very large number of refinements might be required. Stam solved this problem by introducing an eigen-decomposition of the refinement matrix, and was able to evaluate arbitrary close to irregular nodes in a constant number of operations. In our work, we choose a simpler solution in which we refine until the requested position is covered by a regular triangle or until some maximum subdivision depth is reached. If the maximum depth is reached (usually around 25 subdivisions), then we *nudge* the requested position onto the center triangle (which will be regular) and subdivide once again. At this point the evaluation using a regular neighbor can be applied. We use this special treatment for points close to the irregular nodes and in the vicinity of hard edges or vertices.

The crucial step of our scheme is to subdivide around the point v, w , identify the new triangle and the new parameter values v, w , and repeat recursively until the neighborhood is regular. During this refinement we use a local representation of the relevant triangles only. Figure 6 shows our node numbering for this local submesh. It is constructed by selecting a starting vertex in the triangle (this also sets the new v, w). The other two vertices are selected in a right-handed manner. The fourth vertex is selected as the node opposite the first. The rest are defined by winding around the triangle in a right-handed fashion and collecting all of the next level vertices that support any triangle that touches the target. This neighborhood is fully defined by the valance of each of the vertices of the target triangle (3 integers) and the positions of each of the neighborhood nodes, where the number of nodes is $k_1 + k_2 + k_3 - 6$. This small memory footprint does not overwhelm the stack as the recursive procedure continues.

The next level is reached by selecting the appropriate subtriangle (as seen in the center of Fig. 7) based on v, w . A new neighborhood is specified by

Algorithm 1: Subdivision Surface Evaluation

Description: Evaluate a subdivision surface for arbitrary parameter values

Input: Control-mesh, triangle t , local coordinates v, w

Output: Surface location \mathbf{x} and its first and second derivatives $d\mathbf{x}, dd\mathbf{x}$

```

function [ $\mathbf{x}, d\mathbf{x}, dd\mathbf{x}$ ] = loopeval( $t, v, w, depth$ )

if  $t$  regular
    Evaluate surface location and derivatives using (8)
else
    Subdivide  $t$  and all triangles sharing its nodes
    if  $depth \leq 25$ 
         $t' =$  new triangle covering  $v, w$ 
         $v', w' =$  new local coordinates in  $t'$ 
    else
         $t' =$  center subtriangle
         $v', w' =$  new local coordinates (on edge of subtriangle)
    end if
    [ $\mathbf{x}', d\mathbf{x}', dd\mathbf{x}'$ ] = loopeval( $t', v', w', depth + 1$ )
    Inverse mapping:  $\mathbf{x} = \mathbf{x}', d\mathbf{x} = \pm d\mathbf{x}'/2, dd\mathbf{x} = dd\mathbf{x}'/4$ 
end if
    
```

Fig. 8. High-level pseudo-code for evaluation of subdivision surfaces at arbitrary parameter values.

applying the subdivision rules (as seen in Fig. 4) where the *hard edges* are defined during the splitting of existing *hard edges*. *Hard vertices* need not be explicitly marked because they are defined when the number of *hard edges* touching a node is greater than two. The valence of the 3 subtriangle vertices, the neighborhood coordinates, as well as the new v, w are passed on to the next recursion level. Finally, the computed derivatives are adjusted because of the mapping between the new and the old parameters v, w . Our complete algorithm is described in pseudo-code in Fig. 8.

3.3 Generating an Interpolating Result

The fact that the Loop scheme is not interpolating might be a major problem in some applications. The subdivision schemes that are interpolating [10] produce less smooth surfaces than the Loop scheme, and we would therefore lose the ability to provide geometry “snaps” (solving with Newton’s scheme across the triangles). This would make the scheme unusable in our context.

An alternative method that we have used with some success is to solve for new node locations in the control-mesh such that the “true” Loop surface interpolates the original nodes. This simply amounts to solving

$$S^\infty(\mathbf{x}_{\text{interp}}) = \mathbf{x} \tag{9}$$

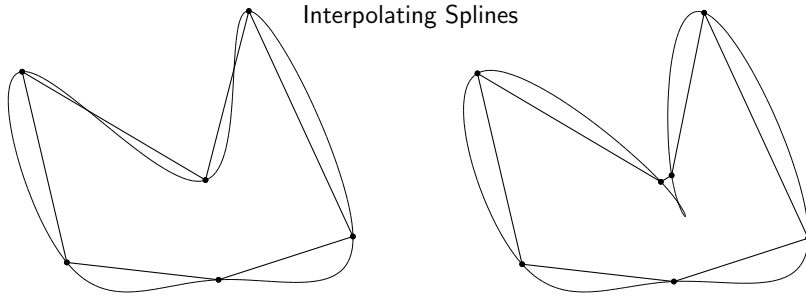


Fig. 9. Splines computed from new control points such that the curve interpolates the original points. The right plot shows that the resulting curve is not always well-behaved. In this case it has a cusp.

for the new nodes $\mathbf{x}_{\text{interp}}$, where \mathbf{x} are the original node locations and S^∞ the linear operator that produces the limiting node locations. For the Loop scheme, S^∞ is essentially the stencil in Fig. 3, except for hard edges/vertices. The linear system of equations (9) is well conditioned and can be solved in just a few iterations with a Krylov subspace solver.

Our initial experiments show that the surfaces generated by $\mathbf{x}_{\text{interp}}$ are well-behaved for uniform control-meshes, see left plot in Fig. 9 for an example of a spline. However, for more general control-polygons, the resulting curve might have cusps (right plot). A similar example for surface meshes using the Loop scheme is shown in Fig. 10.

One correction algorithm that shows more success is to post-process the results in a manner similar to that in [5]. By maintaining v, w and the parent triangle for any vertex, it is possible to adjust the position by applying the linear weights times the displacement of the control-net to the limiting surface. This insures that the adjusted surface passes through the control-net at the expense of C^1 . Approaches, such as multi-resolution [9] and adaptation [4] promise both smoothness and interpolation, and these are subject of on-going investigations.

4 Results

This work is aimed at investigating the utility of Loop subdivision surfaces as surrogate geometry. In order to demonstrate their use in this role, we have constructed a very simple surface mesh refinement application. Within this application, all geometry constructors (requests for new points on the surface) are based on the version of Stam's evaluation procedure described in Algorithm 1. We simply provide the local parameterization at the desired vertex insertion point, and this algorithm returns the xyz -triple for the constructed vertex. An analogous constructor is central to virtually all geometry/CAD-based mesh generation systems, and is the key ingredient in the use of Loop

Interpolating Subdivision Surfaces

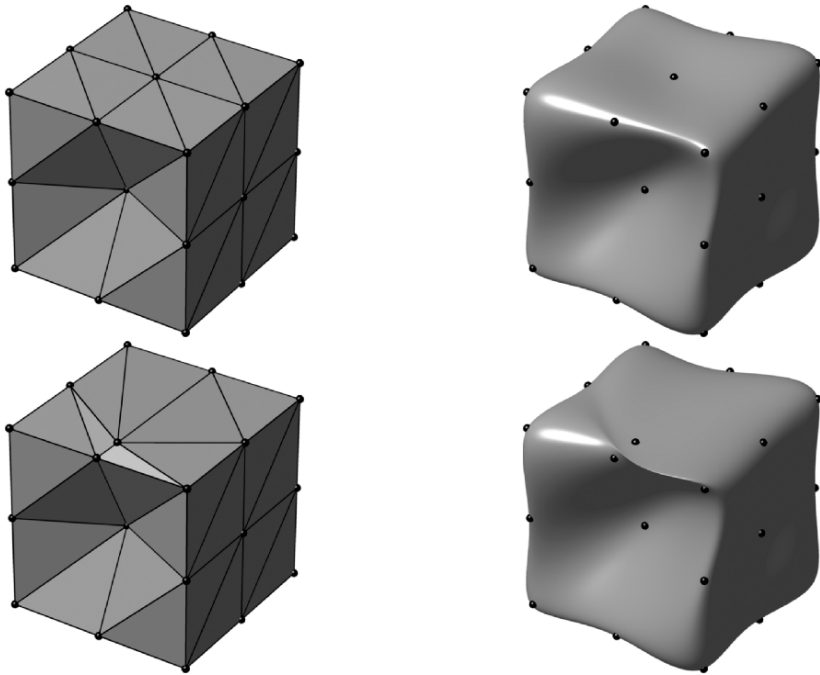


Fig. 10. Loop subdivision surfaces computed from new control points such that the surface interpolates the original points. The left plots show (original) control-meshes, and the right plots show the surfaces. The bottom example shows a (probably undesired) negative effect.

subdivision surfaces as a CAD-free modeler. Algorithm 1 also provides first and second derivatives of the surface at the evaluation location.

Aside from this central feature, the mesh refinement application demonstrated in this section is very simplistic and has relatively few features of merit. It was written simply as a platform for testing/demonstrating the constructor, and is not (in any way) intended as a viable mesh generator. Mesh refinement proceeds by performing a set of centroidal insertions within a set of triangles, followed by an edge-swap pass which performs swaps based upon the evaluation of a *maxmin* predicate. It is important to recognize that the tessellations shown are not generated in the traditional subdivision construction manner as so often seen in the literature.

Each example begins with the control-net for the underlying subdivision surface read in from a legacy triangulation file. No coarsening of the legacy triangulation is performed. For the purpose of demonstration, a simple dihedral-angle criteria on triangles in this control-net is used to establish hard edges.

Figure 11 shows the effect of hard edges and two example meshes generated using this adaptation application. The legacy geometry in this figure is the

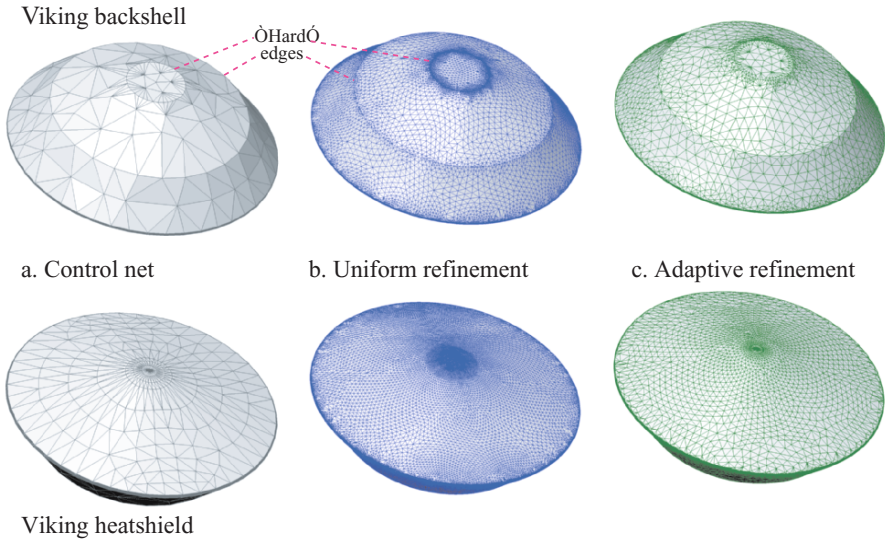


Fig. 11. Control-net, uniform and adaptive tessellations of the Viking spacecraft aeroshell geometry. The upper row of images shows the backshell (back) and the lower shows the heatshield (front). Both tessellations demonstrate preservation and refinement of hard edges in the input mesh.

aeroshell of NASA's Viking spacecraft which was the culmination in a series of exploratory missions to Mars from 1964-1975, and pre-dates much of the modern CAD industry. The upper row of images in Fig. 11 shows the view from the back, while the lower shows the geometry from the front. The back of the aeroshell is a truncated bi-conic. The control-net shown in Fig. 11a is composed of 1700 triangles and the dihedral-angle criteria identifies one circle of hard-edges at the sharp transition between the two conics and another at junction with the flat backface.

The simple mesher described above was run using both uniform refinement and curvature adaptive refinement. In the adaptive example, refinement was triggered using the maximum local surface curvature at each triangle centroid scaled by the triangle's area. Surface curvature was evaluated making use of Algorithm 1's ability to return the first and second derivatives at any point on the surface. The uniformly refined mesh has 35000 triangles while the adaptive triangulation has 9100. The triangulation algorithm produced approximately 7500 triangles-per-second on a 2Ghz CPU, however no serious attempt has been made to optimize the mesher or subdivision surface library.

While the Viking aeroshell is an example of legacy geometry that pre-dates CAD-based modeling, Fig. 12 shows an example of geometry that comes from a 3D scanner and similarly has no underlying CAD definition. The figure shows the control-net (45k triangles) and a curvature adapted triangulation (105k triangles) for an irregularly shaped piece of foam. The 3D scan produced a

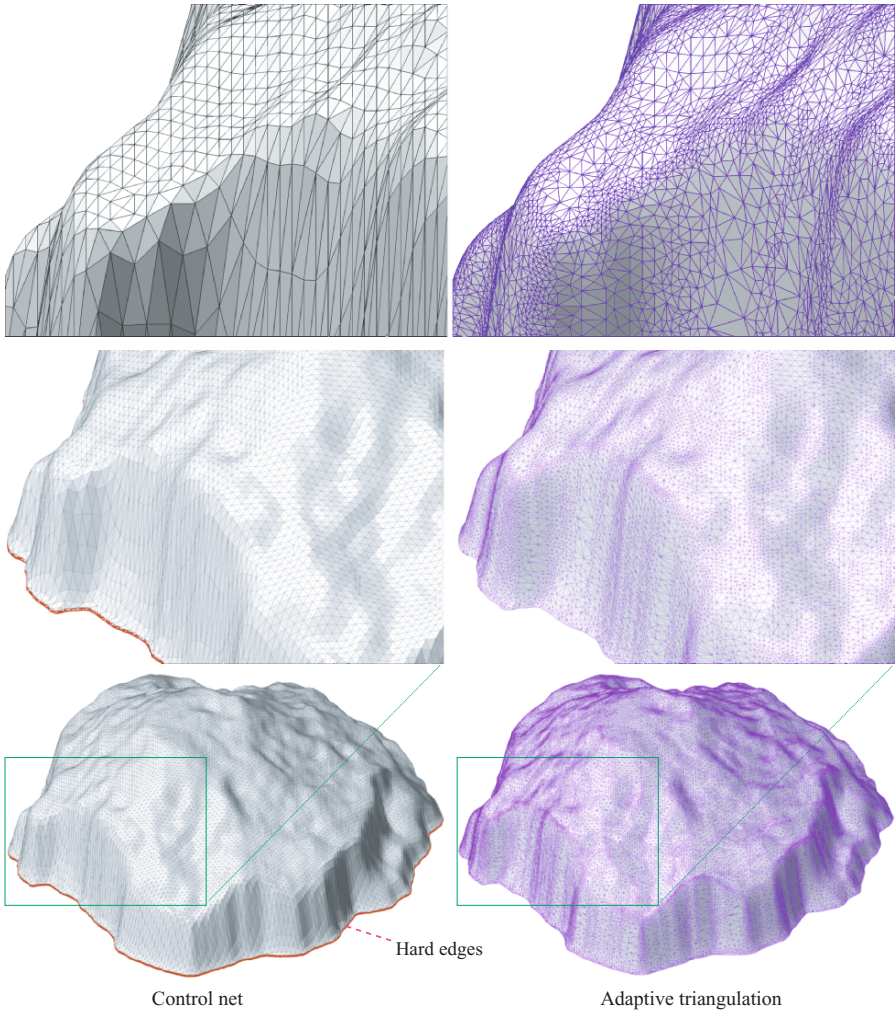


Fig. 12. Control-net (45k triangles) and curvature-adaptive triangulation (105k triangles) for irregularly shaped foam piece from high-resolution 3D scan. Inset frames show details of control-net and surface triangulation.

STL triangulation file and this triangulation (without decimation) was used as the control-net for the subdivision surface. The simple adaptive mesher used in the previous example was then run using this surface as an underlying geometry. While the original scan is at quite high resolution, the enlargements of the control-net shown in Fig. 12 (inset top-left) exhibit substantial faceting due to the small characteristic feature size on the irregular surface. The adaptive triangulation shown at the right of Fig. 12 offers improved resolution of regions with high curvature. Hard edges are indicated on the figure and form a single closed loop around the lower perimeter of the piece.

Figure 13 shows a final example with the control-net and adaptive triangulation generated on a model of a pig. The control-net in this case is quite coarse with only about 7000 triangles. Despite this, the model is remarkably detailed and includes all major anatomical features as well as details like eyelids, hoofs, nostrils, and jowls. These features are even more apparent in the adapted triangulation shown in the center and two inset frames of Fig. 13. This adapted triangulation was built using the simple mesher described earlier with curvature-sensitive refinement and includes 244k triangles. In addition to the obvious features, refinement reveals more subtle detail captured by the control-net. The refined triangulation reveals the pig’s shoulder blades, pelvis, hamstring definition, and additional facial detail. As with the earlier cases, the goal of this example is not to show the “best” triangulation for this geometry, but to illustrate the use of this very coarse legacy triangulation as a control-net definition and to mesh using the Loop subdivision surface defined by that control-net.

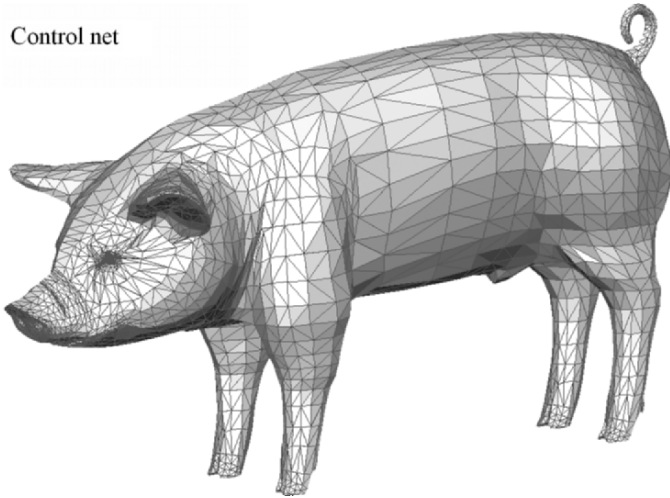
5 Conclusions

This work outlined a method for using existing surface triangulations as underlying geometric models for the construction of more highly refined models complete with local surface derivatives and curvature information. The method is based upon the construction of a Loop subdivision surface using the legacy triangulation at full resolution. The presentation outlined a simplified surface evaluation procedure based on Stam’s exact method for Loop surfaces. The presentation of this simplified approach highlighted the construction of surface coordinates, derivatives, and curvatures for evaluations at arbitrary locations on the surface. The implementation also provides the ability to tag *hard-edges* (and therefore *hard-vertices*) in the imported geometry to preserve creases and points using one-dimensional cubic-spline scheme which preserves C^2 continuity along hard-edges. Away from hard-edges and vertices, the Loop surface is C^2 continuous everywhere except right at irregular vertices in the control-net, where it is still C^1 .

The modeler was demonstrated using a simplistic meshing application which queries arbitrary locations on the surface to support either uniform or curvature-adaptive triangle refinement. Curvature information for adaptation parameter was obtained through direct evaluation on the subdivision surface using the algorithm presented in section 3. Examples were presented using control-nets built from a variety of legacy triangulations with widely varying complexity. The ability to query the surface at arbitrary locations and quickly find surface derivatives and curvature information makes this modeler very attractive for users of finite element analysis methods that require this information to achieve higher-order accuracy.

The simple evaluation rules for surface coordinates and derivatives make the modeler extremely fast and robust. Even with no special effort to optimize

Control net



Adaptive triangulation

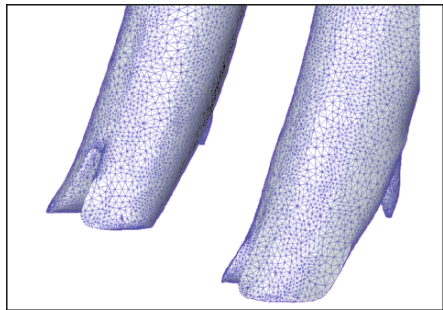
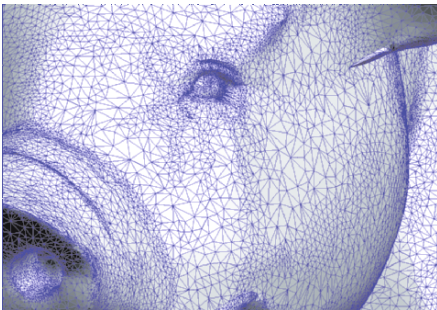
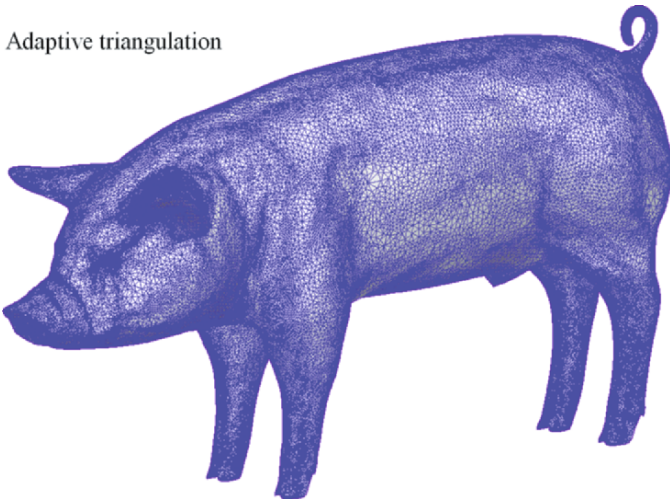


Fig. 13. Control-net and curvature-adaptive triangulation for pig geometry. The control-net contains about 7000 triangles, while the adaptive triangulation contains about 244k. Inset frames at the bottom show details of facial structure and hooves.

the implementation, surface triangulations with nearly 500k triangles can be generated in under a minute on a single processor desktop machine. Immediate plans will focus on experiments with the construction of the subdivision surface itself. Since the input triangulation becomes the control-net of the subdivision surface, it is not, in general, an interpolant for the input data. However, the deviation is small when the control-net resolves the geometry and hard edges are tagged appropriately. For example, the distances between the control-net nodes and the subdivision surfaces in the three examples in Section 4 are in average less than 0.06% of the geometry size, and the maximum deviations are less than 0.2%. Research examining various approaches for making the surface exactly interpolating is ongoing.

Acknowledgements

The authors wish to thank the Boeing Company (technical monitor Mori Mani) for their support in this effort. This assistance has enabled the generation of a surface modeling geometry software kernel as part of **TURIN** – the **T**etrahedral **U**nstructured **R**emeshing **I**Nterface. This software library was used to generate the figures seen in the results section.

References

1. E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.
2. C. de Boor. *A Practical Guide to Splines*. Springer, 2001.
3. C. T. Loop. Smooth subdivision surfaces based on triangles. Master’s thesis, Department of Mathematics, University of Utah, August 1987.
4. M. Marinov and L. Kobbelt. Optimization methods for scattered data approximation with subdivision surfaces. *Graphical Models*, 67(5):452–473, 2005.
5. A. Lee H. Moreton and H. Hoppe. Displaced subdivision surfaces. In *ACMSIGGRAPH ’2000 CDROM Proceedings*, pages 85–94, 2000.
6. J. Stam. Evaluation of loop subdivision surfaces. In *SIGGRAPH ’98 CDROM Proceedings*, 1998.
7. J. Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Computer Graphics Proceedings, Annual Conference Series*, pages 395–404, July 1998.
8. J. Warren. Subdivision methods for geometric design. Unpublished manuscript, November 1995.
9. D. Zorin. *Subdivision and multiresolution surface representations*. PhD thesis, Caltech, Pasadena, California, 1997.
10. D. Zorin and P. Schröder. Subdivision for modeling and animation. *SIGGRAPH 2000 Course Notes*, 2000.

Surface Mesh Generation for Dirty Geometries by Shrink Wrapping using Cartesian Grid Approach

Y. K. Lee¹, Chin K. Lim², Hamid Ghazialam³, Harsh Vardhan⁴ and Erling Eklund⁵

¹ Fluent USA Inc., Evanston, IL 60201 ykl@fluent.com

² Fluent USA Inc., Austin, TX 78746, ckl@fluent.com

³ Fluent USA Inc., Lebanon, NH 03766, hsg@fluent.com

⁴ Fluent India Pvt Ltd., Hinjewadi, Pune 411057, India, hxv@fluent.co.in

⁵ Fluent France S.A., 78180 Montigny le Bretonneux, France, erling@fluent.fr

Summary. A Cartesian shrink wrapping technique has been investigated in this study to construct triangular surface meshes for three-dimensional dirty geometries. The geometries dealt in this paper are defined by faceted representation with dirtiness such as non-conforming edges, gaps and overlaps. The objective of the proposed technique is to deliver a way constructing triangular surface meshes for upstream solutions in design processes without extensive labors for healing dirtiness in complicated dirty geometries. A Cartesian grid is overlaid onto the dirty geometries and its cells are adaptively refined until target resolution is achieved while recording intersections with geometric facets in cells. An initial watertight shell called the wrapper surface is constructed by selectively extracting the boundary sides of intersected cells. The wrapper surface is improved by a subsequence of operations such as projecting nodes onto geometry, adjusting nodes on the geometry and editing local triangular faces to achieve better approximation. The meshes generated using the presented technique may not be geometrically accurate but their quality is good enough to quickly deliver upstream fluid analysis solutions with significantly reduced engineering time for problems of extreme complexity such as the full underhood fluid/thermal analysis for automobiles. Mesh generation experiments have been carried out for complicated geometries and results from some applications are presented in this paper.

1. Introduction

Automatic mesh generation has become an essential tool for the finite element or finite volume analyses of practical engineering problems. The geometries of the problems are defined as CAD data and access to information stored in CAD data can be provided either through geometric

modeling kernel or exporting the data in a format loadable into the target system [BWS03]. The IGES and STEP are most commonly used protocols for exchanging CAD data from one CAD system to the other system. Even though the STEP normally delivers better translation results than IGES does by providing information with representations and global tolerances, immigration of CAD data from one system to the other often results in dirty geometries containing gaps, holes, overlaps and non-conformal edges which do not exist in the native data. The inconsistency of tolerant modeling methods in two systems is one of major reasons causing dirtiness in the imported geometry. Another source of dirty geometries often ignored is the urgent need of upstream solutions in practice. In such cases, engineers have to carry out simulation even before all the parts in the model have not been designed by leaving small voids or using similar parts in their legacy library.

Imbedding the mesh generation system in CAD systems can be a way to avoid the dirtiness from happening, since the meshing operation can be applied on the native data without translation. The cost for imbedding is extremely expensive and it cannot be done often due to issues other than technical ones. An alternative for imbedding is to develop the mesh generation system inter-operatable with CAD systems. The development cost for this framework is lower than that of imbedding. However, this approach needs both the CAD and the mesh generation system to be available locally. In addition, the CAD data and the mesh data should be managed separately afterward.

Most conventional approach is to heal the dirtiness by directly editing geometric entities of dirty geometries. Manual healing for dirty geometric models is very labor intensive and time consuming as engineers should destructively replace the old geometric entities using sophisticated reasoning. An investigation was carried out for detail suppression using topological modifications and the concept of virtual topology was suggested to define topologies of problems using underlying dirty geometries without extensive editing [SBC97]. Yet cleaning up dirty geometries requires considerable user interactions.

While the previous algorithms intend to fix dirtiness in CAD models, the scope of this study is focused on generation of meshes without altering the input geometries. A major bottleneck from dirty geometries to 3D fluid simulations lies in constructing watertight surface meshes as the subsequent tetrahedral volume mesh generation is relatively straightforward. The Cartesian grid approach and shrink wrapping technique are employed to tackle the problem in the present study. The shrink wrapping approach was proposed by Kobbelt et al. [KVL99] In their approach, a plastic membrane is wrapped around an object and shrunk either by heating the material or by evacuating the air from the space in between the membrane and

the object's surface. Theoretically, the plastic skin provides an exact imprint of the given geometry at the end of process. The problems in applying their technique to dirty geometry meshing are, first, the construction of wrapping membrane surface is difficult and, second, the projection operator may not always provide reliable results to compute traction and relaxation due to the nature of dirtiness. Thus, the Cartesian grid approach is introduced to construct the wrapping (or wrapper) surface and geometric modification on the surface is taken to improve closeness of the wrapper surface to the given geometry.

The Cartesian grid generation is a well-established technique. Basically, it overlays an axis-aligned structured grid onto the geometry of a problem and takes a part of the grid, which is in the region of interest. Later, the nodes near the input geometry can be repositioned or local structure of mesh can be modified. The octree technique for constructing tetrahedral meshes can be classified into this technique as it checks intersections between the cells in the tree and the input geometry and refines each cell in the region of interest based on intersecting pattern [She85]. Schneider is an early investigator with full hexahedral elements for 3D volume mesh generation [Sch95]. Aftosmis and his colleague presented a technique clipping geometric facets with Cartesian cells and using such information in fluid simulations [ABM97] and an extended technology has been used in extracting large scale models such as building geometries using shape profiles [WCG05]. Wang extended the application of this technique to dirty geometry cases [ZF02]. He pointed out that the Cartesian mesh generation is tolerable to dirtiness of geometries as long as the region of interest can be classified. And the zigzag boundary of the meshes can be improved using the best approximation available. There have been active development works to exploit such technology in extremely complicated applications with many components and some dirtiness in industrial field [CDA06, CEI06, CFD06] but few articles have been published in the open literature. More recently, Boschhoff and Pavic carried out an extensive research for extracting clean surface meshes from architectural models containing penetrating and touching components [BP05].

2. Outline of the Proposed Algorithm

The major difficulty in generating 3D volume meshes for dirty geometries for fluid simulations with traditional mesh generation tools is to make the dirty geometry clean so that the meshing tools can be applied step by step, for instance, from edge meshing to surface meshing and, then, to volume meshing. The main idea of the technique presented is (1) to construct a

watertight surface mesh for a dirty geometry by extracting a closed surface representation from underlying Cartesian grid and (2) improve the surface mesh while maintaining water-tightness for better approximation to the original geometry. The watertight faceted representation (or surface mesh) may not precisely represent geometric details of the model but, at least, provides a quality surface mesh for the subsequent tetrahedral volume mesh generation and, eventually, the upstream fluid simulation. The two phases can be concisely summarized as follows.

Overlaying Cartesian grid and extracting wrapping surface: A simple Cartesian grid of small number of cells is overlaid to the input geometry. The cells in the grid are refined until all of them satisfy criteria. The refinement for Cartesian cells is carried out while checking intersections between cells and the input geometry. Given the intersecting cells, all the non-intersected cells are classified into regions bounded by the intersecting cells. By collecting outer front of intersecting cells, it is possible to construct a watertight surface, so-called the wrapper surface, for each region.

Modifications of wrapper surface: In general, the initial wrapper surface represents the topology of a volume collectively defined by the input surfaces. However, the detailed geometry of the wrapper surface is far different from that of the input volume. The wrapper surface is modified by adjusting nodal positions or editing local connectivity of nodes to get better geometric approximation to the input geometry.

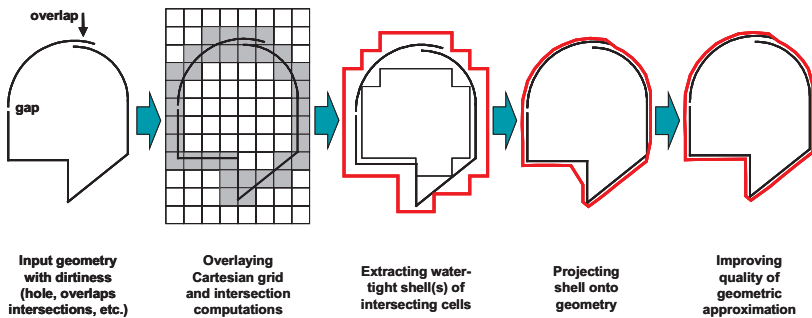


Fig. 1. Schematic description of wrapping procedure

As briefly demonstrated in Figure 1, it is always possible to construct a watertight surface approximating the volume boundary by extracting the connected boundary faces of the intersected cells. The typical case in which the previous statement is not true is when a cell completely falls between gaps, so-called invisible gap [WS02]. However, geometries in practice are reasonably well defined and the failure would not happen if cells around gaps are larger than gap distance.

3. Initial Wrapper Surface Generation

The first phase of the current technique is to construct a rough approximation for the input dirty geometry, the wrapper surface. This section describes how to generate the initial wrapper surface using the Cartesian grid technique.

3.1 Initial Cartesian Grid Generation

The faceted representation such as the stereolithography (STL) format is used to define the input geometry in this study. The STL format is popular due to its simplicity and portability. The facets in the geometry are stored in a search tree to be used for the intersection checks and projection in this study.

A uniform Cartesian grid is overlaid on the input geometry and its cells are adaptively refined later. The Cartesian grid is defined with cells and faces. In this paper, cells are defined with their six faces, locations and dimensions. Each face records its left and right neighbor cells. They also store references to child faces in case of refinement. Vertices are not used because they are not necessary [ABM97].

3.2 Adaptation of the Grid

Starting from the initial grid, an adaptive grid is constructed by gradually refining cells until all cells satisfy give size criteria. The size functions [ZBS02] are used to define the desired local sizes. While the curvature size function generally regards quality of geometric approximation, the proximity size functions may result topological difference of surface meshes obtained. For example as shown in Figure 2, insufficient refinement between gaps in a single connected simple geometry results a double connected representation. The problem is resolved only after two more refinement for cells (see Figure 2(b)). In 2D cases, cells of size $2G/\sqrt{2}$ are required to ensure that the gaps are resolved regardless to orientations and translations. Similarly, the factor becomes $2G/\sqrt{3}$ in 3D cases. The maximum difference of refinement between neighboring cells is limited to 1 both for simplicity in the data structure and smooth transition of cell sizes.

3.3 Extracting Initial Wrapper Surface

In the adapted Cartesian grid, there may be several groups of contiguous non-intersecting cells, the *regions*. A region may represent a virtual volume. In complicated models, there are multiple virtual volumes and geometric complexity often results exceptional cases.

Let us paint cells intersecting the input geometry in a simple grid shown in Figure 3(a). Naturally, those cells form virtual walls which separate non-intersecting cells into regions. Starting from a side of an intersecting cell on the exterior region, a closed shell can be extracted by following neighboring boundary sides of intersecting cells (see Figure 3(b)). There might be exceptional cases as shown in the interior regions causing non-manifold connection and fictitious regions. The intersection status of some cells should be manipulated to resolve such cases.

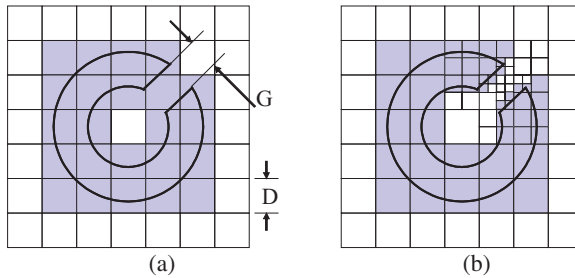


Fig. 2. Proximity size function to resolve gaps

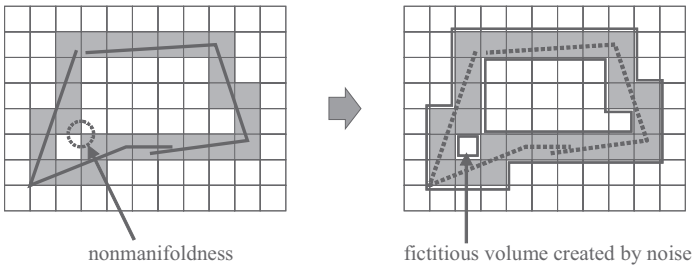


Fig. 3. Nonmanifoldness and fictitious region

4. Improvement of Wrapper Surfaces

In general, the initial wrapper surface extracted from the Cartesian grid topologically represents the virtual volume that is desired to recover. However, its geometric details, the zigzag configurations, are far different from the desired geometry. The remaining procedure of the developed technique is to improve such poor geometric approximation that it represents the input geometry better. This is done by adjusting node positions and locally editing triangular elements.

4.1 Projection of Nodes

The first step for improving the wrapping surface is to move nodes of the zigzagged wrapper surface onto the input geometry. The simplest way is to project the nodes onto the nearest points on the input geometry. However, the nearest point projection may not give desired results as schematically shown in Figure 4.

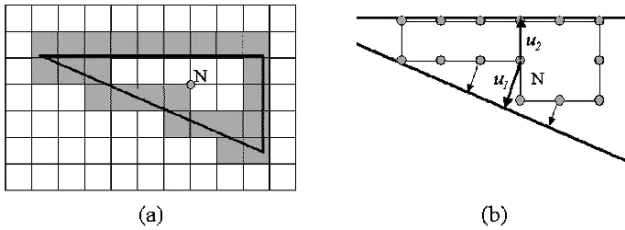


Fig. 4. Abnormal projection of node

Due to the nature of the current approach, it is obvious that all the nodes are all inside of the volume. (Or all nodes should be outside of the volume if the exterior wrapper surface was taken.) Thus all nodes should be projected to either outward or inward. An approximated normal vector, $\tilde{\mathbf{n}}_v^N$, at a node, N , is computed as

$$\mathbf{n}_v^N = \frac{\sum_i \mathbf{n}_i^F}{\left\| \sum_i \mathbf{n}_i^F \right\|}, \quad \tilde{\mathbf{n}}_v^N = \frac{\mathbf{n}_v^N + \sum_j \mathbf{n}_j^N}{\left\| \mathbf{n}_v^N + \sum_j \mathbf{n}_j^N \right\|} \tag{1}$$

where \mathbf{n}_i^F is a normal vector of its adjacent face and \mathbf{n}_v^N is the averaged normal vector of them. When a node is found to have the sign of inner product of its approximated normal vector and projection vector that is inconsistent to those of its neighboring nodes, some preconditioning such as

a weighted Laplacian smoothing should be done to prevent invalid projection.

In many practical cases, there are distinctive features on the boundary such as sharp edges and boundary between two zones. Often, it is highly desired to restore such features in the wrapper surface. In addition to the projection onto the surface, nodes in the vicinity of feature curves in the geometry are projected onto the feature curves. For a given feature curve, the closest nodes from the two ends are found on the wrapper surface and a path between the nodes is traced by using the feature curve as a guide. This procedure is basically traveling through mesh edges from one end node to the other while comparing distance to the feature curve and dot product of the edge vectors and the feature curve tangential vector. The tracing may fail to identify a reliable path and further investigation is undergoing. Even after identifying a reliable path, it is often observed that the compulsive projection of the nodes onto the feature curves deteriorates the configurations of neighboring faces. The projection of nodes is carried out in an incremental manner while checking validity of neighboring faces.

4.2 Editing Local Connectivities

In general, a finer initial wrapper surface provides better approximation for the input geometry after the projection. However, there may be very slender and skewed triangles. A set of standard local modification operations for triangles is used to improve such undesirable configurations further.

Edge collapsing: An edge that is unacceptably shorter than its neighboring edges is removed by merging its two end nodes as shown in Figure 5 (a). Also, this operation is used to coarsen surface meshes [GH97].

Edge splitting: If an edge is too long comparing to its neighboring edges, then a new node is introduced at its center and its two adjacent faces are divided into four triangles in Figure 5(b). The edge splitting is also used to improve skewed triangles.

Edge swapping: The edge swapping shown in Figure 5(c) can be very effective to improve the closeness of the faces to the geometry as well as to improve skewness of slender triangles.

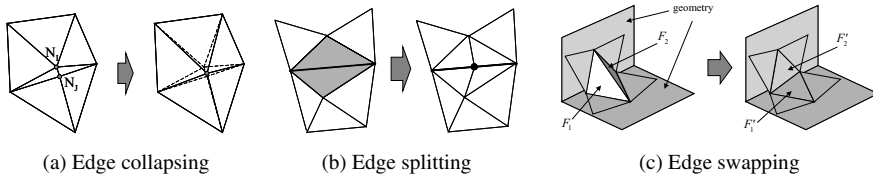


Fig. 5. Local mesh editing

4.3 Node Smoothing

Mostly, a weighted Laplacian smoothing [Her76] is used to improve skewed triangles by repositioning nodes. The nodes are projected back to the input geometry afterward. When a large number of iterations were applied to improve triangular quality, excessive smoothing deteriorates details of configurations and makes projection of nodes back to the geometry harder by moving nodes too far from the geometry. In such cases, enhanced smoothing techniques preserving features and volumes can be very effective [Tau95, VMM99, ZF02].

4.4 Zone Partitioning

The wrapped surface can be separated into several regions based on the underlying surfaces. The closest input surfaces are checked for every faces and faces of same corresponding input surface are grouped and separated into a zone. This is particularly useful when boundary conditions should be applied for certain zones in the fluid simulations later. In many cases, there are many small fictitious zones after initial separation due to the fact that the wrapper zone boundaries do not exactly follow those of input surfaces. In such cases, the faces in small islands are redistributed to neighboring larger zones.

5. Mesh Generation Examples

The presented algorithms are implemented in TGrid, a preprocessor for FLUENT solver. The developed mesh generator has been exercised on a set of example problems and the results are presented in the following sections. The presented examples include a simple geometry to graphically demonstrate the meshing procedure as well as fairly complicated geometries having intersecting facets to validate effectiveness of the proposed technique.

5.1 Transport Aircraft: General Wrapping Procedure

A relatively simple example is taken in this section to demonstrate the typical mesh generation procedure developed.

The model is an artificial transportation aircraft which is composed of five separate parts – fuselage, two wings and two sheet metal parts joining the fuselage and wings. An initial Cartesian grid is overlaid to the input geometry shown in Figure 6(a) and the configurations of intersected cells can be shown as in Figure 6(b). The connected exterior faces of the cells were extracted and the faces were triangulated based on predefined pattern in terms of hanging node configurations. Figure 6(c) shows the initial shrink wrapped surface after projection. The configuration in Figure 6(c) was improved using the smoothing, local mesh editing, etc. The final mesh is presented in Figure 6(d) after the zone partitioning.

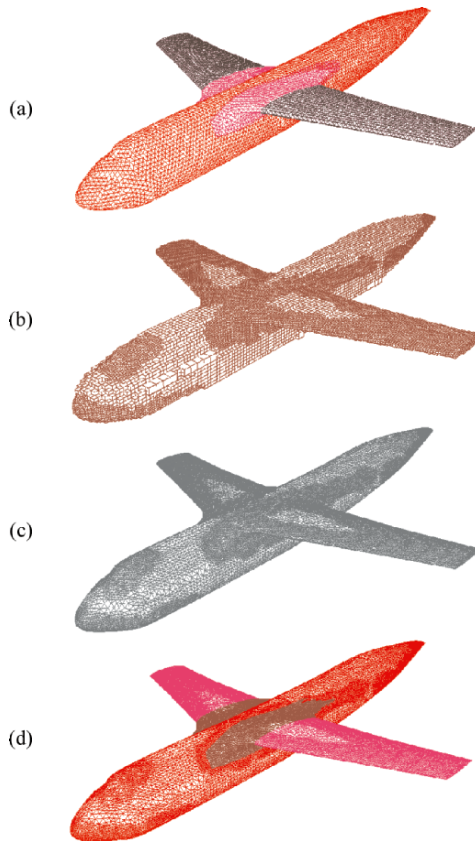


Fig. 6. Typical mesh generation procedure. (a) Input geometry; (b) Intersected Cartesian cells; (c) Initial wrapper surface; (d) Final wrapper surface after improvements

The following figure shows the minor dirtiness of the model. The part joining the wing and the fuselage does not share identical nodes and edges with either parts and some portion is penetrating into the fuselage.

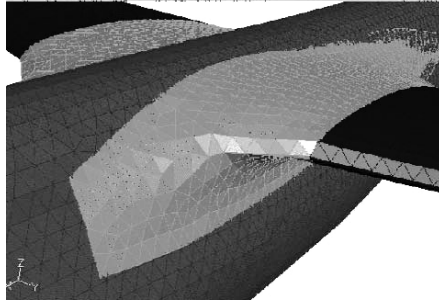
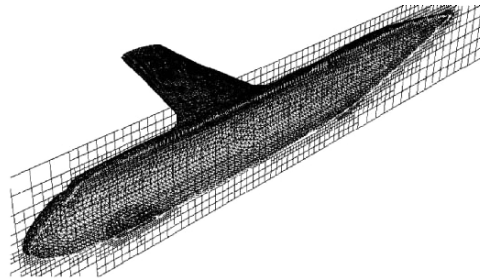
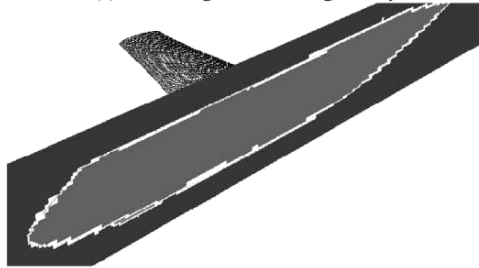


Fig. 7. Dirtiness in aircraft model



(a) Cartesian grid lines with geometry



(b) Highlighted intersected cells and inside and outside regions

Fig. 8. Cartesian grid lines on cutting plane

Figure 8(a) displays the gridlines of the underlying Cartesian grid on a cutting plane. The white lines are on cells intersecting the input geometric facets. The regions of different shades in Figure 8(b) represent that they are in separated regions bounded by the intersected cells shown as the white region.

In the existence of distinctive edges, so-called feature lines, in the initial geometry, such features can be restored in the final mesh by projecting nodes onto the feature lines as shown in Figure 9.

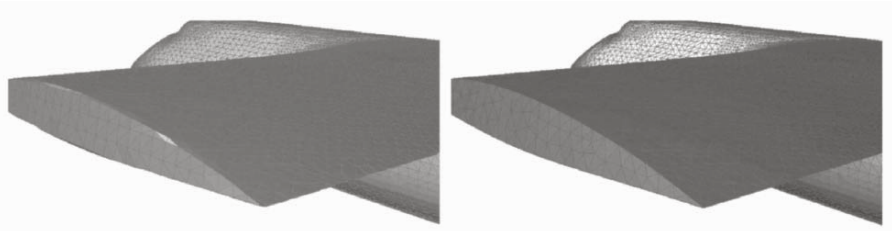
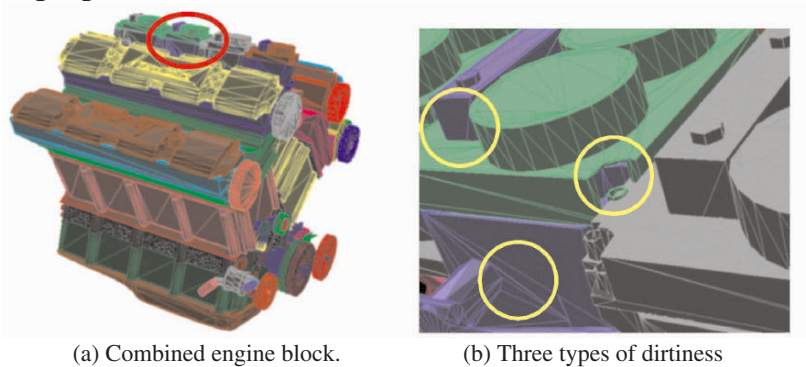


Fig. 9. Feature line recovery

5.2 Combined Engine Block

The second example in this paper is a V8 engine (see Figure 8(a)). The model contains three types of typical dirtiness to be dealt in dirty geometry meshing, intersections by penetrating parts, holes due to missing parts and unreliable facets due to poor triangulation common in STL files. Figure 8(b) highlights such dirtiness in circles.



(a) Combined engine block.

(b) Three types of dirtiness

Fig. 10. V8 engine

In general, any holes whose sizes are larger than cells in the Cartesian grid may result leakages in wrapping procedure. Thus such holes are desired to be filled prior to the wrapping operation. However, it is almost impossible to identify such holes in many complicated models in practice. If a model is wrapped with leakages due to such holes, the resulting wrapped surface is folded at a certain location, where the leakage occurs and the wrapper surface propagates into the inside of the volume. In such case, a

pair of faces can be chosen and the path between them can be tracked as shown in Figure 11 and the location of the hole should lie on the path. The hole should be resolved by adding additional facets manually.

Figure 12 shows a contour plot displaying the distance of each face center to the underlying input geometry. This plot can be exploited to quickly identify locations where higher resolutions with smaller cells are required if necessary.

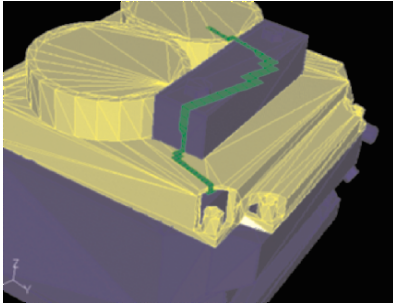


Fig. 11. Hole identification with path tracking

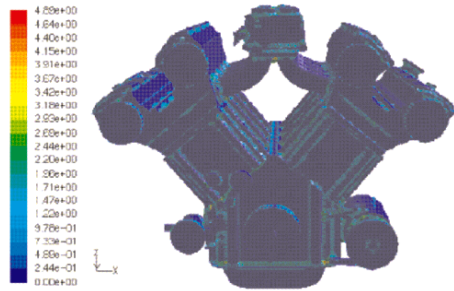


Fig. 12. Distribution of distance between face centers and underlying geometry

Figure 13(a) illustrates noisy zigzag configurations along the zone boundaries after partitioning mentioned in the section 4.6. The difficult cases are to recovery smooth boundary when there is no explicit boundary between the two zones in the underlying geometry, for instance, a pair of parts penetrating each other. In such cases, the nodes on the zigzag boundary can be projected onto both parts alternatively and the boundary forms a smooth line as shown in Figure 13(b).

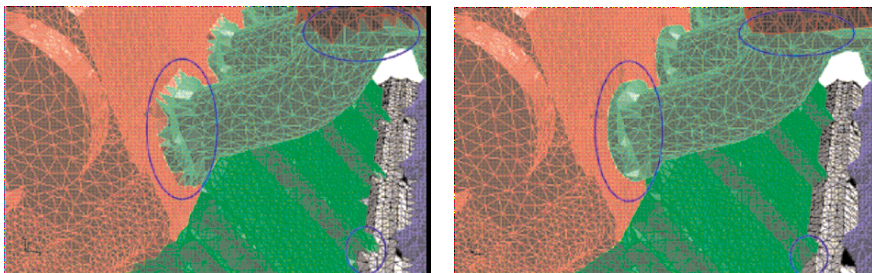


Fig. 13. Zone partitioning and boundary recovery

The CPU time complexity for Cartesian grid generation and region separation is shown in Figure 14 with a linear trend line. The experiment was done using a Linux machine with 2 Pentium 4 (3.4GHz) processors. The CPU times spent for subsequent modifications such as smoothing, swapping and coarsening were excluded as they are interactive operations which are triggered one after the other. From the latest refinement stage of 8,882k cells, 4.12 million triangles were extracted into the initial wrapper surface and it took 298 seconds to extract those triangles, incrementally project onto the input geometry and improve severely skewed triangles by edge collapsing and swapping.

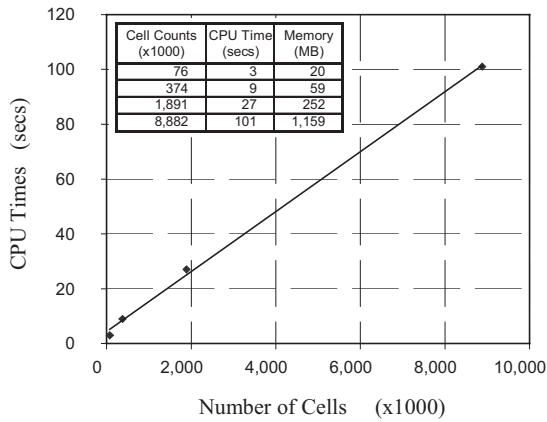


Fig. 14. CPU time complexity for Cartesian grid generation and region generation

5.3 Meshing For Underhood and External Aero-Simulation

The example in this section is taken to illustrate an application of the developed mesh generator for a model of industrial complexity and to discuss the turnaround time in industrial applications. The model for a whole truck body is presented in Figure 15(a) and it contains more than 1250 assembly parts including ones in underhood shown in Figure 15(b). The original geometry contains 473156 facets with 237761 vertices.

Several resolution levels have been tested and numerical experiment carried out using a AMD 64bit Opteron machine shows that the developed mesh generator takes 3.5 GB for 10 million cells and spends 1 hour to generate 60 million cells with intersection checks. The truck body was placed in an artificial wind tunnel and wrapped with it. The final wrapper surface consists of roughly 2 million faces and 1 million nodes after coarsening. It

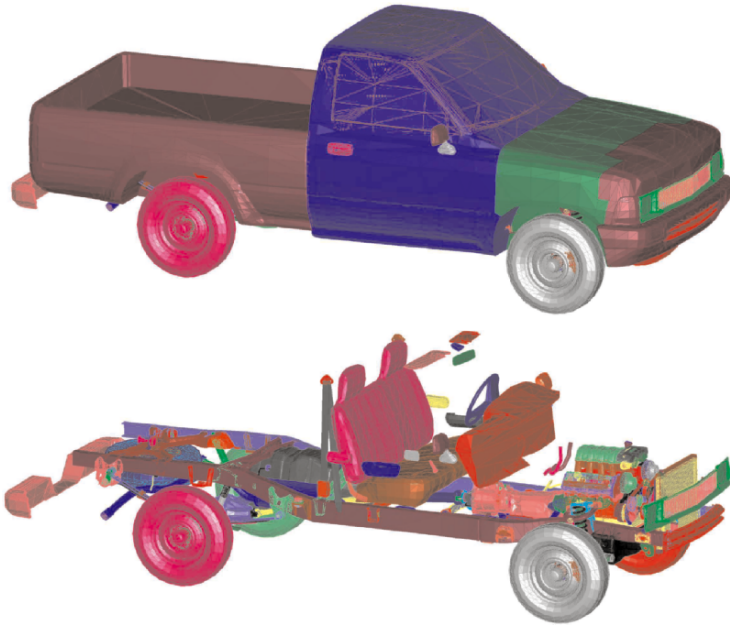


Fig. 15. Truck model for underhood thermal management simulation

took roughly 1 man week to construct the wrapped surface mesh ready for tetrahedral volumetric mesh generation from the initial STL files including hole filling and tailored mesh generation and connection for critical parts such as the heat exchanger.

The volume mesh generation was carried out mainly with tetrahedral elements and wedge prism layers were applied to some parts. Several failures occurred due to poor triangular quality and excessively close proximity between opposing faces. The final volume mesh used for the fluid simulation contains 6.8 million cells with 0.7 million interior nodes with the boundary entities mentioned prior. Figure 17 shows streamlines and pressure distributions obtained by the simulation.

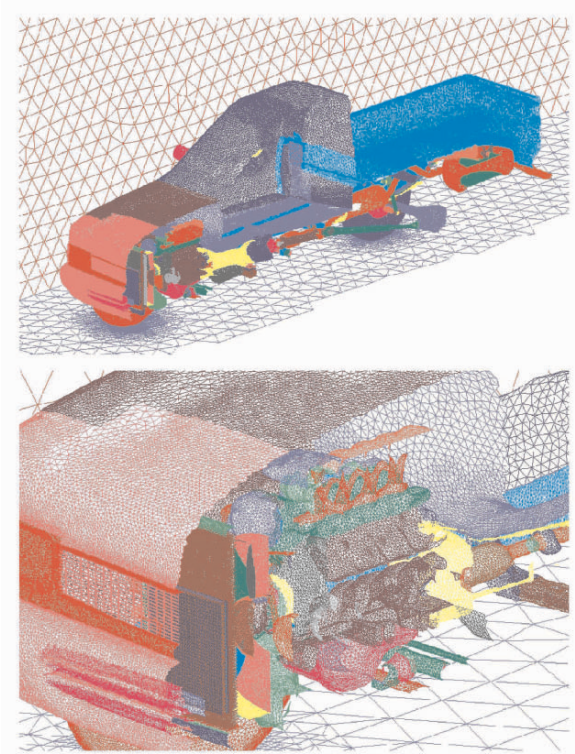


Fig. 16. Configurations of final wrapper surface for 3D volume mesh generation

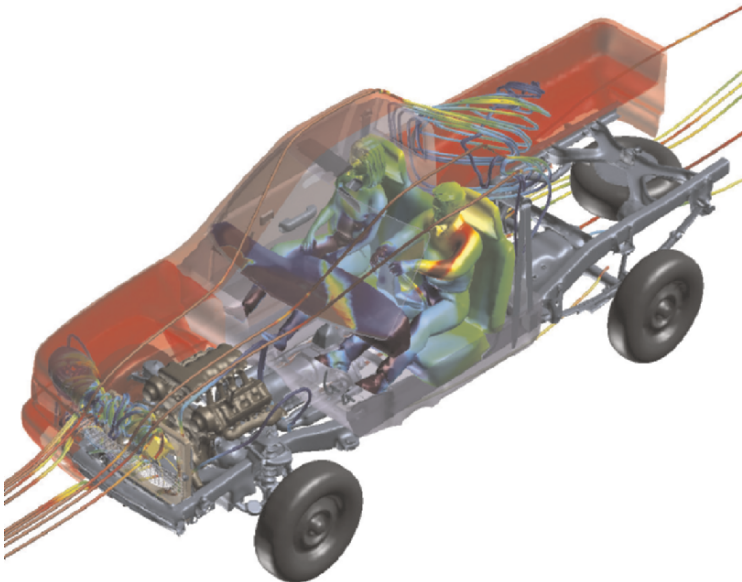


Fig. 17. Streamlines and pressure distributions from CFD simulations

6. Discussions and Future Works

A mesh generation technique has been investigated to construct triangular 3D surface meshes for dirty geometries and applications of the developed mesh generator have been presented to demonstrate effectiveness of the proposed technique. The proposed algorithm constructs initial watertight triangular meshes called the wrapper surfaces from the Cartesian grid overlaid onto the input dirty geometries by analyzing intersections of its cells. The final wrapper surface is obtained by gradually improving until a quality surface mesh is achieved to proceed to the subsequent tetrahedral volumetric mesh generation with. As the intersection check in Cartesian grid generation is tolerant to geometric dirtiness to a certain extent, it is possible to reduce a significant amount of user interactions that were necessary in traditional geometric healing.

Further works are under investigation in some areas. For example, it will be necessary to improve memory efficiency to exploit the presented technique on extreme applications such as combined fluid simulations of external aero, underhood thermal management and cabin HVAC. The cells in the presented study are represented a face-based unstructured data structure. A comparative study is undergoing using the octree data structure. Also, several key improvements should be made to deliver special need to construct meshes for volumes with thin surfaces attached to. Automated hole fixing is another challenging issue. The two major problems are, first, to identify a hole and, second, resolve it. According to our experience, the hole can be an artifact one caused by missing faces as well as structural one, for example, a hole on a sheet metal component. The former is relatively easier to detect due to the existence of free mesh edges each having only one connected face. The later often appears clean as the thin sheet metal is defined very close two surfaces that are fully connected to neighboring surfaces. In the existence of holes, the traditional way is to patch them with extra triangles manually. An interesting alternative is to coarsen the cells around a hole so that the hole does not appear in the wrapper surface. Further study will be carried out for this issue.

References

- [ABM97] M. J. Aftosmis, M. J. Berger and J. E. Melton (1997) Robust and efficient Cartesian mesh generation for component-based geometry, AIAA Paper 97-0196
- [BPK05] S. Bischoff, D. Pavic and L. Kobbelt (2005) Automatic restoration of polygonal models, ACM Trans. Graphics, 24(4), pp. 1332-1352.
- [BWS03] M. W. Beall, J. Walsh and Mark S. Shephard (2003) Accessing CAD geometry for mesh generation. Proc. 12th International Meshing Roundtable, pp. 33-42

- [CDA06] CD-Adapco (2006) PROSTAR Automatic Meshing, <http://www.adapco.com/SoftwareProducts/proam-htm.htm>
- [CEI06] CEI Enight (2006) Harpoon: The Extreme Mesher, <http://www.ensight.com/products/harpoon.html>
- [CFD06] CFDRC, CFD-VisCART, http://www.cfdrc.com/serv_prod/cfd_multiphysics/software/ace/viscart.html
- [GH97] M. Garland and P. S. Heckbert (1997) Surface simplification using quadric error metrics, Proc. 24th Annual Conf. on Computer Graphics and Interactive Techniques, pp. 209-216
- [Her76] L. R. Hermann (1976) Laplacian-Isoparametric Grid Generation Scheme, J. of the Engineering Mechanics Division of the American Society of Civil Engineers, 102, pp. 749-756
- [KVL99] L. Kobbelt, J. Vorsatz, U. Labsik and H.P. Seidel (1999) A shring wrapping approach to remeshing polygonal surfaces, Computer Graphics Forum, 18(3), pp. 119-130
- [PER04] M. Peric (2004) Simulation of flows in complex geometries: New meshing and Solution Methods, Proc. NAFEMS Seminar: "Simulation of Complex Flows (CFD) - Application and Trends", Niedernhausen/Wiesbaden, Germany
- [She85] M. S. Shephard (1985) Automatic and adaptive mesh generation, IEEE Trans. Magnetics, 21, pp. 2482-2489
- [SBC97] A. Sheffer, T. Blacker, J. Clements and M. Bercovier (1997) Virtual Topology Operators for Meshing, Proceedings, 6th International Meshing Roundtable, pp. 49-66
- [Sch95] R. Schneiders (1995) Automatic Generation of Hexahedral Finite Element Meshes, Proceedings, 4th Int. Meshing Roundtable, pp. 103-114
- [SWC00] J. P. Steinbrenner, N. J. Wyman and J. R. Chawner (2000) Fast Surface Meshing on Imperfect CAD Models, Proc. 9th Int. Meshing Roundtable, pp. 33-41
- [Tau95] G. Taubin (1995) Curve and Surface Smoothing without Shrinkage, Proc. 5th Int. Conf. Computer Vision, pp. 852-857
- [VMM99] J. Vollmer, R. Mencl and H. Müller (1999) Improved Laplacian smoothing of noisy surface meshes, Computer Graphics Forum, 18(3), pp. 131-138.
- [WCG05] A. Wissink, K. Chand, B. Gunney, C. Kapfer, M. Berger, B. Kosovic, S. Chan and F. Chow (2005) Adaptive urban dispersion integrated model, 8th American Meteorological Society Annual Meeting, Atlanta, GA.
- [WS02] J. Z. Wang and K. Srinivasan (2002) An adaptive Cartesian grid generation method for 'Dirty' geometry, Int. J. Numer. Meth. Fluids, 39, pp. 703-717
- [ZBS02] J. Zhu, T. Blacker and R. Smith (2002) Background Overlay Grid Size Functions, Proc. 11th Int. Meshing Roundtable, pp. 65-74
- [ZF02] H. Zhang, E. Fiume (2002) Mesh smoothing with shape or feature preservation, Proc. Computer Graphics International 2002, pp. 167-182.

A Hole-Filling Algorithm for Triangular Meshes Using Local Radial Basis Function

John Branch¹, Flavio Prieto² and Pierre Boulanger³

¹Universidad Nacional de Colombia, Sede Medellín
jwbranch@unal.edu.co

²Universidad Nacional de Colombia, Sede Manizales
faprieto@unal.edu.co

³Department of Computing Science
University of Alberta, Canada
pierreb@cs.ualberta.ca

Abstract. Creating models of real objects is a complex task for which the use of traditional modeling techniques has proven to be difficult. To solve some of the problem encountered, laser rangefinders are frequently used to sample an object's surface from several viewpoints resulting in a set of range images that are registered and integrated into a final triangulated model. In practice, due to surface reflectance properties, occlusions and accessibility limitations, certain areas of the object's surface are not sampled leaving holes which create undesirable artifacts in the integrated model. In this paper, we present a novel algorithm for the automatic hole-filling of triangulated models. The algorithm starts by locating hole boundary regions. A hole consists of a closed path of edges of boundary triangles that have at least an edge, which is not shared with any other triangle. The edge of the hole is then fitted with a b-spline where the average variation of the torsion of the b-spline approximation is calculated. Using a simple threshold of the average variation of the torsion along the edge, one can automatically classify real holes from man-made holes. Following this classification process, we then use an automated version of a radial basis function interpolator to fill the inside of the hole using neighboring edges. Excellent experimental results are presented.

1. Introduction

Creating accurate models of real environments is a nontrivial task for which traditional modeling techniques are inappropriate. In these situations, the use of laser rangefinders [5] seems attractive due to their relative independence from the sampled geometry and their short acquisition time. The combined use of range and color images is very promising and it has been demonstrated that together they can produce an unprecedented degree of photorealism [15, 16]. Unfortunately, surface properties (i.e., low or specular reflectance), occlusions and accessibility limitations cause the scanner to miss some surface elements, leading to incomplete reconstruction of the scene and introducing holes in the resulting models. Creating high-quality mesh representations of objects based on such incomplete information remains a challenge [24]. Due to the costs and difficulties involved in scanning real environments, it is desirable to have automatic or semiautomatic tools for helping users to improve the quality of incomplete data sets.

The problem of filling holes in a triangulated mesh can be divided into two sub-problems: hole identification and construction of the missing data using the available data near the holes. Unfortunately, none of these problems are trivial since holes created during the scanning of geometrically rich objects, such as detailed sculptures, can be quite complex [9]. However, in many practical cases, holes occurring in range images can be topologically simpler. This is the case of many holes found when scanning interior environments where most surfaces tend to be smooth and planar areas are abundant (for example: imagine a home or an office environment). For these situations, simpler algorithms for identifying holes and for parameterizing their neighbors can be specified to avoid the problems usually associated with more general cases.

This paper presents a novel algorithm for automatically identifying and filling holes in regions associated with smooth surfaces. Although our algorithm is targeted towards filling holes in smooth surfaces, it does not provide a general solution to the hole-filling problem. It is conceptually very simple and its implementation is straightforward. The algorithm takes a triangulated mesh, which is analyzed for the existence of boundary edges (edges that belong to a single triangle). The occurrence of a

hole implies the existence of a cycle defined by boundary edges. Thus, once a boundary edge is found, the algorithm traces the entire boundary. A ring of points around the boundary is used for an interpolation procedure that will eventually fill in the hole. The points near the hole are then used to fit a surface using a radial basis function (RBF) interpolator. An important feature of our algorithm is that it guarantees that the reconstructed patches blend smoothly into the original surface. Moreover, the reconstructed surface preserves the original sampling rate of the original mesh. As the new primitives are distinguished from the original points, they can be processed further. Since the algorithm works after surface reconstruction, it can be used with any reconstruction technique and its processing is only limited to the size of holes. In this paper, we demonstrate the effectiveness of our approach in real data sets and show how it can significantly improve the overall quality of a triangular mesh model.

The paper is organized as follows: Section 2 discusses previous work. Section 3 presents the details of the hole-filling algorithm. Section 4 discusses results obtained using this algorithm, and Section 5 summarizes the contributions of this paper and proposes future developments.

2. Previous and Related Work

Hole filling is an important problem in object reconstruction and this work benefits from previous efforts in areas such as range image registration [6, 17, 20] and surface reconstruction from point clouds [2, 3, 11, 12].

Curless and Levoy [8] used a hole-filling technique to interpolate non-sampled surfaces in concave regions of objects. In this case, the added surfaces were intended to produce “watertight” models for reproduction using rapid prototyping machines. Their algorithm has little or no impact on the appearance of the objects. In our work, we are concerned about the reconstruction of holes that, if not fixed, would result in major rendering artifacts.

Carr et al. [7] use polyharmonic radial basis functions (RBF) to fit an implicit representation to a set of sampled points. This technique consists of creating a signed distance function, fitting an RBF to the resulting distance function, and extracting iso-surfaces from the

fitted RBF. It is general and produces very impressive results, but the entire set of points is treated as a single implicit function. In order to create the signed distance function, the system needs to know what portion of the space corresponds to the exterior of the surface and what portion corresponds to its interior, which may not always be readily available.

Davis et al. [9] use a volumetric diffusion approach, analogous to in-painting techniques [4, 18], to fill holes in range scans. The process consists of converting a surface into a voxel-based representation with a clamped signed distance function. The diffusion algorithm consists of alternated steps of blurring and compositing, after which the final surface is extracted using Marching Cubes [14, 28].

Alexa et al. [1] used point sets to represent shapes and employed an approach similar to ours as they also locally project points onto planes and fit surfaces through those points. The reconstructed surfaces are used to sub-sample the point set. Their method, however, does not attempt to fill holes on surfaces.

Wang and Oliveira [21] proposed a pipeline to improve the reconstruction of scenes represented as sets of range images. The pipeline consists of a segmentation step followed by the reconstruction of missing geometric and textural information for individual objects. The reconstruction of missing geometric data exploits the fact that real (indoor) scenes usually contain many planar and symmetric surfaces. Thus, a 3D Hough transformation is used to identify large planar regions, whose corresponding samples are replaced by texture-mapped polygons and are also removed from the point cloud [10, 22, 23]. In the remaining dataset, individual objects are segmented as clusters of spatially closed points, using an incremental surface reconstruction algorithm [11]. Inside each cluster, the point cloud is analyzed and searched for approximately symmetric patterns using a variation of the 3D Hough transformation [21]. As these patterns are identified; the algorithm automatically proceeds with reconstruction by mirroring data from one part of the model to another. Figure 1 shows a chair extracted from a real data set. The image on the left corresponds to the original samples with large holes. The image to the right shows the model recovered using the symmetry-based reconstruction algorithm

described in [21]. It provides a significant improvement with respect to the original model, but some holes are still visible. Such holes essentially result from the lack of data on both sides of the model and from limitations of surface reconstruction algorithms [2, 3, 11, 12] that work effectively on areas with variable sampling density. The algorithm presented in this paper intends to fill these remaining holes.



Figure 1. a) Chair from the range image has big holes due to occlusion, b) Symmetry-based reconstruction.

3. Hole-Filling Algorithm

With the objective of correcting the topological anomalies related to the absence of information in mesh representing objects, it is necessary to generate new points in those regions which have not been scanned.

The proposed methodology first identifies and analyzes the holes present in the grid to determine which ones must be filled and which ones belong to the topology of the object. For example, discontinuities are present in the eye area on the surface of the mask as shown in Figure 2. Figure 3 shows the block diagram of the proposed algorithm.

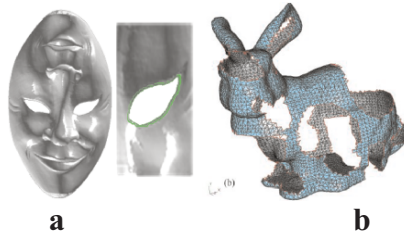


Figure 2. a) Surface Hole, b) Representation Hole.

The analysis of the holes consists of studying the torsion of the contour curve of every hole. This analysis is based on the idea that every hole belonging to the surface is smooth and regular, but the holes generated by occlusion often present irregularities reflected in high torsion of the contour. Some examples are shown in Figure 4. Next, an iterative process is started to fill the holes using neighboring points. These points are generated by means of local interpolators of radial basis functions originating from a neighborhood selected iteratively around a hole until a pre-selected fitting threshold is reached.

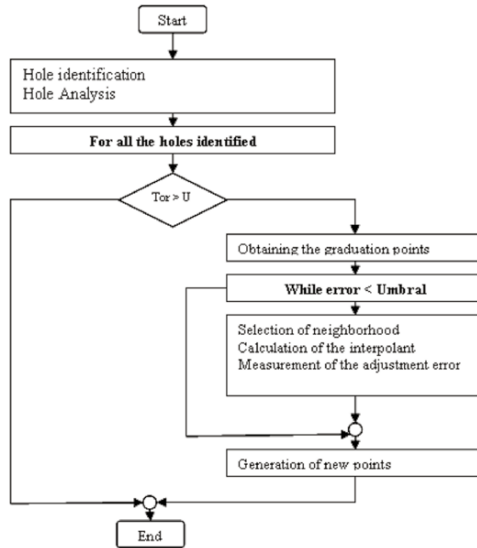


Figure 3. Block diagram of the algorithm.

3.1. Hole Identification

The first step for the process of hole filling is the process of identification, during which it is possible to find the different types of holes present in the topology of an object. There are those that really belong to the surface, and those that are caused by the acquisition process itself, i.e, such as holes due to occlusion or due to insufficient views as illustrated in Figure 4.

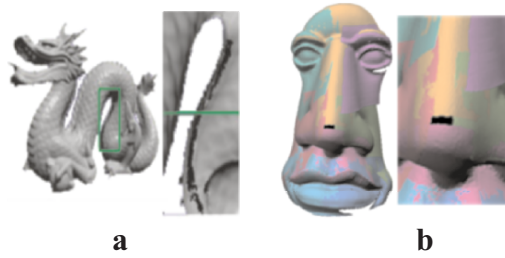


Figure 4. a) Hole generated by occlusion, B) Hole generated by partial scanning.

If the object is a triangular mesh, a hole consists of a closed path of edges of border triangles. A border triangle is the one that has at least an edge which is not shared with any other triangle (that edge is called a limiting edge).

By following these border edges the holes can be detected automatically. However, it is possible to differentiate two kinds of borders: an internal limit which delimits a hole on a surface, and an external limit which delimits a patch or an island inside a hole or the limits of the surface. For the filling process, the path which represents the limits of the surface is eliminated within the set of detected holes. This elimination can be done by verifying that every one of the holes identified does not enclose a surface.

Initially, the algorithm takes a seed triangle located at any part of the grid, and it searches on the whole mesh until it finds a border triangle from which it starts a recursive search to find a closed path. This search is done by determining which one of the three edges is on a border, and then it searches for an adjacent border triangle that shares some of those vertices. The algorithm proceeds until the starting triangle is found again.

Figure 5 shows lateral views of the Stanford bunny data set. It has five holes, two of them are part of the real object. The other holes were generated in early steps of surface reconstruction (acquisition and registration). Figure 6 shows the final result of the hole identification algorithm.

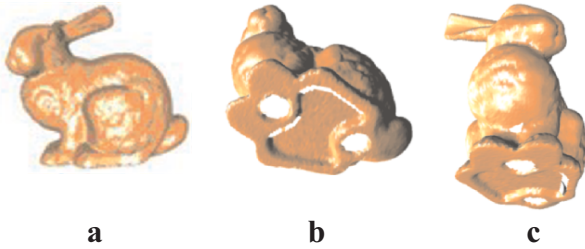


Figure 5. a) Lateral view of Stanford's Bunny, b) and c) View of the five holes that the object owns.

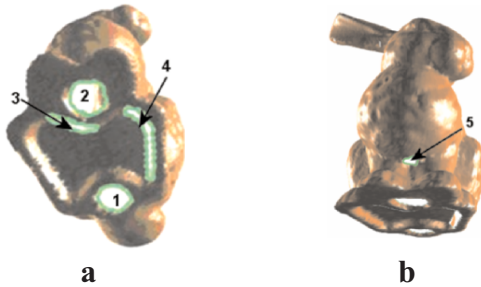


Figure 6. a) and b) Result of the algorithm of holes identification on the mesh, five identified holes.

3.2. Hole Analysis

The step following the detection of holes is analysis. This stage tries to determine whether a hole must be filled or not, whether the hole is present on the surface of the real object, or if the hole was caused during an intermediate stages of reconstruction.

There are an infinite number of configurations of holes on free form objects which makes it very difficult to identify the actual hole belonging to the surface. This is why the hole-filling process requires an interaction with the user. An attempt to automate this procedure consists in analyzing the contour curve of each hole. Keeping in mind that the smoothness of the contour is relative to the density of the sampling points, the holes present in man-made objects usually have smooth contours.

On the other hand, the holes caused by occlusion, present great variability of the contour curve as shown in Figure 7.



Figure 7. a) Internal contour, b) External contour.

Since the contour line is a curve, it can be studied and classified according to its geometric properties as curvature and torsion. To classify the contour curves we use the torsion and not the curvature because a hole on a surface can have a wide range of curvature variations. It is important to establish the behaviour of every curve in the space, such as its smoothness or high variability.

The study of the torsion of a curve depends on the behaviour of the osculating plane. The osculating plane is the plane nearest to the curve at an arbitrary point A. This plane crosses the A point and it contains the tangent \vec{T} and the normal \vec{N} to the curve at A, as illustrated in Figure 8.

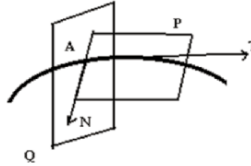


Figure 8. Osculating plane (Vector for N and T necessary).

From point-to-point through the curve, the position of the osculating plane varies in a similar way to the tangent vector. This simple characteristic allows characterization of the curvature. The variation of the osculating plane allows calculation of the torsion of a curve. Similar to the curvature, the variation of the osculating plane is measured according to the arch longitude. That is, if ψ is the angle between the osculating planes at a fixed point A and a neighbouring point X, and if Δs is the arch longitude AX, then the torsion τ at the point A is defined as the limit

$$\tau = \lim_{\Delta s \rightarrow 0} \frac{\psi}{\Delta s} .$$

The sign of the torsion depends on the side of the curve towards which the osculating plane turns when it is moving across the curve. However, according to differential geometry, the properties of the curve at a point are those properties which depend on an arbitrarily small environment. The properties of this type of curve are defined in terms of derivative on the curve at the point. The estimation of the torsion is defined as follows:

$$\tau = \frac{|(\vec{f}'(t) \times \vec{f}''(t)) \bullet \vec{f}'''(t)|}{|\vec{f}'(t) \times \vec{f}''(t)|^2}.$$

To estimate the torsion of the curve of at least three degrees are necessary. In order to compute these derivatives, the boundary points are fitted with the Bézier's parametric curve $\vec{f}(t)$ defined as follows:

$$\vec{f}(t) = \sum_{i=0}^n \vec{p}_i B_{i,3}(t),$$

where:

$$x(t) = \sum_{i=0}^n x_i B_{i,3}(t),$$

$$y(t) = \sum_{i=0}^n y_i B_{i,3}(t),$$

$$z(t) = \sum_{i=0}^n z_i B_{i,3}(t),$$

and where \vec{p}_i are the control points of the Bézier curve and $B_{i,3}(t)$ are the Bernstein polynomials of the third degree. These are defined by:

$$B_{0,3} = (1-t)^3,$$

$$B_{1,3} = 3t(1-t)^2,$$

$$B_{2,3} = 3t^2(1-t),$$

$$B_{3,3} = t^3.$$

The contour is partially approximated by means of Bézier curves of the third degree with sets of four continuous points until the estimation of the torsion is computed at every point along the contour.

Once they are obtained, the equations of the torsion for the segments along the curve are evaluated at the last point. Since Bézier curves guarantee that the obtained curve has the extreme points of the set from which it is calculated, the error of approximation that is present at the intermediate points does not affect the torsion’s estimation (see Figure 9).

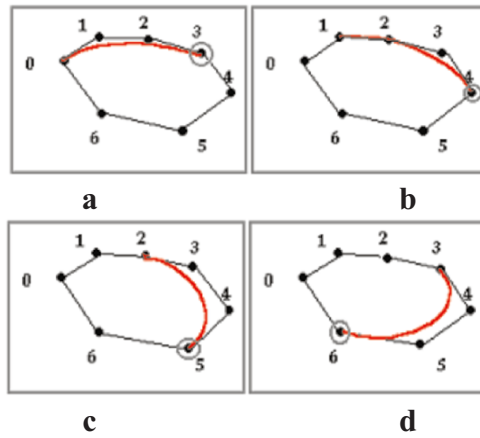


Figure 9. Approximation to the contour curve by means of Bézier’s curves and points on which the torsion is estimated.

Finally, the variance of the torsions is calculated to measure the level of dispersion of the measured values at each point. The holes whose contours have a variance of torsion higher than a pre-established threshold, will be filled. The value of this measure is obtained by:

$$S_{torsion} = \frac{\sum_{i=0}^n (\tau_i - \bar{\tau})^2}{n}.$$

In general, low values of dispersion suggest a surface hole. Figure 10 shows different cases of the hole and its values of dispersion. In these cases, low dispersion is considered as values less than 0.1. Although the smoothing of long contour curves depend on sample

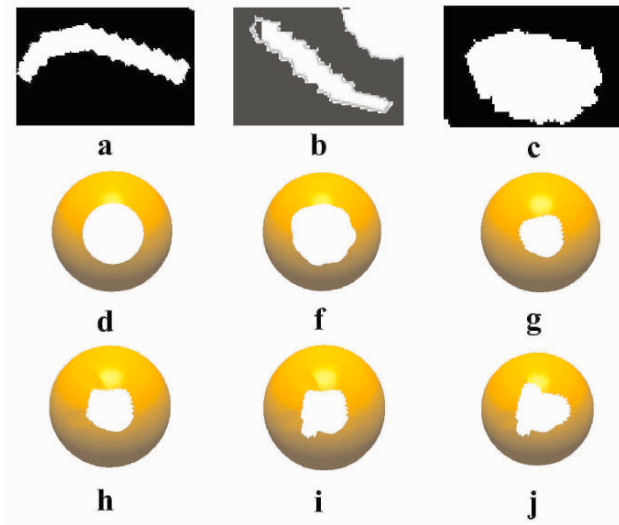


Figure 10. Measurement of the torsion of contour curve in six different cases a) 0.31, b) 0.42, c) 0.245, d) 1.1E-4, e) 3.2E-3, f) 0.04, g) 0.09, h) 0.38, i) 0.51, j) 0.63.

density. We assumed that in a real data range the density is adequate and that the mesh does not yet have any reduction procedure.

3.3. Calculation of the Interpolant

Once the holes to be filled are classified, the missing points are computed by an interpolation function using a continuous interpolation scheme. In order to do this, a function $h(x)$ is calculated from a set of points distributed in a homogenous way around the contour of the hole. This interpolating function is constructed using a radial basis function.

The procedures based on radial basis functions have proven to be very useful in the reconstruction of shapes from noisy, disperse, and incomplete data [25, 26, 27]. Recent studies about RBF are centred on the reconstruction of a big set of points produced by modern acquisition devices [11, 17, 7, 26].

The radial basis functions are circularly symmetric functions centred on a point called *centre*. To calculate an interpolant of RBF, let us consider a set of points $P = \{\vec{p}_1, \dots, \vec{p}_N\}$ sampled from a surface S and with a set of normals, $N = \{\vec{n}_1, \dots, \vec{n}_n\}$, which indicate

the orientation of the surface. The main goal is to build a function $h(\vec{r})$ in such a way that the set of zeroes satisfies the equation $\vec{r} \in \mathfrak{R}^3 : h(\vec{r}) = 0$, approximating the set of points P.

The typical interpolation function $h(\vec{r})$ interpolating P is defined as:

$$h(\vec{r}) = \sum_{p_i \in P} [g_i(\vec{r}) + \lambda_i] \phi_\sigma \left(\left\| \vec{r} - \vec{C}_i \right\| \right),$$

where $\phi_\sigma(s) = \phi(s/\sigma)$, $\phi(s) = |s|^3$ is a tri-harmonic radial basis function used for the approximation of the absent surface. The λ_i are the set of weights associated to each centre, g_i are typically a polynomials of second or third degree, and the \vec{C}_i are the set of centres.

For hole filling, it is necessary to compute independent interpolators local to every hole. Therefore, for every hole, a different interpolant function is estimated with a reduced set-of-points. This set-of-points should be as big as possible and homogeneously distributed so that the obtained function can estimate the topology of the missing points.

3.4. Centres Selection

The computation of the interpolating radial basis functions, which is not a compact support, is expensive. Therefore, the selection of the centres or set-of-points on which the interpolant will be calculated carefully. The estimation of the adequate neighbourhood is done by means of an iterative procedure (see Figure 11).

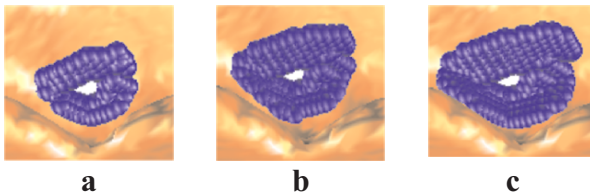


Figure 11. a) Iteration 1, Neighbourhood size: 38. b) Iteration 2, Neighbourhood size: 106. c) Iteration 3. Neighbourhood size: 172.

This process starts with a small number of selected centres, as a set of points radially near to each one of the vertices on the contour curve, the size of radius is initially established by the user, this selection is shown in Figure 12. An interpolant is calculated for these points and the precision is measured on a set of control points. If the precision measured is not equal to the threshold or lower than a pre-established threshold, the size of radius is duplicated, and a new selection process is done until a threshold is reached. In every one of these iterations an interpolant is calculated. The evaluation of the quality of the interpolation is done over a set of reference points, which initially belong to the neighbourhood of the hole but are not used to calculate the interpolant.

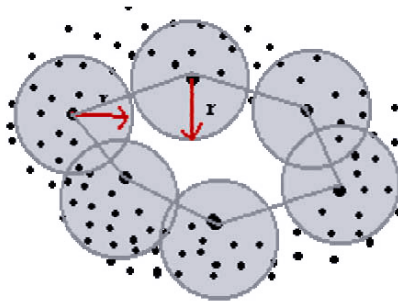


Figure 12. Estimation of the set of centres.

Figure 13 shows the result of the selection of the adequate neighbourhood for the calculation of the interpolating function on a real object. Once the initial neighbourhood is obtained, the reference points set must be determined to measure the quality of the interpolating function. This set-of-points should be kept constant. In the proposed algorithm, the set-of-points of the initial neighbourhood is clustered to obtain homogeneous regions which describe different variations of the topology in the regions around the hole. A cluster type K-mean [19] is used, where the parameter K will be equal to the number of vertices that form the hole's contour.

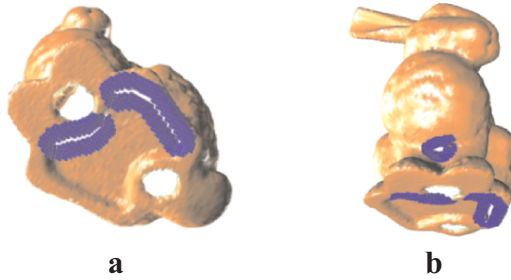


Figure 13. a) and b) final neighborhood for each hole in the bunny.

Once the set of regions defined for every cluster is obtained, a point is randomly selected from each of them (see Figure 14), which will represent every region. In this way, it is assured that the evaluation is done homogeneously around the hole. If an interpolating function reaches the threshold with high precision, it means that it represents the topology of the hole’s neighbourhood.

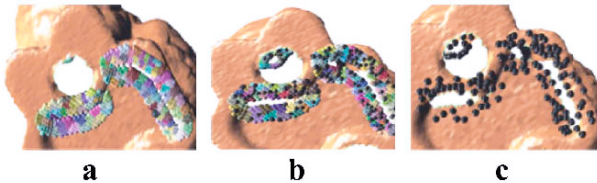


Figure 14. a) Neighbourhood clusterization. b) and c) Selection of the reference points set.

3.5. Filling the Hole

In order to fill each hole, it is important to remember that the reconstructed segment preserves the sampling density of the original mesh; that is, the sampling density that is measured for each one of the holes.

In general, two important criteria are used for determining the new points that fill each hole. First, the position of new points should be inside the hole and the new triangles added to the hole must be easy to merge with the original mesh. The local triangulation is an efficient procedure for hole-filling because it avoids the remeshing of the cloud-of-points. Additional procedures such as the normal estimation over the new points and the new normal of the contour points that will be different due the new

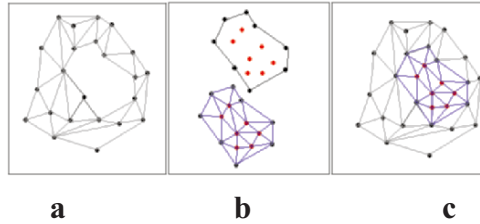


Figure 15. Hole triangulation a) Initial mesh, b) Contour extraction, new points generation and local triangulation, c) Filling hole.

segment of the surface, can also be made locally (see Figure 15). Second, the density of the new set-of-points must be of that vicinity.

In order to generate the new segment of the mesh, we use an iso-surface algorithm with a RBF interpolator [7]. Given a density function S , an iso-surface at value a is defined as the set of all points \vec{p} where $f(\vec{p}) = a$. In the context of surface reconstruction, where S represents a signed-distance function. The reconstructed surface corresponds to the iso-surface where $f = 0$.

The function S is sampled at regular intervals to construct a manifold mesh of polygons representing the desired iso-surface at a specified resolution. The density of the new segment of the mesh is equal to the mean value of the original mesh. Facet vertices are ordered such that the cross-product of adjacent edges (the facet normal) is consistent with the gradient of the density function S . The marching cubes algorithm is a well-known general purpose algorithm and we use it to fill the holes as illustrated in Figure 16.

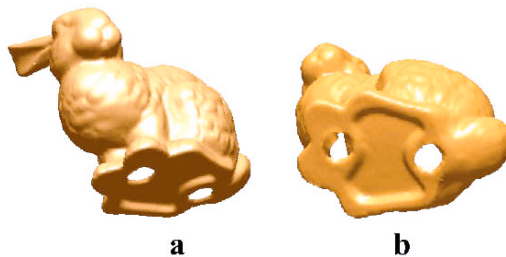


Figure 16. a) and b) show the filled holes identified in Section 3.1.

4. Experiments and Results

All tests were made with a computer with a 3.0GHz processor, 1.0Gb of ram memory running under the Microsoft XP operating system. The implementations of the models were made in C++ and MATLAB. In addition, a graphical motor in OpenGL was programmed to obtain the graphical representation of the images. The data were acquired with the Kreon sensor available in the Advanced Man-Machine Interface Laboratory–Department of Computing Science, University of Alberta, Canada.

In order to calibrate the model and to validate the correct behavior of the interpolator over 3D points, several tests were made on synthetically generated holes. The generation of synthetic holes is necessary to evaluate the quality of the points obtained with the interpolator since in real cases, the degree of precision of the points generated with respect to the real section of the surface is impossible to measure. The test consisted of generating synthetic holes on real range image, extracting a near points neighborhood by means of a kd-tree structure. Next, the hole is filled with the proposed strategy and the error of fit between the extracted data of the real surface and the new points is measured. The error reported in Table 1 corresponds to the error of the distance between both sets of points. Figure 17, shows the variation of neighborhood size to fill the hole.

Table 1. Variation of neighborhood size to fill the hole.

Number of Points	Size of Hole=50	Surface size = 6051
Neighborhood Size	error	% of Surface
100	1.79E-2	1.65%
200	1.60E-2	3.31%
500	1.03E-3	8.26%
1000	0.83	16.53%

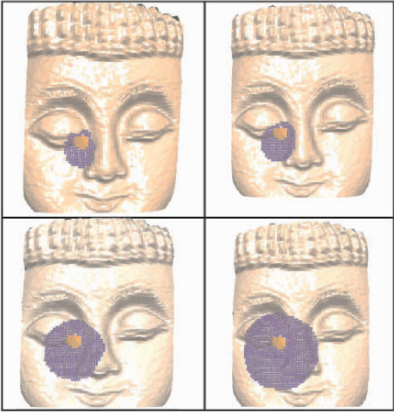


Figure 17. Different neighborhood sizes.

To show the behavior of our algorithm one can see in Figures 18 and 19 results for several real 3D cases. These images show that the algorithm generates smooth segments to fill the surface holes in different configurations and correctly identifies each one of the holes.

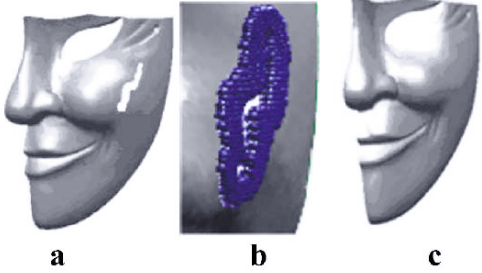


Figure 18. a) Original mesh, b) set of center and c) hole filled with RBF interpolant.

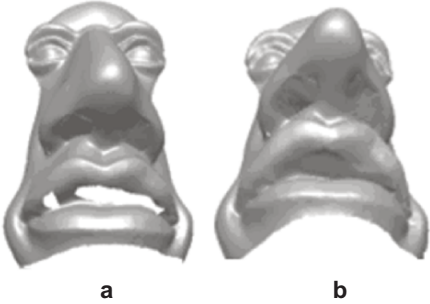


Figure 19. a) Original mesh, b) hole filled with RBF interpolating function.

5. Conclusion

We have presented a new technique for filling holes in a triangulated model using a local radial basis function interpolant defined for each one of the holes detected using the torsion fluctuation around the contour curve. In the algorithm, each contour curve is approximate with a Bézier curve segment from which the torsion is calculated analytically. The method is simple and effective, since the radial basis function fits the surface smoothly and always generates a closed manifold triangular mesh.

When big holes are present in a mesh, the interpolating function cannot adequately fit the surface. No big holes can exist if a good scanning process is done, that is, holes whose sizes do not exceed 3% of the total size of the mesh. Our algorithm has only one parameter: the predefined threshold for the variation of the torsion to determine if a hole must be filled. The other values like the size of vicinity are automatically calculated.

The threshold for the variation of the torsion to identify a hole in the mesh will be affected by the sample density and the method to make the mesh. So an approach to automatically define this value from a given mesh would be desirable. In some cases the holes would be on a plane. For these cases our algorithms must complete with a normal variation analysis of limit edges; but this is not a general case because the holes too often are caused by occlusion and are not trivial surfaces.

References

1. Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C., "Point Set Surfaces", *IEEE Visualization*, pp. 21–28, 2001.
2. Amenta, N., Bern, M., and Kamvysselis, M., "A new Voronoi-based surface reconstruction algorithm", *SIGGRAPH'98*, pp.415–421, 1998.
3. Bajaj, C., Bernardini, F., and Xu, G., "Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans", *SIGGRAPH'95*, pp.109–118, 1995.
4. Bertalmio, M., Shapiro, G., Caselles, V., and Ballester, C. "Image Inpainting", *SIGGRAPH'00*, pp.417–424, 2000.
5. Besl, P., "Advances in Machine Vision", *Advances in Machine Vision*, Springer Verlag, Chapter 1 – Active Optical Range Sensors, pp. 1–63, 1989.
6. Besl, P. and McKay, N., "A method for registration of 3D shapes", *IEEE Trans. on PAMI*, 14 (2). pp. 239–256, 1992.

7. Carr, J., Beatson, R., Cherrie, J., Mitchell, T. Fright, W., and McCallum, B., "Reconstruction and Representation of 3D Objects with Radial Basis Functions", *SIGGRAPH'01*, pp. 67–76, 2001.
8. Curless, B. and Levoy, M., "A Volumetric Method for Building Complex Models from Range Images", *SIGGRAPH'96*, pp. 303–312, 1996.
9. Davis, J., Marschner, S., Garr, M., and Levoy, M., "Filling Holes in Complex Surfaces Using Volumetric Diffusion" *Proc. First International Symposium on 3D Data Processing, Visualization, Transmission*, pp. 428–861, 2002.
10. Efron, A. and Freeman, W., "Image Quilting for Texture Synthesis and Transfer", *SIGGRAPH'01*, pp. 341–348, 2001.
11. Gopi, M. and Krishnan, S., "A Fast and Efficient Projection Based Approach for Surface Reconstruction", *Intern. Journal of High Performance Computer Graphics, Multimedia and Visualization*, 1 (1), pp. 1–12, 2000.
12. Hoppe, H., DeRose, T., Duchamp, McDonald, T. J.A., and Stuetzle, W., "Surface reconstruction from unorganized points", *SIGGRAPH'92*, pp. 71–78, 1992.
13. Lancaster, P. and Salkauskas, K., *Curve and Surface Fitting: an Introduction*. Academic Press. 1986.
14. Lorensen, W. and Cline, H., "Marching cubes: A high resolution 3D surface construction algorithm", *Proc. SIGGRAPH'87*, pp. 163–169, 1987.
15. McAllister, D., Nyland, L., Popescu, V., Lastra A. and McCue, C., "Real Time Rendering of Real World Environments", *Proc. Rendering Techniques'99*, pp. 145–60, 1999.
16. Nyland, L., et al., "The Impact of Dense Range Data on Computer Graphics", *Proceedings of Multi-View Modeling and Analysis Workshop*, pp. 3–10, 1999.
17. Nyland, L., Lastra, A., Mc Allister, D., Popescuand, V., McCue, C., and Fuchs, H., "Capturing, Processing and Rendering Real-World Scenes", In *Videometrics and Optical Methods for 3D Shape Measurement, Electronic Imaging., Photonics West*, SPIE Vol. 4309, pp. 107–116, 2001.
18. Oliveira, M., Bowen, B., McKenna, R., and Chang, Y., "Fast Digital Image Inpainting", *International Conference on Visualization, Imaging and Image Processing (VIIP 2001)*, Marbella, Spain, pp. 261–2665, 2001.
19. Hartigan, J. and Wong, M. A., "A K–Means Clustering Algorithm", *Journal of Applied Statistics*, 28 (1), pp. 100–108, 1979.
20. Pulli, K., "Multiview Registration for Large Data Sets", *3DIM'99*, pp. 160–168, 1999.
21. Wang, J. and Oliveira, M., "Improved Scene Reconstruction from Range Images", *Proc. EUROGRAPHICS'2002*, pp. 521–530, 2002.
22. Wei, L.Y. and Levoy, M., "Texture Synthesis over Arbitrary Manifold Surfaces", *SIGGRAPH'01*, pp. 355–360, 2001.
23. Ying, L., Hertzmann, A., Biermann, H., and Zorin, D., "Texture and Shape Synthesis on Surfaces", *Eurographics'2001, Rendering Workshop*, pp. 301–312, 2001.
24. Yu, Y., Ferencz, A., and Malik, J., "Extracting Objects from Range and Radiance Images", *IEEE Transactions on Visualization and Computer Graphics*, 7 (4), pp. 351–364, 2001.
25. Carr, J. C., Beatson, R. K., McCallum, B. C., Fright, W. R., McLennan, T. J., and Mitchell, T. J., "Smooth surface reconstruction from noisy range data", *Proceedings of the 1st international conference on computer graphics and interactive techniques in Australasia and South East Asia*, Melbourne, Australia, February 11 –14, ACM Press, pp. 119–ff, 2003.
26. Buhmann, M., "Radial Basis Fuction: Theory and Implementations", *Cambridge Monographs on Applied and Computational Mathematics*, 2003.

27. Carr, J.C., Beatson, R.K., Cherie, J. B., Mitchell, T.J., Fright, W. R., McLennan, T.J., and Evans, T.R., "Reconstruction and representation of objects with radial basis function", Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 67–76, 2001.
28. C. Montani, R. Scateni, R. Scopigno, A modified look-up table for implicit disambiguation of marching cubes, Visual Comput. 10 (1994) 353–355.

Hexahedral Meshing

A Constructive Approach to Constrained Hexahedral Mesh Generation

Carlos D. Carbonera¹ and Jason F. Shepherd^{2,3}

¹ Gauss Research Laboratory, University of Puerto Rico, Rio Piedras, PR
`carbonera@uprr.pr`

² Scientific Computing and Imaging Institute, University of Utah, Salt Lake City,
UT `jfsheph@sci.utah.edu`

³ Computational Modeling Sciences Dept., Sandia National Laboratories,
Albuquerque, NM
`jfsheph@sandia.gov`

S. Mitchell proved that a necessary and sufficient condition for the existence of a topological hexahedral mesh constrained to a quadrilateral mesh on the sphere is that the constraining quadrilateral mesh contains an even number of elements. S. Mitchell's proof depends on S. Smale's theorem on the regularity of curves on compact manifolds.

Although the question of the existence of constrained hexahedral meshes has been solved, the known solution is not easily programmable; indeed, there are cases, such as Schneider's pyramid, that are not easily solved.

D. Eppstein later utilized portions of S. Mitchell's existence proof to demonstrate that hexahedral mesh generation has linear complexity. In this paper, we demonstrate a constructive proof to the existence theorem for the sphere, as well as assign an upper-bound to the constant of the linear term in the asymptotic complexity measure provided by D. Eppstein.

Our construction generates $76*n$ hexahedra elements within the solid where n is the number of quadrilaterals on the boundary. The construction presented is used to solve some open problems posed by R. Schneiders and D. Eppstein. We will also use the results provided in this paper, in conjunction with S. Mitchell's Geode-Template, to create an alternative way of creating a constrained hexahedral mesh. The construction utilizing the Geode-Template requires $130*n$ hexahedra, but will have fewer topological irregularities in the final mesh.

1 Introduction

Hexahedral mesh generation has been subject to active research during the past twenty years. And, while some progress has been made in the area of

push-button hexahedral mesh generation for fairly specialized classes of complex domains, a generalized hexahedral mesh generation algorithm does not exist which will support all types of domains and applications.

The existence theorem for hexahedral meshes provided by S. Mitchell [6] states that a solid homeomorphic to the sphere, whose boundary is tessellated by an even number of quadrilaterals, can be partitioned into a hexahedral mesh using interior surfaces whose boundaries are the dual cycles of the quadrilateral mesh. The solid partition is referred to as a constrained hexahedral mesh, and the partition of the boundary is known as the constraining quadrilateral mesh.

The problem of constructing constrained hexahedral meshes has proven very difficult to address. The techniques based on S. Mitchell's proof to the existence theorem are difficult to implement; in a few cases, seemingly simple problems are difficult to solve.

D. Eppstein [5] presented a complexity analysis on the generation of hexahedral meshes constrained to a bipartite quadrilateral mesh. Part of his construction depends on adding a layer of cells that have sixteen and eighteen faces; the problem of constructing the hexahedral solution to these cells of quadrilaterals is left open to the reader, and, instead, S. Mitchell's proof is invoked to prove existence of a solution to those cells. In his paper, D. Eppstein focuses on the analysis of the complexity of the generation of constrained hexahedral meshes.

In this paper, a constructive proof is given based on adding four basic transitional cells of hexahedral elements to a quadrilateral mesh: 1) a transition of paired hexahedra, 2) a transition to four-split hexahedra, and 3) a transition from four-split hexahedra to a closed mesh. The rules of how to build the transitional layers of hexahedra using these basic cells will be given. The result presented in this paper is a constructive, easily-programmable, solution that provides a precise, *a priori*, count on the number of hexahedral elements that will be generated.

Additionally, S. Mitchell [7] introduced the Geode-Template to interface a four-split quadrilateral mesh to a diced tetrahedral mesh. In his paper, Mitchell relies on splitting a hexahedral mesh to create a four-split, or diced, quadrilateral boundary. In this paper, we will show how to transition to a four-split mesh without modifying the original boundary.

The remainder of this paper will outline the concepts, definitions, and proofs which ultimately result in a constructive proof of S. Mitchell's existence theorem. The proof of the theorem presented in this paper can be summarized as follows:

1. We introduce the notions of a **Paired Partition** and **Transitions** between quadrilateral meshes. It is shown that every quadrilateral mesh that admits a Paired Partition has a transition to a quadrilateral mesh whose dual has no self-intersecting loops.

2. Given a quadrilateral mesh whose dual has no self-intersecting loops, we introduce a method for transitioning the quadrilateral mesh to a **Four-Split Quadrilateral Mesh**. The transition is created by inserting layers of elements that divide a quadrilateral in two along the each of the two dual cycles that compose the quadrilaterals.
3. It is shown that a **Four-Split Quadrilateral Mesh** on the sphere⁴ is the boundary of a hexahedral mesh.
4. We demonstrate a **Four-Split Pyramid** cell to close the hexahedral mesh.
5. Finally, we show that any quadrilateral mesh on a sphere with an even number of quadrilaterals is the constraining boundary of a hexahedral mesh.

While topologically valid, the resulting quality of the hexahedral mesh created by this construction will not provide solutions for practical applications and is presented merely to provide a concrete measurable construction of a solution to the problem of constrained mesh generation.

The solution presented for the sphere can be extended to the case of the torus and compact 2-dimensional manifolds in general by using the Geode-template coupled with a constrained tetrahedral mesh. (If every loop in a quadrilateral mesh on a 2-dimensional compact manifold has an even number of quadrilaterals, it is possible to apply all the results of this paper to transition to a **Four-Split Quadrilateral Mesh**. This, then, will permit the use of the Geode-template and reduce the problem to the existence of a constrained tetrahedral mesh.)

Finally, a few solutions to open problems in mesh generation are presented including: a new solution to Schneider's open problem [11], the eight-sided quadrilateral octahedron [5], and Eppstein's cube [5]. Additionally, a question by M. Bern, et al. [2] on the existence of a hexahedral decomposition with linear edges for a convex polyhedron is solved by the construction provided in this paper.

2 Basic Terminology

The terms quadrilateral and hexahedral mesh follow the definition given by S. Mitchell in [6]. The dual of a quadrilateral mesh on a compact manifold is a graph where every vertex is connected to four other vertices (i.e. a 4-regular graph). A structure referred to as the Spatial Twist Continuum or STC for short is associated with this graph [9]. In this definition, the notion of chord is introduced. A chord is a chain of quadrilaterals that is constructed by traversing the adjacent quadrilaterals through opposite edges. A loop is

⁴Technically, the construction requires a four-split quadrilateral mesh with a 'star-shaped' boundary (i.e. there must be a point which can be seen by all nodes on the four-split boundary simultaneously.)

a closed chord. In particular, for a quadrilateral mesh on a closed compact manifold, every chord belongs to a loop.

Loops may self-intersect. T. Suzuki, et al. [12] gave a detailed description of how to untangle self-intersecting loops to create the interior surfaces necessary to generate the hexahedral mesh. Their results are used to resolve R. Schneider's pyramid [11].

3 Element Representation

The quadrilateral and hexahedral elements to be referenced throughout the paper will follow the conventions used in finite-element analysis. A quadrilateral is represented by an ordered set of vertices $\{v_1, v_2, v_3, v_4\}$, and bounded by the four edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$, and $\{v_4, v_1\}$. The edges in the quadrilateral that do not share vertices are called opposite edges of the quadrilateral. A hexahedron is represented by an ordered set of vertices $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, and bounded by the six faces $\{v_1, v_4, v_3, v_2\}$, $\{v_5, v_6, v_7, v_8\}$, $\{v_5, v_6, v_2, v_1\}$, $\{v_8, v_7, v_3, v_4\}$, $\{v_6, v_2, v_3, v_7\}$, and $\{v_1, v_5, v_8, v_4\}$.

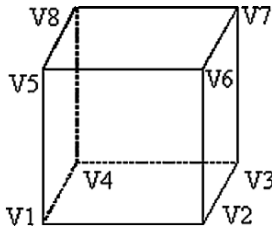


Fig. 1. Element configuration

Additional requirements of a valid quadrilateral mesh are that each edge in the mesh must contain exactly two distinct vertices, and each interior edge must be shared by exactly two quadrilaterals. Similarly, for a valid hexahedral mesh the faces of a hexahedron must contain exactly four distinct vertices, and each interior face of the hexahedral mesh must be shared by exactly two hexahedra.

4 Hexahedral Transitions of Quadrilateral Meshes

Definition 1 *Two distinct quadrilateral meshes are **transitions** of each other if there is a hexahedral mesh whose boundary contains the union of both meshes.*

By solving the hexahedral mesh of the transition of a given quadrilateral mesh, the original hexahedral problem is resolved, because the union of the

hexahedral mesh with the layer of transition elements gives the solution to the original quadrilateral mesh.

4.1 Paired Partition Transition

Definition 2 A **Paired Partition** of a quadrilateral mesh, Q , is a partition PQ of Q such that each element in the partition is a pair of quadrilaterals that share at least an edge.

In other words, a quadrilateral mesh Q admits a paired partition if there exist a set

1. $PQ = \{ \{p, q\}, \text{ such that } p \text{ and } q \text{ are quadrilaterals in } Q \}$,
2. Any two distinct elements in PQ $\{p, q\}$ and $\{p', q'\}$ are disjoint,
3. Q is the union of PQ , and,
4. For each element $\{p, q\}$ in PQ , p and q share an edge or, equivalently, p and q are neighbors.

Since the dual of a quadrilateral mesh on a closed manifold is a 4-regular graph, a **Paired Partition** also corresponds to the graph-theoretic problem known as a perfect matching, or a 1-factor, of a 4-regular graph. We utilize the following theorem (a proof is given in [3]):

Theorem 1. *Every quadrilateral mesh on a 2-Dimensional manifold in \mathbb{R}^3 with an even number of quadrilaterals admits a **Paired Partition**.*

Removal of Self-Intersecting Loops

Theorem 2. *Every quadrilateral mesh on the sphere with n elements that admits a **Paired Partition** transitions to quadrilateral mesh with no self-intersecting loops. The total number of hexahedral elements within the transition between the original quadrilateral mesh and the new quadrilateral mesh with no self-intersecting loops is n .*

Proof: Construct the **Paired Partition** of the quadrilateral mesh. Let $\{p, q\}$ be in PQ .

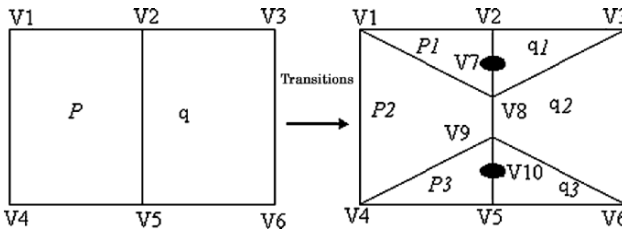


Fig. 2. The Paired Partition Transition. The image on the right is the ‘Cell of Six Quadrilaterals’.

The two quadrilaterals with vertices $\{\{v1, v2, v5, v4\}, \{v2, v3, v6, v5\}\}$ (see Figure 2) transition to the quadrilateral mesh with six quadrilaterals $\{\{v1, v2, v7, v8\}, \{v7, v2, v3, v8\}, \{v1, v8, v9, v4\}, \{v8, v3, v6, v9\}, \{v4, v9, v10, v5\}, \{v9, v6, v5, v10\}\}$. The boundary of the hexahedral mesh comprised of the two hexahedra with vertices $\{v1, v2, v5, v4, v8, v7, v10, v9\}$, and $\{v2, v3, v6, v5, v7, v8, v9, v10\}$ mesh below is the exclusive union of the two sets of quadrilaterals. For any two paired quadrilaterals, p and q , with vertices $\{v1, v2, v3, v4\}$, and $\{v2, v5, v6, v3\}$, construct the two hexahedral elements with vertices $\{v1, v2, v3, v4, v7, v8, v9, v10\}$, and $\{v2, v5, v6, v3, v8, v7, v10, v9\}$.

This transition is applied to each set of paired quadrilaterals in the **Paired Partition**. The boundary of the transition mesh minus the original quadrilateral mesh is composed of cells of six quadrilateral elements (see Figure 2). Thus each paired element in the **Paired Partition** is mapped to a unique set of quadrilaterals $\{p1, p2, p3, q1, q2, q3\}$ in the transitioned mesh.

There is a natural partition induced by mapping each element $\{p, q\}$ in the **Paired Partition** to a unique subset of quadrilaterals $\{p1, p2, p3, q1, q2, q3\}$ of the transitioned mesh. We will call the newly introduced partition of the mesh ‘**Cell of six quadrilaterals**’. As a consequence of a peculiar property of these new cells, it will be shown that all the loops in the new quadrilateral mesh that result from the transition are non-self-intersecting.

The fact that the new mesh will not contain any self-intersections can be seen by taking any quadrilateral in the transitioned quadrilateral mesh, and finding the Cell of six quadrilaterals that contains the quadrilateral. Notice that there are two types of loops that go through a **Cell of six quadrilaterals**: there is one loop that is fully contained inside a **Cell of six quadrilaterals**, and three others that are not (see Figure 3). Notice, also, that the only intersections in the cell take place between the fully contained loop in the cell and one of the loops that is not fully contained in the cell (see also Figure 4). Therefore, for any given quadrilateral, the intersection must take place between two distinct loops. Hence, there cannot be any self-intersections.

A total of n hexahedral elements were used to transition to a quadrilateral mesh with no self-intersecting loops. We should also note here, that it may be possible that changing only a subset of the pairs may be required to remove

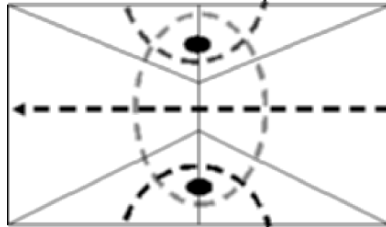


Fig. 3. Dual chords on the Paired Partition Transition boundary

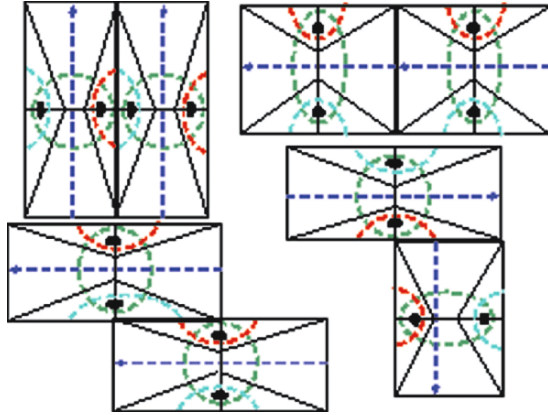


Fig. 4. Various arrangements of dual chords with adjacent **Paired Partition** transitions.

all self-intersections in the mesh. That is to say that n is an upper bound on the minimum transition layer to obtain no self-intersections.

4.2 The Four-Split Transitions

The Four-Split Transition

In this section, we will describe a transition from a non-self-intersecting quadrilateral mesh to a **Four-Split Quadrilateral mesh**. A **Four-Split Quadrilateral mesh** is the collection of four cells that result from splitting a single quadrilateral into four quadrilaterals; five points are added: four at the mid-edges, and one at the center as shown in Figure 5. Any quadrilateral mesh that results from a transition which splits one quadrilateral into four is called a **Four-Split Quadrilateral Mesh**.



Fig. 5. Four-Split Transition

A **Four-Split Quadrilateral Mesh** has the following properties:

1. There are four vertices labeled as corner vertices.
2. There is a unique vertex labeled as interior.
3. There are four vertices labeled as mid-edges.

4. Adjacent cells share corner vertices with corner vertices, and mid-edge vertices with mid-edge vertices.

The fourth property is essential to the proof of the next theorem to ensure that the faces of the **Four-Split Pyramid** given below will match appropriately.

We now define Theorem 3 which provides a sufficient condition for the existence of a transition to a **Four-Split Quadrilateral Mesh**.

Theorem 3. *If each of the dual loops of the quadrilaterals on the sphere does not self-intersect, there is a transition of the mesh to a four-split quadrilateral mesh.*

Proof: A given oriented loop in the dual of the quadrilateral mesh which does not self-intersect splits the mesh into three disjoint sets of quadrilaterals: 1) the set of quadrilaterals that lie to the right of the loop, 2) the set of quadrilaterals that compose the loop, and 3) the set of quadrilaterals that lie to the left of the loop. The quadrilaterals that compose the loop are a boundary between the quadrilaterals labeled left and right. This partition of the mesh exists because there are no self-intersections that change the orientation of the curve.

By utilizing these three sets of quadrilaterals, we can begin to add layers of hexahedra onto the quadrilaterals which will result in a four-split quadrilateral transition. There are three cases to consider when adding a layer of hexahedra (shown in Figure 6): case 1 is used when a quadrilateral lays in the regions labeled as right of the oriented loop, case 2 is utilized when a quadrilateral belongs to the loop, and case 3 is used when a quadrilateral lies to the left of the loop. We will discuss each of these cases separately.

- **Case 1:** If a quadrilateral lies to the right of an oriented loop, a single hexahedron is added on top of the quadrilateral towards the center of the sphere.
- **Case 2:** When a quadrilateral belongs to a loop, a cell of three quadrilaterals is placed as illustrated in Figure 6.
- **Case 3:** When a quadrilateral lies in the region labeled as left of the loop, two hexahedra are added on top of the quadrilateral towards the center of the sphere.

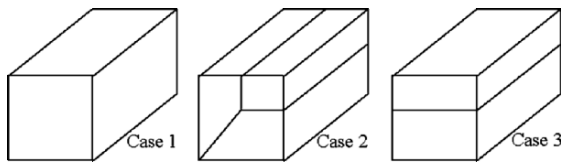


Fig. 6. Transition cell for the Four-Split transition

Thus, three types of transition cells placed at each quadrilateral of the mesh: one is a single hexahedra, another is a pair of hexahedra one on top of the other, and the third one is the one illustrated in Figure 6.

By placing the set of cells on top of each quadrilateral as described above for each of the cases, we will now show that the cells will match appropriately at the interface with each of the other sets of cells for each quadrilateral. For a single loop, take any two quadrilaterals sharing an edge with the cells placed on top of the quadrilateral as described above, and the result is one of six possible scenarios:

- Scenario A - Both quadrilaterals are in the set of ‘right’ quadrilaterals: In this case, two hexahedra are placed next to each other, and will match up appropriately.
- Scenario B - Both quadrilaterals are in the set of ‘left’ quadrilaterals: In this case two pairs of hexahedral elements are placed next to each other, and the hexahedra will match up appropriately.
- Scenario C - One quadrilateral is labeled ‘left’, and a neighbor is labeled ‘right’: This scenario is not possible, because the oriented loop with no self-intersections is the boundary that divides the ‘left’ quadrilaterals from the ‘right’ quadrilaterals by definition.
- Scenario D - One quadrilateral is labeled ‘right’ and the other belongs to the loop: The quadrilateral labeled ‘right’ must lie in the region to the right of the loop. In this case, there is one hexahedral element on the right side of the loop that is matched with an appropriately aligned hexahedra from case 2 template.
- Scenario E - One quadrilateral is labeled ‘left’ and the other belongs to the loop: Using similar reasoning to case D, the ‘left’ quadrilateral is to the left of the quadrilateral in the loop, and there are two hexahedral elements matching two hexahedral elements from the case 2 template, and the hexes match up appropriately.
- Scenario F - Both quadrilaterals belong to the loop: Label the two quadrilaterals p and p' having vertices v_1, v_2, v_3, v_4 , and v_3, v_2, v_5, v_6 respectively. The quadrilaterals p and p' share the edge v_2, v_3 . There are two possible cases: i) one quadrilateral is a successor of the other quadrilateral with respect to the loop, or ii) the two quadrilaterals are not successors of each other.
 - Case i - The quadrilaterals are traversed successively in the loop: Suppose the loop traverses quadrilateral p through edges v_4, v_1 and v_2, v_3 . If the loop traverses quadrilateral p' through v_2, v_5 and v_3, v_6 (see Figure 7), the loop must also traverse quadrilateral p' through two additional edges v_3, v_2 and v_5, v_6 . If the loop traverses quadrilateral p' at four edges, the loop is self-intersecting at p' , which is contradictory to the assumption of no self-intersections. Therefore, the loop will traverse the adjacent quadrilateral p' through edges v_3, v_2 and v_5, v_6 , and the hexes match up appropriately.

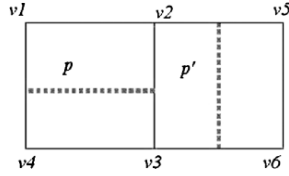


Fig. 7. This configuration is will occur only if there is a self-intersecting loop in the quadrilateral mesh.

- *Case ii* - The quadrilaterals are not successively traversed in the loop: In this case, the case 2 templates must face in opposite directions with respect to the loop; that is, the left edge of one template must be sharing the left edge of the adjacent template, or the right edge of one template is sharing the right edge of the adjacent template, as shown in Figure 8).

If the loop is traversing p through edges $v1, v2$ and $v3, v4$, the loop cannot traverse at p' through the shared edge $v3, v2$. Otherwise, p could be considered a successor or predecessor of p' ; this would lead to the loop being self-intersecting at p and contradicting our assumptions on the loop. Hence, the loop must traverse p' through the edges $v2, v5$ and $v3, v6$. Therefore, we need to show that p cannot be to the ‘left’ of p' while p' is to the ‘right’ of p . We can demonstrate that this is impossible by using basic properties of simple Jordan curves.

We create an oriented, closed loop of segments by connecting the mid-point of each edge traversed by the loop from all the quadrilaterals in the loop. This oriented, closed loop of segments is a simple Jordan curve because the loop that induced it is not self-intersecting.

Construct an oriented, open poly-segment with vertices $v1, v2, v5$. This poly-segment must intersect the oriented, closed curve induced by the loop at two different points. As a direct result of the property of simple Jordan Curves, one intersection point the tangent of the oriented closed loop must be pointing to left of the oriented poly-segment, and, at the other intersection point, the tangent must be pointing to the right.

We are, therefore, left with two possibilities: the loops are either to the right of each other, or to the left of each other. In either case, the hexahedra from the case 2 template will match up appropriately as illustrated in Figure 8.

We now return to the **Paired Partition** transition described earlier, and we notice an interesting pattern which can be exploited to reduce the number of elements in the four-split transition. Each **Cell of Six Quadrilaterals** (as shown in Figure 3) contains two types of loops: the inner loops, and the exterior loops (as described earlier). None of the inner loops intersect each other; hence, it is possible to orient all loops so that their direction is consistent. Similarly, none of the exterior loops intersect each other, and we can

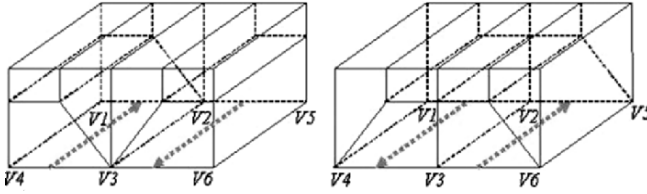


Fig. 8. Possible loop orientations in Scenario F, case ii

therefore orient these loops, as well. Thus, by adding only two layers of two splits, it is possible to transition to a four-split quadrilateral mesh. Each of the cells has two outer loops oriented to the left of each other, and the remaining loop to the right of the center loop, and placing each element within the cell distinctly into one of the scenarios described above.

As a result of this ability to distinctly orient each of the loops, we can calculate the total number of transition elements needed to convert from a **Paired Partition** to a **Four-Split Quadrilateral Mesh** as:

$$\text{number of transition elements} = \text{number of quadrilaterals to the right of the loop} + 2 * \text{number of quadrilaterals to the left} + 3 * \text{number of quadrilaterals from the loop}$$

The hexahedral elements transition the original quadrilateral elements to one where the quadrilaterals to the left and right of the mesh remain identical to the originals, but the ones along the loop are split into two quadrilaterals. The set of original chords is transferred to the transition quadrilateral mesh as follows:

1. The loop just processed is discarded from the set of chords,
2. The remaining chords are mapped onto the faces of the transition quadrilateral mesh. In particular, when a loop is projected onto one of the pair of quadrilaterals resulting from the split along the loop, it must be transverse to the loop.

The operation of adding two split transition layers described above is repeated for another loop of the remaining loops projected onto the new quadrilateral mesh that results from the transition elements. Each time a loop is processed the set of remaining loops diminishes by one. The process continues iteratively until no more loops remain from the original set. Wherever two loops intersect, a group of **Four-Split Cell** is created, because two quadrilaterals resulted from the loop processed at an earlier stage in the process, and the secondarily processed loop adds two more quadrilaterals along the transversal direction. Since each loop is processed only once, and exactly two loops cross each quadrilateral cell in traversal directions; hence, the resulting transition of quadrilaterals will be a **Four-Split Quadrilateral Mesh**. Figure 9 illustrates the results of two layers from two different loops that intersect.

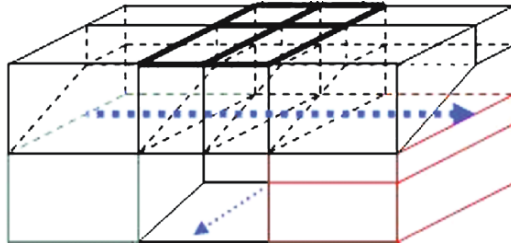


Fig. 9. Two layers result from two different loops intersecting.

An Alternative Four-Split Transition

Scott Mitchell suggested an alternative construction [10]. The alternative transition is accomplished by adding one layer of hexahedra on the elements that are either on the loop or to the left of the loop. All other quadrilaterals are left as they are. The proof of the theorem is very similar to the one above. Using his approach, the number of transition elements added at each layer is given by the formula:

$$\text{number of transition elements} = \text{number of quadrilaterals to the left of the loop} + \text{number of quadrilaterals on the loop}$$

It is critical for both four-split transitions that the quadrilateral mesh is on the sphere. A torus, for example, may have loops that do not split the domain in two regions as required by the proof. In general, non-intersecting loops could be processed simultaneously by the approach given above if the chords are carefully oriented to reduce the number of layers.

4.3 Four-Split to Closure Transitions

Once a **Four-Split Quadrilateral Mesh** is in place it is possible to transition to a constrained hexahedral mesh utilizing one of several other transitions. We now demonstrate how the remainder of the sphere can be filled using a Four-Split Pyramid, or the Geode-template to obtain an all-hexahedral mesh.

The Four-Split Pyramid Transition

Theorem 4. *A **Four-Split Quadrilateral Mesh** is the boundary a hexahedral mesh containing $16 \cdot n$ hexahedra, where n is the number of quadrilaterals before four-split division.*

Proof: This construction is done by utilizing a hexahedral decomposition of a pyramid into sixteen hexahedra (a detailed construction of the pyramid is given in [4].) This pyramid is characterized by having a **Four-Split Quadrilateral Mesh** at the base of the pyramid (the pyramid is illustrated in Figure 10). The **Four-Split pyramids** are placed inside the sphere with their bases

aligned at each of the four-split cells with the apex of pyramids being connected to the center of the sphere, and the midpoints to the corresponding midpoints of the faces to the adjacent cells as illustrated in Figure 10. The connectivity to adjacent **Four-Split Pyramids** is guaranteed by the fundamental property of the **Four-Split Quadrilateral Mesh** that ensures that corner vertices meet with corner vertices, and mid-edge vertices meet with mid-edge vertices.

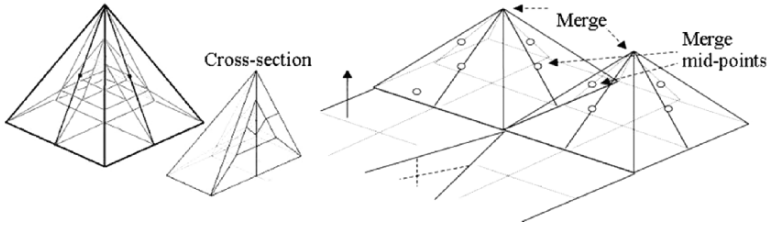


Fig. 10. A cross-sectional view of the Four-Split Pyramid. The mid nodes and the tips of each of the Four-Split pyramids are merged to ensure conformal meshes.

S. Mitchell’s Geode Template

The **Four-Split Pyramid** contains several hexahedra which share two faces or edges with neighboring elements (also known as ‘doublets’ [8]). therefore, a very attractive alternative to the **Four-Split Pyramid** is S. Mitchell’s Geode-Template [7]. The Geode-template contains more elements than the **Four-Split Pyramid**, but reduces the number of doublets in the resulting mesh. In this section, we will show how the Geode-template can be utilized in place of the **Four-Split Pyramid**.

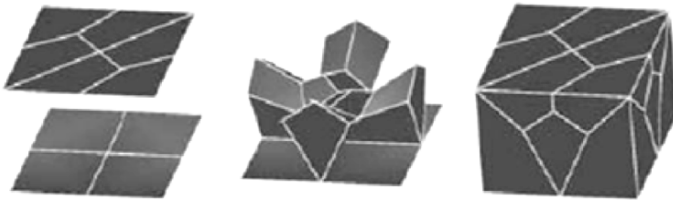


Fig. 11. The Geode-template by S. Mitchell

The Geode-Template contains 26 hexahedral elements, and contains a **Four-Split Quadrilateral Mesh** at the base. The template was designed to match a four-split quadrilateral cell to a diced tetrahedral constrained mesh. The interior of the mesh is filled by two four element hexahedral dicing of a tetrahedral element where the apex is connected at the center of the sphere.

If we cap the geode template with a pyramid split into two diced tetrahedra, the resulting hexahedral decomposition can be used in place of the **Four-Split Pyramid** described earlier.

5 A Constructive Hexahedral Existence Theorem

5.1 Solutions using the Four-Split Pyramid

Theorem 5. *A quadrilateral mesh on the sphere with an even number of quadrilaterals is the boundary of a hexahedral mesh of the interior of the sphere. The total number of hexahedra is $76 * n$ where n is the number of quadrilaterals on the boundary.*

Proof: By Theorem 1, every even-parity quadrilateral mesh on a 2-dimensional compact manifold admits a **Paired Partition**. By Theorem 2, a transition to a quadrilateral mesh can be constructed with no self-intersecting loops. By Theorem 3, the quadrilateral mesh transitions to a four-split, and, by Theorem 4, the quadrilateral mesh is the boundary of a hexahedral mesh.

The total number of hexahedral elements is given by

*number of hexahedra = number of elements to resolve self-intersecting loops + number of elements to transition to a four-split + $16 * \text{number of four-split pyramids}$*

1. The number of hexahedra added by applying the transition that removes self-intersecting loops illustrated in Figure 3 is n .
2. For each of the six quadrilaterals of each cell, the total number of hexahedra needed to transition to a four-split is 9; hence the total number of hexahedra needed to transition the mesh to a four-split is $9 * 6 * n/2$.
3. The total number of hexahedral elements per cell needed to solve the four-split is $16 * 6 * n/2$.

Hence, the total number of hexahedra to fill the interior is $n + 48 * n + 27 * n$ which equals $76 * n$ total elements.⁵

5.2 Solutions Using the Geode-Template

Utilizing the **Paired Partition** transition and the **Four-Split** transition, but replacing the **Four-Split Pyramid** with the modified geode-template results in a solution with fewer ‘doublet’ entities in the mesh. This solution requires:

1. $n + 9 * 6 * n/2$ hexahedra needed to transition to the **Four-Split Quadrilateral Mesh**

⁵Using Mitchell’s alternative Four-Split transition in the section titled **An Alternative Four-Split Transition**, the number of hexahedra required for the constrained solution is $54 * n$.

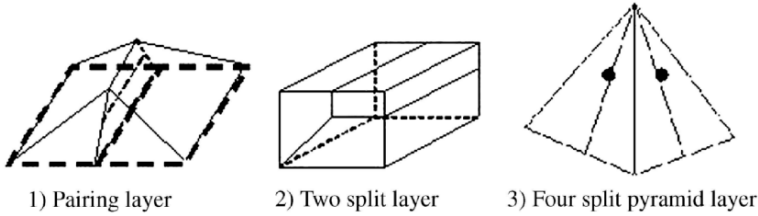


Fig. 12. The various transition layers used in the constructive proof.

- 2. plus, $26*6*n/2$ elements from the Geode-Template
- 3. plus, $4*2*6*n/2$ elements for the diced tetrahedra with apex at the center of the mesh.

This new solution of the constrained hexahedral mesh contains a total of $130*n$ elements,⁶ but will contain significantly less doublets than a solution produced by the **Four-Split Pyramid**.

5.3 Solutions Using a ‘Pillowed’ Four-Split Pyramid

Another slightly different approach replaces the transitions from the paired partition and the **Four-Split Pyramid** by transition cells but do not contain the doublet elements identified earlier. The doublet elements can be removed in the transition layers by applying the doublet-pillowing technique described by S. Mitchell and T. Tautges [8]. The base of the new **Four-Split Pyramid** will still be a four-split quadrilateral cell, and the resulting transition cells still have no self-intersecting loops, and will not contain any doublets. However, by removing the doublets the resulting solution to the constrained hexahedral problem requires $5396*n$ hexahedral elements.

6 Applications

In this section we demonstrate constrained hexahedral solutions to the quadrilateral meshed boundaries described as Schneider’s Pyramid, the Quadrilateral Octahedron, and Eppstein’s Cube. In all cases, the solution is a direct result from Theorem 5. The solutions are presented by conveniently numbering the quadrilaterals such that the quadrilateral admits a **Paired Partition** of the form $PQ = \{\{q1, q2\}, \dots, \{q2k-1, q2k\}, \dots, \{qn-1, qn\}\}$ where n is the number of quadrilaterals for each case; the total number of hexahedra used to solve the constrained hexahedral mesh will be $76*n$ or $130*n$ depending on which solution is used. The source code used in generating these solutions is available at [1].

⁶Using Mitchell’s alternative Four-Split transition in the section titled **An Alternative Four-Split Transition**, the number of hexahedra required for the constrained solution using the Geode-template is $112*n$.

6.1 Schneider's Pyramid

Figure 13 contains an open view of Schneider's pyramid with each of the boundary quadrilaterals being numbered. There are 16 quadrilaterals on the boundary of the pyramid resulting in a total of 1216 hexahedral elements being generated using the **Four-Split Pyramid**, or 2080 hexahedral elements if the Geode-template is utilized.

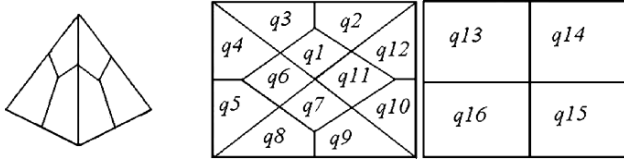


Fig. 13. Schneider's pyramid

6.2 Quadrilateral Octahedron

Figure 14 contains an open view of the Quadrilateral Octahedron with each of the boundary quadrilaterals being numbered. There are 8 quadrilaterals on the boundary of the pyramid resulting in a total of 608 hexahedral elements being generated using the **Four-Split Pyramid**, or 1040 hexahedral elements if the Geode-template is utilized.

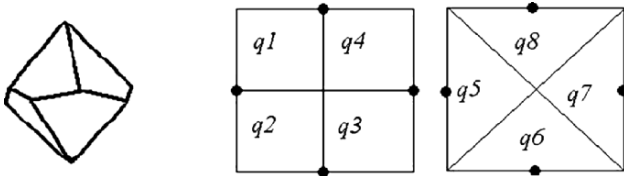


Fig. 14. Quadrilateral Octahedron

6.3 Eppstein's Cube

In Eppstein's original construction, there are two sets of quadrilateral cubes used to transition to the tetrahedral based mesh. One had sixteen elements, and the other eighteen. These cubes contain quadrilateral doublets (i.e faces that share two edges with an adjacent neighbor). There is another version of Eppstein's cubes with 22 and 20 quadrilateral non-degenerate elements respectively. We focus on the 16-quadrilateral cube shown in Figure 15.

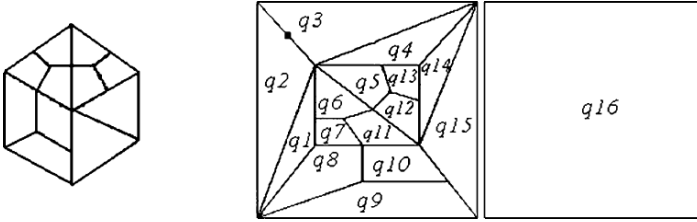


Fig. 15. Eppstein's Cube

The value of n for Eppstein's cube shown in Figures 15 is 16. A total of 1216 hexahedral elements are needed to resolve the constrained mesh utilizing the **Four-Split Pyramid** solution, or 2,080 hexahedra if the Geode-template solution is utilized.

7 Conclusions

The construction presented in this paper demonstrates a solution of S. Mitchell's existence theorem. However, this construction is different from the constructions utilized in S. Mitchell's proof. Indeed, his solution cannot lead to the construction given in this paper. In the approach outlined in the original proof of S. Mitchell's existence theorem, a mesh that contains a loop with an odd number of intersections will have to be connected through an interior surface to another loop with an odd number of intersections on the boundary. The **Paired Partition** transition has the very interesting property of locally connecting all of the loops defined by the initial quadrilateral boundary mesh through interior surfaces, and simultaneously creating a transition to a quadrilateral mesh which contains no self-intersecting loops.

We have given results for Schneider's pyramid and Eppstein's cube and Quadrilateral Octahedron, with element counts needed to generate a hexahedral topology in these solids using the construction outlined in this paper. The solution presented for the sphere can be extended to the case of the torus and compact 2-dimensional manifolds in general by using the Geode Template coupled with a constrained tetrahedral mesh. From the construction, it follows that it is possible to generate a hexahedral decomposition with linear edges for a quadrilateral mesh of a convex region. The question of finding a general construction with a minimal number of elements with linear edges is open.

Acknowledgements

The authors would like to express appreciation to Scott Mitchell for his review of the paper, and the many additional insights provided. We would also like to

thank Cynthia Phillips, M. Gopi, and David Eppstein for their guidance and expertise in graph theory which enabled us to formulate a proof for perfect matching of closed quadrilateral meshes.

References

1. C. D. Carbonera, ‘A Constructive Approach to Hexahedral Mesh Generation: Algorithm Implementation’, source code available from <http://carbonera.uprr.pr/>.
2. M. Bern, D. Eppstein, P. K. Agarwal, N. Amenta, P. Chew, T. Dey, D. P. Dobkin, H. Edelsbrunner, C. Grimm, L. J. Guibas, J. Harer, J. Hass, A. Hicks, C. K. Johnson, G. Lerman, D. Letscher, P. Plassmann, E. Sedgwick, J. Snoeyink, J. Weeks, C. Yap, , and D. Zorin. Emerging challenges in computational topology. In *NSF-funded Workshop on Computational Topology*, pages –, 1999.
3. C. D. Carbonera and J. F. Shepherd. On the existence of a perfect matching for 4-regular graphs derived from quadrilateral meshes. *SCI Institute Technical Report*, UUSCI-2006-021, 2006.
4. Carbonera Pyramid, available from <http://www-users.informatik.rwthachen.de/~roberts/SchPyr/index.html>.
5. D. Eppstein. Linear complexity hexahedral mesh generation. In *12th ACM Symposium on Computational Geometry*, pages 58–67. ACM, 1996.
6. S. A. Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volumes. In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume Lecture Notes in Computer Science: 1046, pages 465–476, 1996.
7. S. A. Mitchell. The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh. In *Proceedings, 7th International Meshing Roundtable*, pages 295–305. Sandia National Laboratories, October 1998.
8. S. A. Mitchell and T. J. Tautges. Pillowing doublets: Refining a mesh to ensure that faces share at most one edge. In *Proceedings, 4th International Meshing Roundtable*, pages 231–240. Sandia National Laboratories, October 1995.
9. P. J. Murdoch and S. E. Benzley. The spatial twist continuum. In *Proceedings, 4th International Meshing Roundtable*, pages 243–251. Sandia National Laboratories, October 1995.
10. Scott Mitchell to Jason Shepherd, email entitled ‘constructive approach to constrained hex mesh generation’, dated March 31, 2006.
11. Schneiders Pyramid Open Problem, <http://www-users.informatik.rwthachen.de/~roberts/open.html>.
12. T. Suzuki, S. Takahashi, and J. F. Shepherd. Practical interior surface generation method for all-hexahedral meshing. In *Proceedings, 14th International Meshing Roundtable*, pages 377–397. Sandia National Laboratories, September 2005.

Automatic Hexahedral Mesh Generation with Feature Line Extraction

Masayuki Hariya¹, Ichiro Nishigaki², Ichiro Kataoka³, Yoshimitsu Hiro⁴

1, 2, 3 Mechanical Engineering Research Laboratory, Hitachi, Ltd.
832-2, Horiguchi, Hitachinaka, Ibaraki, Japan

¹masayuki.hariya.sa@hitachi.com

²ichiro.nishigaki.mp@hitachi.com

³ichiro.kataoka.pq@hitachi.com

⁴Industrial Information System Division, Hitachi, Ltd.

Omori Bellport B Bldg., Minami Oi 6-chome, Shinagawa-ku, Tokyo,
Japan

yoshimitsu.hiro.yc@hitachi.com

Abstract. An improved method using feature line extraction is described for automatically generating hexahedral meshes for complex geometric models that automate the normally interactive operations (such as model editing). Testing showed that the time taken for these interactive operations was significantly reduced, making it possible to quickly generate hexahedral meshes with sufficient quality for complex models. Application of this method to mechanical part models showed that it shortened the time to generate a mesh in about 10% the time required with the previous method.

1. Introduction

The use of digital engineering in manufacturing to shorten product-development times is becoming more widespread. Both three-dimensional computer-aided-design (3D-CAD) modeling and computer-aided engineering (CAE) are widely used by product designers. A promising approach to reducing the CAE preprocessing time is automatic mesh generation [1].

Hexahedral meshes are widely used in numerical analyses because they generally provide more accurate results than tetrahedral meshes. They are

usually generated by hand by dividing a complex geometric model into simple geometrical blocks. As this requires many interactive procedures, various methods for automatically generating them have been proposed [2–6].

One is based on the Octree method [2]. It generates small cubes inside the geometric model and generates a mesh by mapping the surfaces of the cubes on the boundary onto the surfaces of the geometric model. Another method automates the division of the blocks [3–6].

Our previous method for automating the process is based on shape recognition and boundary fitting [7–8]. A unique shape-recognition technique is used to change a geometric model into an approximate one consisting of straight lines. Boundary fitting maps small cubes that are generated by dividing the approximate model, onto the geometric and generate hexahedral meshes. This reduces the number of interactive procedures, and thus reduces the time to generate a hexahedral mesh by about 60%. However, this sometimes can not generate meshes automatically in case of some complicated models. We have now improved our method by also automating the model-editing task by using feature line extraction.

2. Previous Mesh Generation Method

2.1 Shape Recognition and Boundary Fitting

The procedure for generating hexahedral meshes using shape recognition and boundary fitting is illustrated in Figure 1.

Step 1-1: An approximate model (Figure 1(b)), called a recognition model, is constructed from a geometric model (Figure 1(a)) by using only the lines parallel to the Cartesian axes. The recognition model must be topologically identical and geometrically similar to the geometric model.

Step 1-2: The edge lengths of the recognition model are adjusted to the nearest integral multiple of the standard element sizes, and the model is divided into cubes, producing what is called a mapping model (Figure 1(c)).

Step 1-3: The mapping model is mapped onto the original geometric model by boundary fitting, generating the final hexahedral mesh (Figure 1(d)). The input data is the standard element size.

The shape recognition is based on fuzzy-logic theory. All edges of the geometric model are arranged parallel to the Cartesian coordinates (ξ, η, ζ) of the mapping space. The axis where an edge is arranged is determined based on information obtained from the geometric model, such as the

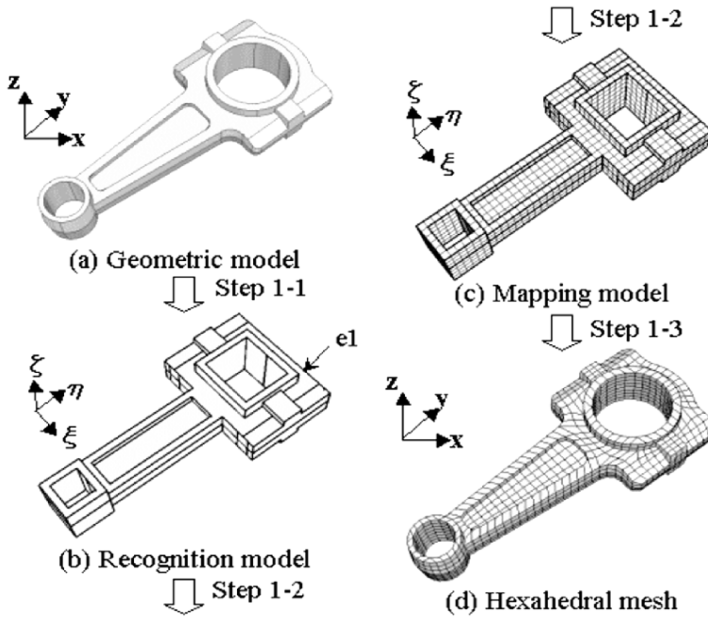


Figure 1 Procedure for generating hexahedral meshes using shape recognition and boundary fitting

direction vectors of the edges and the vertex angle of two edges. The directions of the coordinate axes are called the assigned directions. The shape of the recognition model is determined from the assigned directions of the edges. For example, in Figure 1(b), the assigned direction of edge e_1 is the ξ axis.

2.2 Problems

With this mesh generation procedure, two typical errors may result in a failure to generate a recognition model.

(1) If the geometric model contains surfaces where the boundary has three or fewer edges, a recognition model that is topologically equal to the geometric model cannot be generated.

(2) If the assigned edge directions are not correct, a recognition model cannot be generated even if the geometric model is topologically correct.

Interactive operations (adding or deleting a line to or from the geometric model, correcting an assigned direction, etc.) are necessary in such cases before the recognition model can be generated.

3. Improved Mesh Generation Method

Our improved method avoids the two problems described above by using feature line extraction to generate a recognition model. The procedure for generating a recognition model is illustrated in Figure 2.

Step 2-1: Triangular meshes (Figure 2(b)) are generated on the surface of the geometric model [Figure 2(a)].

Step 2-2: A feature-shape model [Figure 2(c)] is generated by extracting the feature lines from the boundaries of the triangular meshes [Figure 2(b)].

Step 2-3: The recognition model [Figure 2(d)] is generated from the feature-shape model.

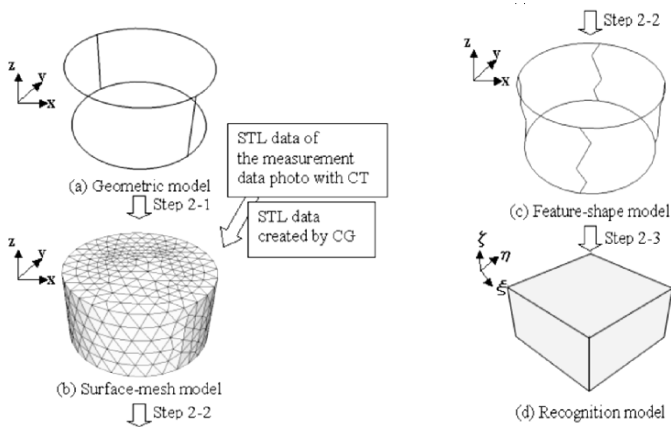


Figure 2 Procedure for generating recognition model using feature line extraction

This improved method has two key features. First, the feature lines used for generating the recognition model can be automatically extracted from the triangular meshes on the geometric model. This means that the generation of the recognition model does not depend on the topology of the geometric model. Therefore, a hexahedral mesh can be generated without having to manually divide a complex geometric model into simple geometrical blocks. Second, the feature lines are selected by taking the assigned directions into consideration using a method that will be described below. This means that a hexahedral mesh can be generated without correcting the assigned directions.

Furthermore, STL data of the measurement data photo with computed tomography (CT) equipment and STL data created by a computer graphics (CG) program can be used for the surface-mesh model (Figure 2(b)). This enables the use of finite-element analysis, which uses a high-quality hexahedral mesh about a wide range of steps of the engineering.

As shown in Table 1, the current method (based on feature line extraction) is better than the previous one (based on fuzzy logic) in terms of the automation of mesh generation and the application range, while the previous method is better in terms of the generation of meshes that represent the geometric model. This is because, in the current method, the shape of the generate meshes is affected by the surface meshes used.

Table 1. Comparison between mesh generation methods

	Previous	Current
Automation of mesh generation	average	good
Representation of geometric model	good	average
Application range	wide	very wide

3.1 Generating Recognition Model

The recognition model is generated as shown in Fig. 3.

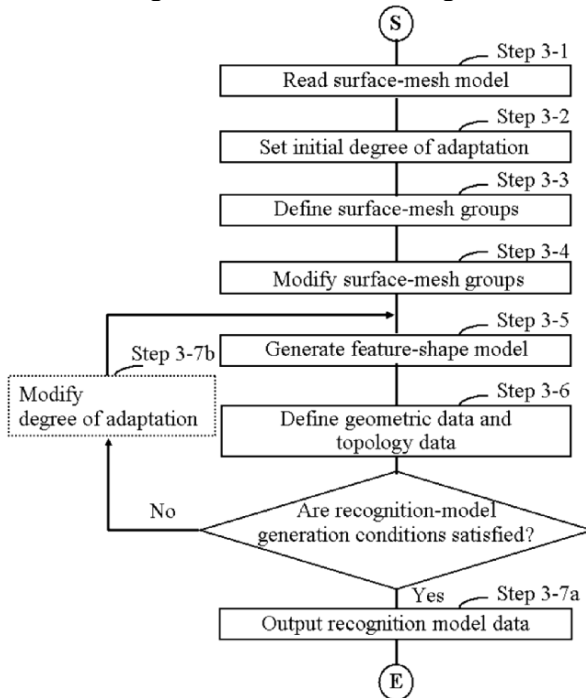


Figure 3 Steps in generating recognition model

3.1.1 Read Surface-Mesh Model: Step 3-1

The recognition-model generation program reads the surface-mesh model and registers it in a data table. The mesh corresponding to the surface-mesh model is called the mesh face, the boundaries of the mesh face are called the mesh lines, and the nodal points of the mesh face are called mesh points.

3.1.2 Set Initial Degree of Adaptation: Step 3-2

Unit normal vector $v = (vcx, vcy, vcz)$ is set as the initial degree of adaptation $P = (P_\xi, P_\eta, P_\zeta)$ for each mesh face. This degree is what a mesh face is assigned in the direction of a particular coordinate axis (ξ, η, ζ) in the mapping space. The normal vector can be calculated from the coordinates of the mesh points composing the mesh face.

3.1.3 Define of Surface-Mesh Groups: Step 3-3

Surface mesh groups are defined by grouping the mesh faces based on their degrees of adaptation.

First, the assigned directions of the mesh faces are determined: $P_\xi, P_\eta,$ and P_ζ are compared, and the assigned directions are taken as the maximum absolute values of the axial directions. The assigned directions are modified based on the signs of the degrees of adaptation. For example, if $P_\xi = 1.0, P_\eta = 0.0,$ and $P_\zeta = 0.0,$ the initial assigned direction is the ξ axis. The actual direction is $-\xi$ because the sign of P_ξ is minus.

Next, the mesh faces with the same assigned direction and sharing a mesh line are defined as a surface- mesh group. The assigned direction of the group is the assigned direction of the mesh faces composing it.

Example surface-mesh groups are shown in Figure 4. Mesh faces with the same shading represent one surface-mesh group. The assigned direction of surface-mesh group 1 (Gr1) is $+\xi,$ based on the assigned direction of mesh face MF1.

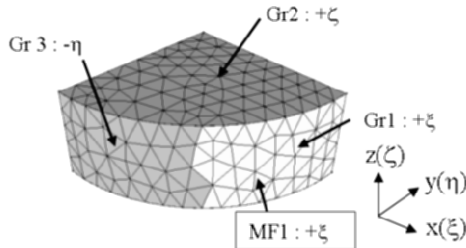


Figure 4 Surface-mesh groups

3.1.4 Modify Surface-Mesh Groups: Step 3-4

The defined surface-mesh groups are modified based on group correction rules, which are necessary conditions for relating two or more surface-mesh groups. Here we consider two example conditions.

Condition 1: A surface-mesh group must have four or more neighboring surface-mesh groups.

Condition 2: The assigned direction of a surface-mesh group should not be in the same axial direction as the assigned direction of a neighboring surface-mesh group.

Example surface-mesh groups not satisfying these conditions are shown in Figure 5. In Figure 5(a), Gr1 touches only three surface-mesh groups. In Figure 5(b), Gr1 and Gr2 are both assigned the ξ -axis direction.

An example of modifying a surface-mesh group to satisfy Condition 1 is shown in Figure 6. First, the length of the line shared by each pair of surface-mesh groups is calculated. The shared line length for Gr1 and Gr2 is L_2 , the shared line length for Gr1 and Gr3 is L_3 , and the shared line length for Gr1 and Gr4 is L_4 . A surface-mesh group that has three or less neighboring surface-mesh groups is combined with the one with which it has the longest shared line, and a new surface mesh group is generated. Because L_3 is the longest shared line for Gr1, Gr1 is combined with Gr3, creating Gr5.

An example of modifying a surface-mesh group to satisfy Condition 2 is shown in Figure 7. One of the two groups with the same assigned direction is divided in two and setting a new assigned direction. In the figure, Gr5 is created from Gr1 and is assigned the direction of the ζ axis because the assigned directions of the neighboring surface-mesh groups are ξ and η . The actual assigned direction is thus $-\zeta$ based on the z component of the average normal vector of the mesh faces composing Gr5.

Many other group correction rules are used besides the two described above.

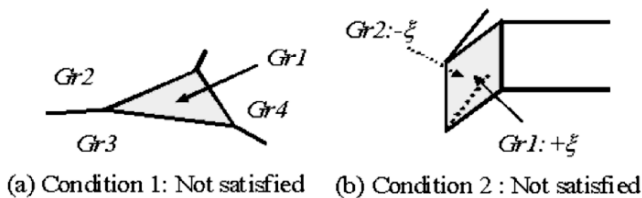


Figure 5 Irregular-mesh groups

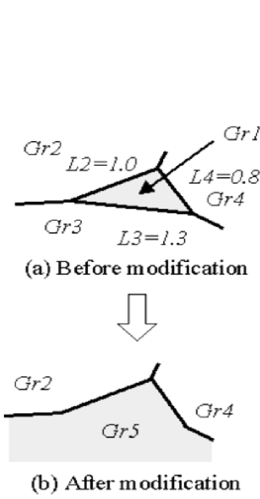


Figure 6 Group correction rule to satisfy Condition 1

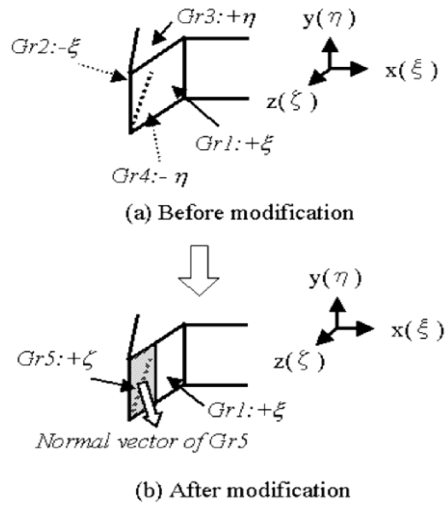


Figure 7 Group correction rule to satisfy Condition 2

3.1.5 Generate Feature-Shape Model: Step 3-5

Feature surfaces are defined as the defined and then modified surface-mesh groups, and their assigned directions are the assigned directions of the groups. A feature line is defined as a set of mesh -lines shared by two different surface-mesh groups. The set of mesh lines, ML1–4, for the surface-mesh model in Figure 8 comprising the boundary between GrA and GrB is defined as one feature line.

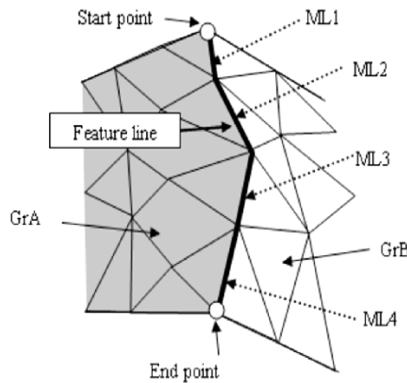


Figure 8 Feature line

The assigned direction of a feature line is determined from the combination of the assigned directions of the two surface-mesh groups to which the feature line belongs. The relation between the assigned directions of the surface-mesh groups and the assigned direction of the feature line is shown in Table 2.

Table 2 Assigned direction of feature line

Assigned direction of surface-mesh group A	Assigned direction of surface-mesh group B	Assigned direction of feature line
ξ	η	ξ
η	ξ	ξ
ξ	ξ	η


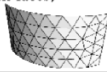

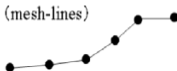


The sign of the assigned direction is the sign of the direction vector of the feature line. For example, if the assigned direction is ξ and the sign of the x value of the direction vector is minus, the assigned direction is modified to $-\xi$.

3.1.6 Define Geometric Data and Topology Data: Step 3-6

The data for the feature-shape model are defined using boundary representation, which is expressed using topology data showing the connection between geometric elements and using geometric data showing the shape of each geometric element.

The correspondence between the geometric model and the feature-shape model is shown in Table 3. The geometric data for the feature points are the 3-D coordinates of the points. Moreover, the feature surface is composed of mesh -faces, and the feature line is composed of mesh -lines. The topology data has a hierarchical structure like the geometric model.

Table 3 Correspondence between geometric model and feature-shape model

	Geometric model	Feature-shape model
2 nd -order geometric element	Surface 	Feature face (mesh-faces) 
1 st -order geometric element	Line 	Feature line (mesh-lines) 
0 th -order geometric element	Point 	Feature point (mesh-point) 

3.1.7 Check Feature-Shape Model and Dssigned Directions

To generate the recognition model, the feature-shape model and assigned directions must satisfy the following recognition-model generation conditions.

<Condition 1> At least one feature surface is assigned the plus or minus direction of one of the (ξ, η, ζ) axes.

<Condition 2> Neighboring feature surfaces are not assigned to the opposite direction of same axial direction.

<Condition 3> There must be at least one feature line comprising the feature surface boundary, which is assigned the plus or minus direction of two coordinates axes.

<Condition 4> Neighboring feature lines are not assigned to the opposite direction of same axial direction.

Example satisfied and unsatisfied conditions are shown in Figure 9.

If the conditions are satisfied, Step 3-7a is carried out. If the conditions are not satisfied, Step 3-7b is carried out.

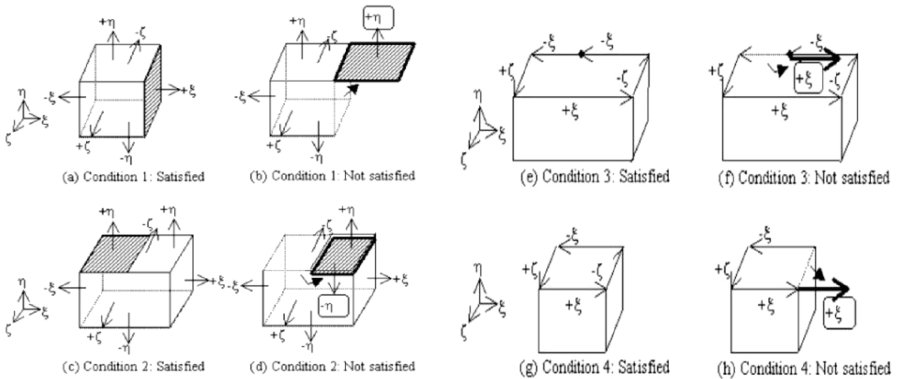


Figure 9 Satisfied and unsatisfied recognition-model generation conditions

3.1.8 Output Recognition Model Data: Step 3-7a

The feature-shape model and assigned directions are registered in the database.

3.1.9 Modify Degree of Adaptation: Step 3-7b

The degree of adaptation of the mesh face (MF) is modified based on the degrees of adaptation of the neighboring mesh faces. The number of neighboring mesh faces is denoted by n . The degree of adaptation of a MF before modification is denoted by $P_b = (P_{\xi b}, P_{\eta b}, P_{\zeta b})$, and the degree of

adaptation after modification is denoted by $P = (P_\xi, P_\eta, P_\zeta)$. The degree of adaptation before correction of neighboring mesh -faces MF(i): ($I = 1, 2, \dots, n$) is denoted by $P_{bi} = (P_{\xi bi}, P_{\eta bi}, P_{\zeta bi})$, and the degree of adaptation of a MF after modification is calculated as follows.

$$\left\{ \begin{array}{l} P_\xi = P_{\xi b} + \frac{\beta}{n} \sum_{i=1}^n P_{\xi bi} \\ P_\eta = P_{\eta b} + \frac{\beta}{n} \sum_{i=1}^n P_{\eta bi} \\ P_\zeta = P_{\zeta b} + \frac{\beta}{n} \sum_{i=1}^n P_{\zeta bi} \end{array} \right. \quad (1)$$

$$\left\{ \begin{array}{l} P'_\xi = P_\xi \\ P'_\eta = P_\eta \\ P'_\zeta = P_\zeta \\ val = \sqrt{P'_\xi \times P'_\xi + P'_\eta \times P'_\eta + P'_\zeta \times P'_\zeta} \\ P_\xi = P'_\xi / val \\ P_\eta = P'_\eta / val \\ P_\zeta = P'_\zeta / val \end{array} \right. \quad (2)$$

In the equations, β is a parameter reflecting the speed of modification. Equation (1) is used to smooth the degree of adaptation, and equation (2) is used to normalize the degree of adaptation calculated using equation (1). Small surface-mesh groups are removed at the time of a surface mesh-group definition by modification, as described above. As a result, the generation of the recognition model is simplified.

4. Examples of Generated Meshes

We extracted feature lines to generate a hexahedral mesh of a mechanical-part model (Figure 10(a)) for use in structural analysis. Figure 10(b) shows the surface-mesh model used to generate the mesh. The feature-shape model (Figure 10(c)) was generated by extracting the feature lines. Not all

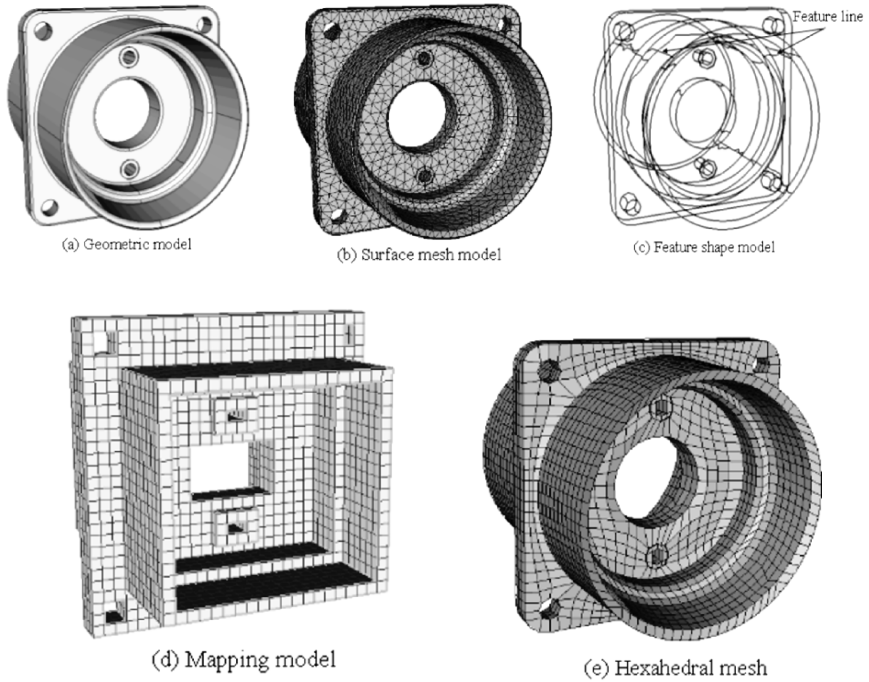


Figure 10 Procedure for generating hexahedral mesh for mechanical part

the feature lines corresponded to the ones in the geometric model. Figure 10(d) shows the mapping model, and Figure 10(e) shows the generated hexahedral mesh.

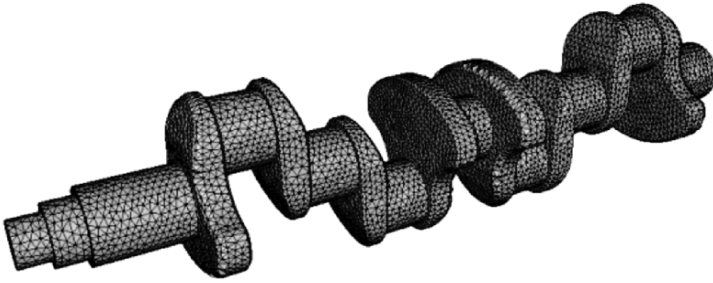
Table 4 summarizes the mesh-generation times for the previous and current methods. The current method requires no time for interactive operations, and its calculation time is about half that of the previous one because there is less trial and error. As a result, its mesh generation time is about 10% that of the previous one.

Two other examples are shown in Figures 11 and 12. The crank-shaft (Figure 11(b)) has 8200 elements, and the connecting rod (Figure 12(b)) has 1788. The calculation time for the former was 30 sec, and that for the latter was 5 sec. In both cases, the hexahedral mesh was automatically generated without interactive operations, such as line creation necessary for recognition model.

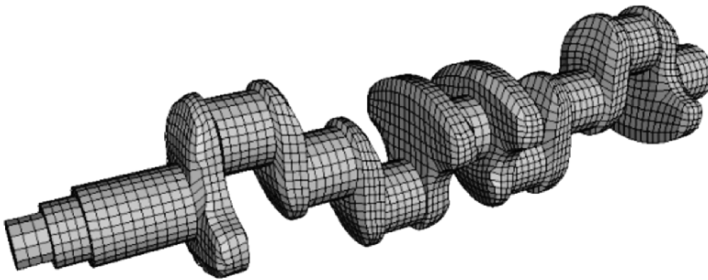
Our improved method does not require the operator to perform any skillful interactive operations such as block division. That is, someone without any specialized knowledge can generate hexahedral meshes.

Table 4 Comparison of mesh generation times (min)

	Previous method	Current method
Calculation	5	2
Interactive operation	15	0
Total	20	2

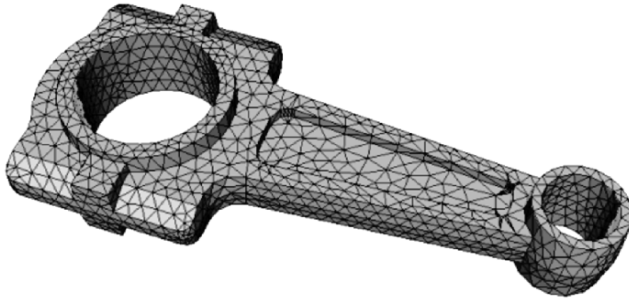


(a) Surface-mesh model

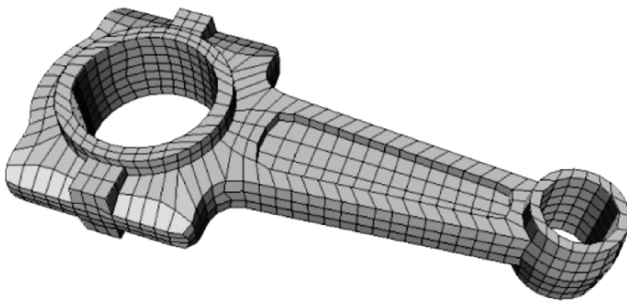


(b) Hexahedral mesh

Figure 11 Surface-mesh model and hexahedral mesh for crank shaft



(a) Surface-mesh model



(b) Hexahedral mesh

Figure 12. Surface-mesh model and hexahedral mesh for connecting rod

5. Conclusion

Our improved method for automatically generating hexahedral meshes can be used to reduce the number of interactive operations needed to generate meshes. Application to complex geometric models showed that it can

(1) automatically extract feature lines from the boundary between surface meshes, enabling a recognition model to automatically be generated and a hexahedral mesh to be quickly generated, and

(2) generate a hexahedral mesh of a mechanical-part model in about 10% the time required with the previous method.

References

1. Takamura, A. Kitani and T. Sasaki, "Automatic mesh generation for vehicle aerodynamics analysis", VDI-Berichte, (1998) 979–988
2. R. Schneiders, "Octree-based generation of hexahedral element meshes", Proc. 5th Int. Meshing Roundtable '96, SAND 96-2301 (1996) 205–215
3. T. Blacker, "The Cooper tool", Proc. 5th Int. Meshing Roundtable '96, SAND 96-2301 (1996) 13–30
4. B.Y. Shin and H. Sakurai, "Automated hexahedral mesh generation by swept volume decomposition and recombination", Proc. 5th Int. Meshing Roundtable '96, SAND 96-2301 (1996) 273–280
5. R. Taghavi, "Automatic block decomposition using fuzzy logic analysis", Proc. 9th Int. Meshing Roundtable '00, (2000)
6. N. A. Calvo and S. R. Idelsohn, "All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision", Comput. Methods Appl. Mech. Engrg. 182 (2000) 371–378
7. H. Takahashi and H. Shimizu, "A general purpose automatic mesh generation using shape recognition technique", Compt. Engrg. ASME 1 (1991) 519–526
8. N. Chiba, I. Nishigaki, Y. Yamashita, C. Takizawa, and K. Fujishiro, "A flexible automatic hexahedral mesh generation by boundary-fit method", Comput. Methods Appl. Mech. Engrg. 161 (1998) 145–154

Unconstrained Paving and Plastering: Progress Update

Matthew L. Staten, Robert A. Kerr, Steven J. Owen, Ted D. Blacker

Sandia National Laboratories*, Albuquerque, NM, U.S.A.
mlstate@sandia.gov, rakerr@sandia.gov, sjowen@sandia.gov,
tdblack@sandia.gov

Summary: Unconstrained Paving and Plastering [1] were introduced as new methods of generating all-quadrilateral and all-hexahedral finite element meshes. Their introduction was after preliminary conceptual studies. This paper presents an update on Unconstrained Paving and Plastering after significant implementation and conceptual development.

1 Introduction

Modeling and simulation has become an essential step in the engineering design process. Modeling and simulation can be used during either the original design phases, or on assessment of existing designs. In either case, the end result is increased confidence in the design, faster time to market, and reduced engineering cost.

An essential step in modeling and simulation is the creation of a finite element mesh which accurately models the geometric features of the model being analyzed. Meshes generated for three-dimensional models are typically composed of either all-tetrahedral or all-hexahedral elements. Some methods exist for the generation and analysis of hybrid meshes

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

The Submitted manuscript has been authored by a contractor for the United States Government under contract. Accordingly, the United States Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for United States Government purposes.

which contain a mix of element types. However, for traditional finite element modeling of continuum mechanics, a single element type is most often used.

Quite a debate has emerged over the advantages and disadvantages of hexahedral versus tetrahedral elements. Tetrahedral meshes are typically much easier to generate. On complicated models with complex geometric features, the time savings on generating a tetrahedral mesh rather than a hexahedral mesh can be orders of magnitude with the current meshing technology. However, the benefit of hexahedral elements is that they often perform better in the analysis stage [2,3,4].

Regardless, the bottom line is that customers of finite element meshing software continue to demand improved ability to generate hexahedral elements. This demand is what drives research in hexahedral mesh generation.

Unconstrained Plastering [1] is a new method for the automatic generation of hexahedral meshes. Unconstrained Plastering continues to show promise, although several technical and implementation challenges remain. This paper presents the current status and some of the challenges which are currently being addressed.

2 Previous Research

The work presented in this paper is built upon the work previously presented in [1], which includes an extensive description of previous research done on hexahedral mesh generation at institutions across the world. If the reader is unfamiliar with the existing research in hexahedral mesh generation or the current state-of-the-art in hexahedral mesh generation, they are encouraged to read the previous research section in [1]. Rather than repeating that information here, the previous research described in this paper will be limited to summarizing Unconstrained Paving and Plastering, and other algorithms which directly contributed to their development.

Paving [5] has been shown to be a robust and efficient solution to the quadrilateral surface meshing problem. However, its three-dimensional extension, Plastering [6,7,8], has not done the same for hexahedral mesh generation. Plastering calls for a pre-meshed boundary, which is created without considering global mesh topology. Fronts are then created, from which hexahedral elements are advanced into the solid in an element-by-element fashion. As fronts collide, complex configurations of closely-spaced randomly-oriented quadrilaterals yield complex unmeshed voids which Plastering is rarely able to resolve. As a result, traditional Plastering is able to completely mesh only simple primitive models with carefully pre-meshed boundaries. Plastering's inability to mesh more complex

solids stems from its element-by-element geometric approach and the added constraints of a pre-meshed boundary. Like Paving, Plastering considers only local element connectivities, with a high priority placed on incremental nodal placement and single element topology. Although this approach worked well in Paving for two dimensional surface meshing, the extra degree of freedom in three dimensions proves that more global consideration of hexahedral topology is required.

Learning from the experience of Plastering, Whisker Weaving [9,10] was developed with an emphasis on global hexahedral topology. The concept of the dual, or Spatial Twist Continuum [11] was key to the development of Whisker Weaving. Like Plastering, Whisker Weaving also starts from a pre-defined boundary quad mesh. Each quad on the boundary represents a whisker, or incomplete chord in the dual. The topology of the boundary quad mesh is traversed until groups of three or more boundary quadrilaterals are found whose corresponding whiskers could be advanced, or crossed, forming the topology of a single hexahedral element. The spatial locations of interior nodes are not calculated until the topology of the entire mesh is determined. Thus, formation of hexahedral element topology is guided by near-exclusive consideration of mesh topology logic. Geometric characteristics of the solid are considered secondary to the overall mesh topology. This is in stark contrast to Plastering which does nearly the opposite. Whisker Weaving is able to successfully generate hexahedral topology for a wide spectrum of solid geometries. However, because it leaves geometric positioning of interior nodes until after the entire mesh topology has been determined, Whisker Weaving is unable to make any guarantees on reasonable element quality. In practice, the element qualities produced by Whisker Weaving are rarely adequate, and are often inverted.

Research on Plastering and Whisker Weaving has shown that any algorithm which attempts to automatically generate hexahedral meshes must take both model topology as well as geometric model characteristics into consideration. Failure to consider geometric features of a solid will almost always result in poor element quality. Failure to consider global mesh and model topology will almost always result in a failure to generate a valid hexahedral mesh topology.

It is with this background that research on Unconstrained Paving and Plastering began. The authors introduced the concepts of Unconstrained Paving and Plastering in [1], and briefly summarize them here for clarity. Based on recent research and development efforts, this paper details new discoveries that help move this technology closer to a general all-purpose hexahedral mesh generator.

Unconstrained Paving and Plastering removes the constraint of a pre-meshed boundary. This allows the meshing process to consider more

global model topologies without being constrained by local mesh anomalies. The domain is then systematically partitioned through the advancement of fronts. In traditional advancing front methods [5,6,12], individual solid elements are generated by following geometric reasoning to build individual nodes, edges and faces, starting from a predefined boundary mesh and advancing inwards. In contrast, Unconstrained Paving and Plastering advance geometric layers or partitions independent of element distribution. Unconstrained Paving and Plastering delay the final definition of elements until it is absolutely necessary, thus removing any artificial constraints that a pre-meshed boundary imposed.

Unconstrained Paving and Plastering partition the domain into regions classified based on the number of remaining degrees of freedom. The Spatial Twist Continuum [11] defines quadrilateral elements as the intersection of two chords and hexahedral elements as the intersection of three chords. As such, a domain which is to be meshed with quadrilaterals must constrain two degrees of freedom for the entire domain corresponding to two chords required for each quadrilateral. Similarly, a domain which is to be meshed with hexahedra must constrain three degrees of freedom for the entire domain corresponding to the three chords required for each hexahedra.

Figure 1 illustrates the meshing of a simple surface with Unconstrained Paving. Unconstrained Paving systematically partitions the surface into sub-regions which are classified as either:

- unmeshed voids (white regions in Figure 1, no degrees of freedom are constrained)
- connecting tubes (light gray regions in Figure 1, one degree of freedom is constrained), or
- final elements (dark gray regions in Figure 1, two degrees of freedom are constrained).

For Unconstrained Plastering, the regions are classified as either:

- unmeshed void (no degrees of freedom are constrained),
- connecting tubes (one degree of freedom is constrained),
- connecting webs (two degrees of freedom are constrained), or
- final elements (three degrees of freedom are constrained).

A front advancement as shown in Figure 1a introduces an unconstrained row of elements with an undetermined number of quadrilaterals. The advancement of a row essential introduces a single new chord to the dual of the eventual mesh. However, the number of quads in this row or chord is left unconstrained at this point. Since a quadrilateral is the intersection of two chords, the insertion of a single new chord through a single front advancement does not uniquely determine any quadrilaterals unless the chord

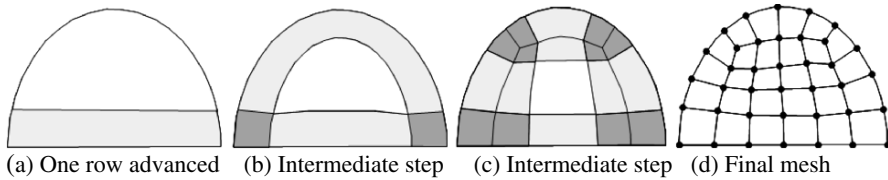


Figure 1. Unconstrained Paving

inserted happens to cross previously inserted chords. In Figure 1b, an additional front is advanced inserting a second chord. At the bottom two corners, this newly inserted row crosses the previously advanced row, thus defining a quadrilateral element at each of the two row crossings.

Figure 1 a, b, and c illustrate that at all times, the unmeshed void (white region) is connected to the boundary either by direct adjacency or through connecting tubes. For example, in Figure 1a, the unmeshed void is connected to one connecting tube on the bottom, and the surface boundary on the top. In Figure 1b, the unmeshed void is connected to two connecting tubes, one on the top and one on the bottom. Likewise in Figure 1c, the unmeshed void is connected to four connecting tubes (i.e top, bottom, left and right). Thus, any of the curves on the unmeshed void can be split into as many mesh edges as needed for resolution of the void. Any splitting of the curves on the unmeshed void can be propagated back to the boundary through the connecting tubes.

Similarly in three dimensions with Unconstrained Plastering, all surfaces of the unmeshed void are connected to the boundary either by direct adjacency or through connecting tubes. As a result, any of the surfaces of the unmeshed void are free to be discretized as required for resolution. Any discretization of the surfaces of the unmeshed void can be propagated back to the boundary through the connecting tubes. This is in contrast to traditional Paving and Plastering where the unmeshed void is completely discretized at all times by either element edges or quadrilateral faces. This discretization proved to be the Achilles heal for traditional Plastering since the unmeshed void is typically discretized with closely spaced randomly oriented quadrilaterals. In general, Unconstrained Paving and Plastering continue advancement of rows and sheets until the unmeshed void can be meshed with Midpoint Subdivision [13].

Unconstrained Paving and Plastering rely heavily upon model topology by removing the constraint of the pre-meshed boundary, advancing unconstrained rows and sheets rather than single elements, and by following strict guidelines which consider global ramifications when local dual operations are performed. Unconstrained Plastering also considers geometric

characteristics of the model by performing proximity and angle checks between nearby fronts, size checks to make sure that front advancements are consistent with desired element sizes, and layer checks to ensure that advancing fronts are boundary-sensitive. In addition, like traditional Plastering, Unconstrained Plastering advances rows in the primal space which provides access to direct geometric properties of the model and previously advanced rows. In contrast, Whisker Weaving operates in the dual space which is part of the reason geometric features are not considered. It is anticipated that through careful combination of both topological and geometric considerations, Unconstrained Plastering will be successful on arbitrary geometry assemblies.

Although Unconstrained Plastering has matured since its initial introduction, there are still several technical hurdles which must be overcome before success can be declared. Section 3.0 introduces incomplete fronts which Unconstrained Paving and Plastering use to handle model concavities. Model concavities appear in everything except trivial primitive models. Section 4.0 describes the processes of merging and seaming which are used to eliminate proximity problems and small angles between adjacent fronts which occur in nearly every model as fronts collide. Section 5.0 shows some example models which have been meshed with the current implementation. Finally, section 6.0 discusses plans for future research and development, followed by conclusions in section 7.0.

3 Model Concavities – Incomplete Fronts

Concavities are a common occurrence in even simple CAD models. A strict geometric definition of a concavity is anywhere on the model where the interior angle at a point is greater than 180 degrees. However, in a hexahedral meshing sense, a concavity is anywhere that has a large enough interior angle that three hexahedra would more accurately model the geometry than only two hexahedra. Submapping technology defines this condition as a “Corner” [14].

Unconstrained Plastering handles concavities through the definition and advancement of “incomplete fronts.” Figure 2a shows a simple solid with one of the surfaces highlighted. Figure 2b shows what the ideal mesh would look like on this model. The left-most curve on the highlighted surface is a concavity in the model since each mesh edge on it has three adjacent hexahedra. Figure 2c shows the sheet of hexes which is directly adjacent to the highlighted surface. Because of the concavity on the left-most curve, this sheet extends out into the interior of the solid.

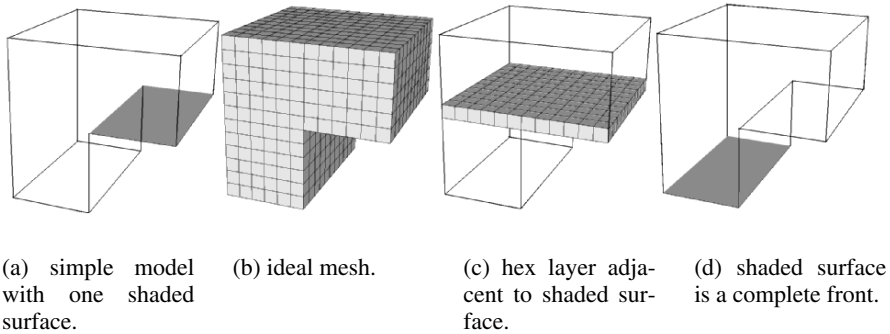


Figure 2. Example of an incomplete front

When Unconstrained Plastering begins, it initializes fronts from each of the boundary CAD surfaces. The highlighted surface in Figure 2a would be initialized as a front to advance. However, because of the concavity, this front is marked as an *incomplete* front. In contrast, the highlighted surface in Figure 2d would be initialized as a *complete* front since all of the curves on it are convex. The advancement of this or any other complete front entirely defines a single hexahedral sheet. This is accomplished because the topology of the model completely defines the path the sheet should take. However, the advancement of an incomplete front can only define the portion of the sheet directly in front of the surface(s) comprising the front. Any further advancement of the front would be arbitrary. Figure 3a, b, & c shows three possible arbitrary advancements of a complete sheet from the incomplete front in this example. All three are valid, and the mesh in Figure 2b demonstrates that the sheet in Figure 3b is eventually used. However, choosing between them is not possible until other adjacent fronts are advanced. The recommended procedure for incomplete fronts is to advance the front to form a partial hex sheet, as shown in Figure 3d, followed by advancement of other complete fronts in the model. Figure 3e shows this simple example after several adjacent fronts are advanced. Notice that the incomplete front was advanced only once. Figure 3f shows that by continuing to advance adjacent *complete* fronts, the *incomplete* front can be completed when adjacent fronts are advanced far enough to guide the incomplete hex sheet to completion.

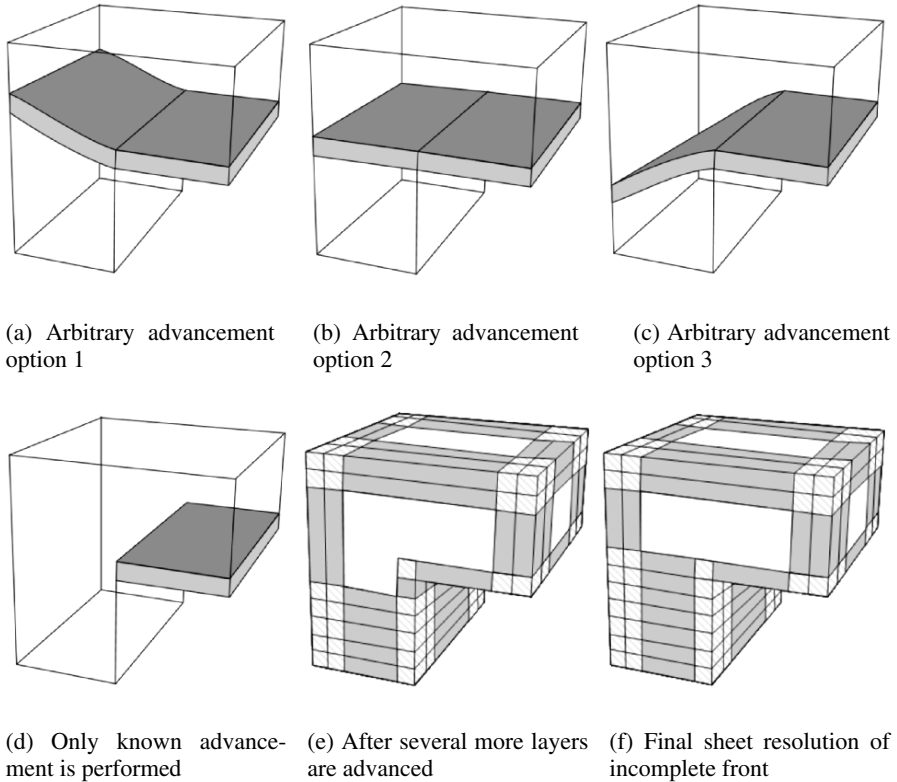


Figure 3. Various options and final resolution of incomplete front

Another option would be to completely refrain from advancing this incomplete front until adjacent fronts have been advanced far enough to “complete” the incomplete front. However, doing so would leave the boundary of the solid exposed to direct collisions from other advancing fronts. When fronts collide, seaming and merging is needed. Seaming and merging operations result in nodes and edges with non-optimal valences, which often results in poor element qualities. By performed seaming and merging directly on the model boundary the risk of creating poor elements directly on the model boundary is increased. Since analysis results are often of greater interest on the boundary, care must be taken to ensure as high an element quality as possible on the boundary. As such, the recommendation is that each incomplete front be advanced once in order to form a protective layer directly adjacent to the model boundary. After a single advancement, the incomplete front can wait until adjacent complete fronts are advanced far enough to complete the incomplete front.

4 Front Collisions

As fronts are advanced in Unconstrained Paving & Plastering, certain operations must be performed. The operations that are performed most often are merging and seaming. Merging is defined as the resolution of small gaps between fronts. Seaming is defined as the resolution of small angles between adjacent fronts. Merging and seaming must be performed iteratively since merging often creates seaming cases; likewise seaming often creates merging cases.

4.1 Merging

During Unconstrained Paving and Plastering, cases requiring merging occur in either the connecting tubes or in the unmeshed void. Figure 4 illustrates the partial meshing of an example surface using Unconstrained Paving. Figure 4c shows a close-up of a connecting tube which is too skinny for an additional advancement. In Figure 4d, the solid dark line represents the front to advance. The dashed dark line represents the desired advanced location of this front, which clearly shows the proximity problem which must be resolved before continuing. Figure 4e shows how the proximity can be resolved by collapsing out the connecting tube, with the solid dark line representing the modified front after merging. Figure 4f shows what the model would look like after the advancement of additional fronts after the merge operation. Essentially, the merge operation inserts a 5-valent node in the quad mesh.

Figure 4c shows the connecting tube which must be collapsed. In this case, proximity problems exist throughout the entire connecting tube. As a result, it makes geometric sense to collapse the entire tube. However, on other geometries, it is possible that the connecting tube might expand in some regions, making proximity only an issue for a portion of the tube. This is illustrated in Figure 5a. However, regardless of the geometric proximity, if part of the tube requires collapsing, then the entire tube must be collapsed in order to keep topological consistency throughout the model. In order to reconcile sizing in cases where only part of the tube has proximity, pillowing [15] can be performed to make the tube sizes more consistent. Figure 5 illustrates that pillowing followed by row smoothing can be performed in the connecting tubes to make the size of the tube consistent so the entire tube can be merged.

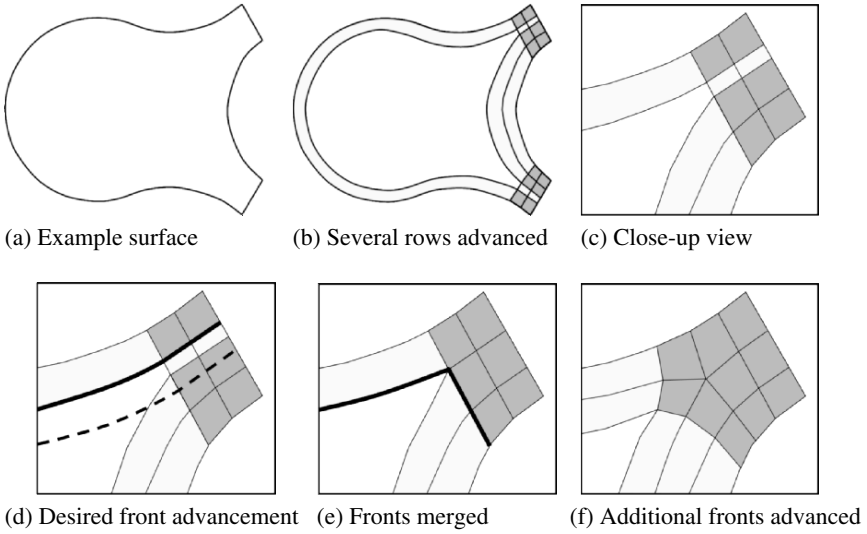


Figure 4. Merging example for connecting tube proximity

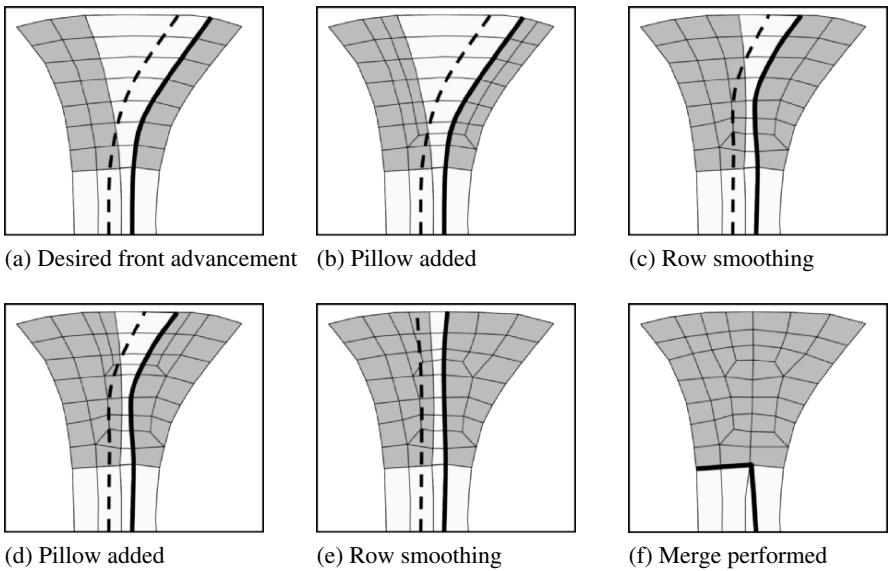
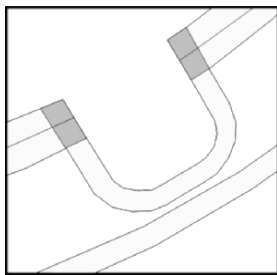
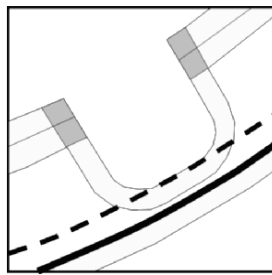


Figure 5. Merging partial tube proximity problems with pillowing

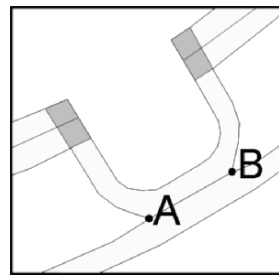
Merge cases can also occur in the unmeshed void as illustrated in Figure 6. The thick dark solid line in Figure 6b is the front to be advanced. The dashed dark line shows the desired advancement which is not possible because of the proximity. Figure 6c shows that the proximity has been merged out. Points A and B are the extremes of the merge and are new topological constraints on the model where nodes must be placed. They will end up being 5-valent nodes in the final mesh. In order to maintain mesh topology consistency, these points must be propagated back to the boundary through the connecting tubes as shown in Figure 6d. The propagation of constraints to the boundary through connecting tubes is called “cutbacks.” Cutbacks split the connecting tube into two or more connecting tubes. The original front that had the proximity problem then needs to be updated as shown in Figure 6e. Additional fronts can then be advanced around the proximity as shown in Figure 6f.



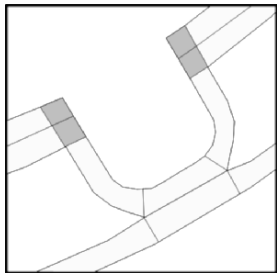
(a) Proximity case in unmeshed void



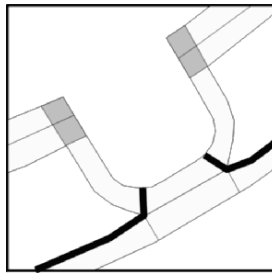
(b) Desired front advancement



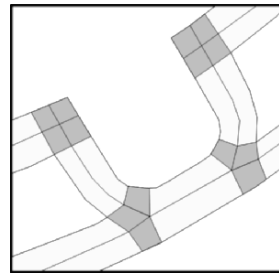
(c) Merged model, A & B are new model constraints



(d) Cutbacks added



(e) New fronts for advancement



(f) Several additional fronts are advanced

Figure 6. Merging example for unmeshed void proximity

Figure 7 illustrates a merge case in the connecting tubes during Unconstrained Plastering. The unmeshed void is drawn in white, the connecting tubes in light gray, the connecting webs in dark gray, and the final elements in black. Proximity in the connecting tube on the right stops additional sheets from advancing from both the top and bottom of the model. The proximity is resolved by merging out the connecting tube and the corresponding connecting webs in Figure 7b. An additional front from the top of the model can now be advanced as illustrated in Figure 7c.

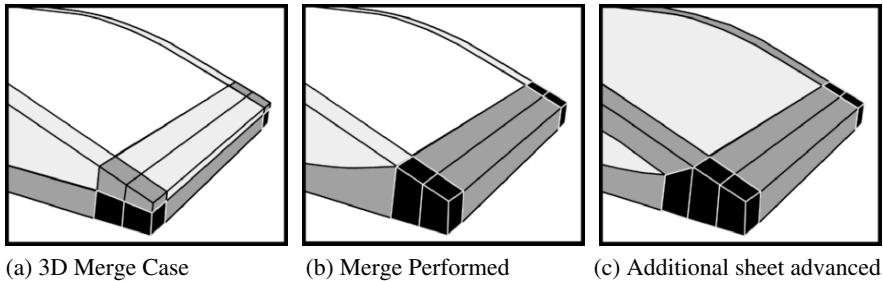


Figure 7. Unconstrained Plastering merge case

4.2 Seaming

During Unconstrained Paving and Plastering, cases requiring seaming occur where two fronts intersect. The need for seaming is based on the angle at which the fronts intersect. Figure 8a shows the model from Figure 4 after the merging was performed. Figure 8b shows a close up showing the intersection between fronts A and B. If the angle of intersection θ , is less than a specified tolerance β , then seaming is required. Seaming is performed, as shown in Figure 8d, by merging the fronts together until the θ increases above $\beta * w$, where w is a weighting factor greater than 1.0. If $w = 1.0$, then the angle will only increase to exactly the seaming tolerance. Subsequent small perturbations nearby could cause the angle to decrease which would cause seaming to be required again. If a larger w is used, such as 1.1, then θ will have increased enough over β that nearby changes will be less likely to drop θ enough to require additional seaming.

Figure 8d illustrates the completion of the seam. Point C is the point where the seaming stops. Point C is a new constraint on the model where a five-valent node will be located. Similar to point A and B in Figure 6c, point C must be propagated back to the boundary with cutbacks. Figure 8e shows the addition of the cutbacks and the modifications to fronts A and B. Figure 8f shows the model after fronts A and B have both been advanced one additional row.

The highlighted region of Figure 8g illustrates that cutbacks in seaming often create a set of connecting tubes which are no longer connected to an unmeshed void. These connecting tubes are like all other connecting tubes in that they still have one degree of freedom remaining for Unconstrained

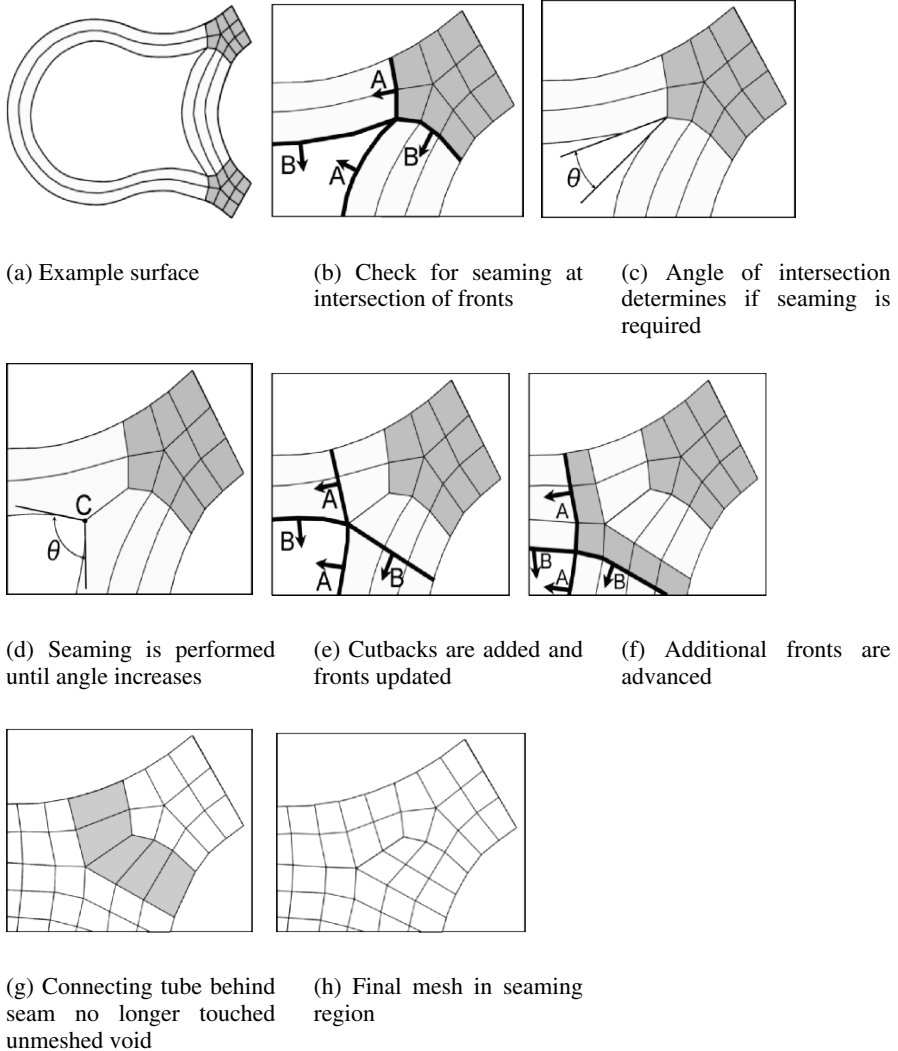
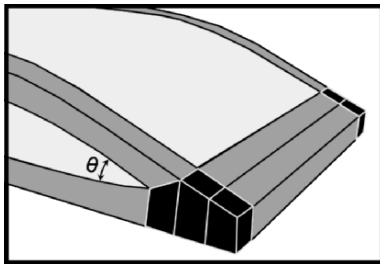


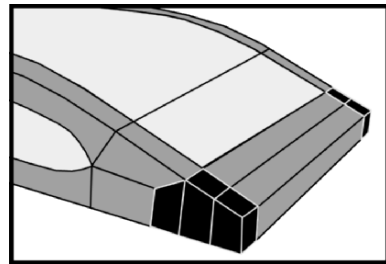
Figure 8. Unconstrained Paving seaming example

Paving and two degrees of freedom for Unconstrained Plastering. For Unconstrained Paving, these connecting tubes can be diced as many times as required to get the proper element resolution as shown in Figure 8h. For Unconstrained Plastering, these connecting tubes which are no longer connected to an unmeshed void represent a partition of the solid which can be meshed with traditional paving and sweeping [16,17,18].

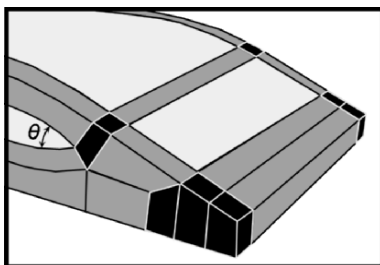
Figure 9 Illustrates a seaming case in Unconstrained Plastering which is a result of the merge operation performed in Figure 7. After the seaming operation is performed, an additional sheet is advanced as illustrated in Figure 9c. This sheet advancement creates an additional seaming case which is seamed as illustrated in Figure 9d. As illustrated in Figure 8g & h, connecting tubes that no longer touch an unmeshed void are often created during seaming. This is also the case in three dimensions with Unconstrained Plastering. The two light gray surfaces on the top of the model in Figure 9d represent two such connecting tubes. These two light gray surfaces can be meshed with quadrilaterals and swept [16,17,18] through the corresponding connecting tube to obtain the final mesh. By definition, the topology of connecting tubes in three dimensions has only a single source and a single target, which simplifies the sweeping process to 1-1 sweeping.



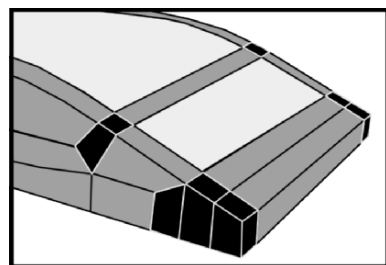
(a) 3D Seam Case



(b) Seam with cutbacks performed



(c) Additional front advanced



(d) Another seaming operation performed

Figure 9. Unconstrained Plastering Seaming example

5 Unconstrained Plastering Examples

Conceptually, Unconstrained Paving and Plastering can handle a wide variety of model complexity. The logic is available to handle concavities, small model angles, collisions between fronts, seaming of adjacent fronts, and assembly models. However, as is often the case, implementation of the logic lags the conceptual progression. At the time of its initial introduction [1], no solids had yet been successfully meshed with Unconstrained Plastering. Since then implementation has progressed to successfully mesh numerous models of simple complexity.

Figure 10 shows a seemingly simple model meshed with Unconstrained Plastering. However, the concavity requires the use of incomplete fronts. In addition, merging in both connecting tubes and in the unmeshed void were required. As expected, the resulting mesh topology is that of a sub-mapped mesh [14]. The minimum scaled Jacobian in this mesh is 0.92 on a scaled from 0.0 to 1.0 where 1.0 is the perfect hexahedral element.

Unconstrained Plastering meshed the model in Figure 11 with a minimum scaled Jacobian of 0.62. The topology of the model is that of a simple brick, however, the top surface has the shape of a rhombus, which results in non-perpendicular angles leading to lower quality elements and some irregular transitioning nodes on the top and bottom surface. The mesh topology of the mesh is that of a swept mesh.

Figure 12 illustrates a model which is also relatively simple. The top surface is a curved nurb. The tapered end caused the merging and seaming cases illustrated in Figure 7 and Figure 9 to arise which resulted in the irregular nodes on the side of the volume. The irregular nodes on the top of the model are introduced by the quad mesher which meshes the top surface of a connecting tube which is then swept as described in section 4.2. The minimum scaled Jacobian in this mesh is 0.577. The mesh topology is neither mapped or swept, but a true unstructured mesh topology. Although the model topology of this model could yield a swept mesh topology, this example illustrates that Unconstrained Plastering is not restricted to such simple mesh topologies. Rather, Unconstrained Plastering is free to insert hexahedral sheets into the solid as required to adequately model the geometric complexity.

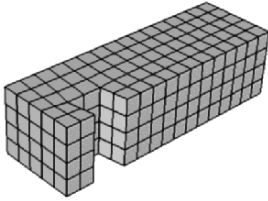


Figure 10. Example model 1

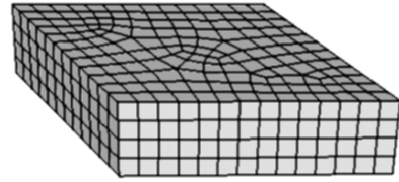


Figure 11. Example model 2

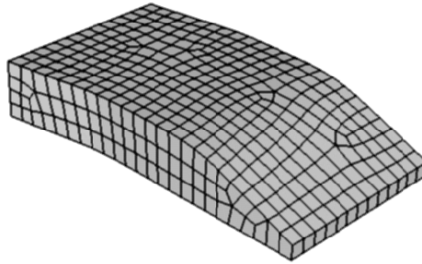


Figure 12. Example model 3

6 Future Research

As mentioned previously, implementation of Unconstrained Plastering lags the conceptual development. As a result, resources are currently allocated to implementation of the currently available logic. Current implementation resources are focused on more complicated concavities, models with very thin sections, boundary topology which intersects at extreme angles, and assembly models. It is anticipated that as implementation progresses, additional cases will be encountered which will require additional logic and operations which are not yet considered.

At the time of publication of [1], implementation on Unconstrained Plastering had begun, but implementation of Unconstrained Paving was not a priority. Priorities and resources have since been modified. As a result, implementation of Unconstrained Paving has now begun. It is anticipated that Unconstrained Paving will behave much better than traditional Paving on surfaces that have skinny regions as illustrated in Figure 13. Figure 13a shows the result from traditional Paving. Since the boundary edges are meshed apriori, the nodes are placed without considering proximity to other curves. As a result, the gray elements illustrate that skewed elements often result in thin sections such as this. Figure 13b shows what a more desirable mesh would be where the nodes opposite the thin section line up. Since Unconstrained Paving is not constrained by an apriori boundary mesh, it is anticipated that the cutback process resulting from the merging in this region will result in a mesh similar to that in Figure 13b.

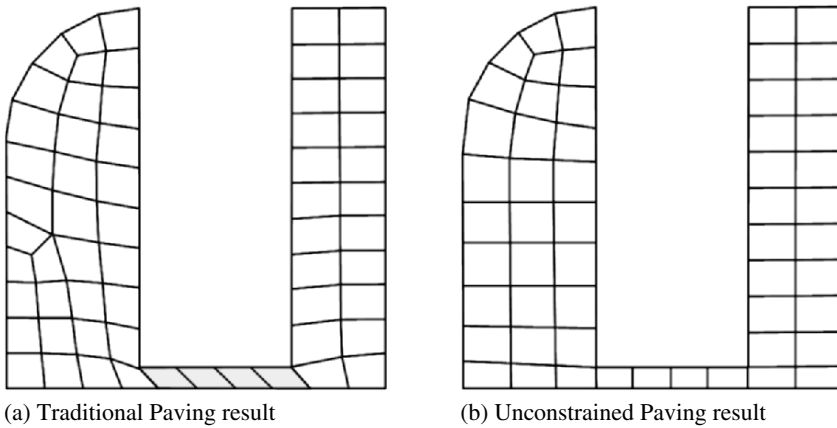


Figure 13. Mesh comparison between traditional Paving and Unconstrained Paving

7 Conclusions

The conceptual understanding of Unconstrained Paving and Plastering has progressed significantly since its initial introduction. Conceptually, many complex models can be handled. Implementation lags the conceptual development, but is the focus of current resources.

Triangular and tetrahedral meshing algorithms are accompanied by long accepted mathematical proofs and theorems [19]. In contrast, Unconstrained Paving and Plastering, like traditional Paving [5], are quite heuristic and currently lack complete mathematical verification. Regardless, traditional Paving is accepted as a robust solution to the quadrilateral meshing problem because it has been implemented dozens of times at both academic and commercial institutions resulting in robust and efficient mesh generation software. Likewise, the current implementation of Unconstrained Plastering indicates that once a complete implementation is in place, Unconstrained Paving and Plastering may also provide efficient and robust mesh generation tools. Future advances in mathematics and mesh generation may provide the mathematical backing they currently lack.

References

1. M.L. Staten, S.J. Owen, T.D. Blacker, "Unconstrained Paving & Plastering: A New Idea for All Hexahedral Mesh Generation," *Proc. 14th Int. Meshing Roundtable*, 399-416, 2005.
2. A.O. Cifuentes, A. Kalbag, "A Performance Study of Tetrahedral and Hexahedral Elements in 3-D Finite Element Structural Analysis," *Finite Elements in Analysis and Design*, Vol. 12, pp. 313-318, 1992.
3. S.E. Benzley, E. Perry, K. Merkley, B. Clark, "A Comparison of All-Hexahedral and All-Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis," *Proc. 4th Int. Meshing Roundtable*, 179-191, 1995.
4. ABAQUS Analysis User's Manual, Version 6.5, Section 14.1.1, Hibbit, Karlsson & Sorensen: USA, 2005.
5. T.D. Blacker, M.B. Stephenson, "Paving: A New Approach to Automated Quadrilateral Mesh Generation," *Int. Journal for Numerical Methods in Eng.*, 32, 811-847, 1991.
6. S.A. Canann, "Plastering: A New Approach to Automated 3-D Hexahedral Mesh Generation," *American Institute of Aeronautics and Astronautics*, 1992.
7. J. Hipp, R. Lober, "Plastering: All-Hexahedral Mesh Generation Through Connectivity Resolution," *Proc. 3rd Int. Meshing Roundtable*, 1994.
8. T. D. Blacker, R. J. Meyers, "Seams and Wedges in Plastering: A 3D Hexahedral Mesh Generation Algorithm," *Eng. With Computers*, 2, 83-93, 1993.
9. T. J. Tautges, T. Blacker, S. Mitchell, "The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes," *Int. Journal for Numerical Methods in Eng.*, 39, 3327-3349, 1996.
10. N. T. Folwell, S. A. Mitchell, "Reliable Whisker Weaving via Curve Contraction," *Proc. 7th Int. Meshing Roundtable*, 365-378, 1998.
11. P. Murdoch, S. Benzley, "Spatial Twist Continuum," *Proc. 4th Int. Meshing Roundtable*, 243-251, 1995.
12. J. R. Tristano, S. J. Owen, S. A. Canann, "Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition," *Proc. 7th Int. Meshing Roundtable*, 429-445, 1998.
13. T. S. Li, R. M. McKeag, C. G. Armstrong, "Hexahedral Meshing Using Midpoint Subdivision and Integer Programming," *Computer Methods in Applied Mechanics and Engineering*, Vol. 124, Issue 1-2, 171-193, 1995.
14. D. White, L. Mingwu, S. Benzley, "Automated Hexahedral Mesh Generation by Virtual Decomposition," *Proc. 4th Int. Meshing Roundtable*, 165-176, 1995.
15. S. A. Mitchell, T. J. Tautges, "Pillowing Doublets: Refining a Mesh to Ensure That Faces Share at Most One Edge," *Proc. 4th Int. Meshing Roundtable*, 231-240, 1995.
16. M. L. Staten, S. A. Canann, S. J. Owen, "BMSweep: Locating Interior Nodes During Sweeping," *Proc. 7th Int. Meshing Roundtable*, 7-18, 1998.
17. T. D. Blacker, "The Cooper Tool," *Proc. 5th Int. Meshing Roundtable*, 13-29, 1996.
18. Mingwu, Lai, "Automatic Hexahedral Mesh Generation by Generalized Multiple Source to Multiple Target Sweep," Ph.D. Dissertation, Brigham Young University, Provo, Utah, USA, 1998.
19. P.L. George, H. Borouchaki, "Delaunay Triangulations and Meshing: Application to Finite Elements," ©Editions HERMES, Paris, 1998.

An Automatic and General Least-Squares Projection Procedure for Sweep Meshing

Xevi Roca and Josep Sarrate

Laboratori de Càlcul Numèric, ETSE de Camins, Canals i Ports de Barcelona,
Universitat Politècnica de Catalunya, Edifici C2, Jordi Girona 1-3, E-08034
Barcelona, Spain

`xevi.roca@upc.edu jose.sarrate@upc.edu` **

Summary. In this paper we present a new node projection scheme to generate hexahedral meshes in sweeping geometries. It is based on a least-squares approximation of an affine mapping. In the last decade several functionals have been defined to perform this least-square approximation. However, all of them present several shortcomings in preserving the shape of the inner part of the projected meshes, i.e. the offset data, for simple and usual geometrical configurations. To overcome these drawbacks we propose to minimize a more general functional that depends on two vector parameters. Moreover, we detail a procedure that automatically selects these parameters in such a way that offset data is maintained in the inner part of projected meshes.

Key words: Finite element method; mesh generation; hexahedral elements; sweep; node projection; affine mapping.

1 Introduction

Several fast and robust algorithms have been developed to generate unstructured tetrahedral meshes [1, 2]. However, fully automatic unstructured hexahedral mesh generation algorithms are still not available. Therefore, special attention has been focused on existing algorithms that decompose the entire geometry into several simpler pieces that can be considered as union of one-to-one extrusion volumes. Sweeping is one of the most robust and efficient algorithms to mesh these simpler volumes with hexahedral elements. Several algorithms have been devised to generate hexahedral meshes by projecting the cap surfaces along the sweep path [3, 4, 5, 6]. In all of them the crucial step is the placement of the inner nodes. From the computational point

**This work was partially sponsored by the *Ministerio de Educación y Ciencia* under grants DPI2004-03000 and CGL2004-06171-C03-01/CLI, and *E.T.S. d'Enginyers de Camins, Canals i Ports de Barcelona*

of view, sweep methods based on a least-squares approximation of an affine mapping are the fastest alternative to compute these projections [7]. Several functionals have been introduced to perform the least-squares approximation, see section 2. In spite of their computational efficiency (both in terms of CPU time and memory), these methods present several drawbacks. For instance, the minimization of these functionals may lead to a set of normal equations with a singular system matrix for very usual geometrical configurations. In addition, the obtained mesh may present several undesired features such as flattening and skewness, see [9] for details.

In order to overcome these shortcomings, in reference [9] we introduced a new functional that depends on two vector parameters that can be selected by the user. However, only a feasible selection of these parameters, based on our experience, was provided. In this paper we first prove the relationship between the optimal solution of the classical functional and the optimal solution of the new functional proposed in [9]. In addition, we propose a definition of a measure of the normal vector to a given loop of nodes that we denote by *pseudo-normal*. Based on the previous relationship and the definition of the pseudo-normal, we prove and detail a new algorithm that automatically selects the functional parameters. These parameters are selected in order to preserve the shape of the inner part of projected meshes, i.e. *offset data*. It is important to point out that the geometrical cases that lead to a set of normal equations with a singular system matrix are identified from the singular value decomposition (SVD) of the optimal solution of the classical functional. Moreover, to increase the computational efficiency of the proposed algorithm, the minimization of the new functional adequately reuses the optimal solution of the classical functional. Finally, we present two simple examples that show the robustness and the reliability of the proposed algorithm.

2 Problem Statement and Functional Definitions

Let $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^n$ be a set of source points, and $Y = \{\mathbf{y}^i\}_{i=1,\dots,m} \subset \mathbb{R}^n$ a set of target points with $m \geq n$. In a sweep application $\{\mathbf{x}^i\}_{i=1,\dots,m}$ are the nodes that belong to the boundary of the projected layer (where the initial layer is the source surface mesh). Similarly, $\{\mathbf{y}^i\}_{i=1,\dots,m}$ are the nodes that belong to the boundary of the target layer. Our goal is to find a mapping $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$ such that

$$\mathbf{y}^i = \phi(\mathbf{x}^i), \quad i = 1, \dots, m. \quad (1)$$

We approximate ϕ by an affine mapping φ from \mathbb{R}^n to \mathbb{R}^n ,

$$\varphi(\mathbf{x}) = \mathbf{A}(\mathbf{x} - \mathbf{c}^X) + \mathbf{c}^Y, \quad (2)$$

where

$$\mathbf{c}^X := \frac{1}{m} \sum_{i=1}^m \mathbf{x}^i \quad \text{and} \quad \mathbf{c}^Y := \frac{1}{m} \sum_{i=1}^m \mathbf{y}^i$$

are the the geometrical centers of the sets X and Y , respectively. The affine mapping φ is computed by minimizing the functional

$$F(\mathbf{A}) := \sum_{i=1}^m \|\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)\|^2 = \sum_{i=1}^m \|\bar{\mathbf{y}}^i - \mathbf{A}\bar{\mathbf{x}}^i\|^2, \tag{3}$$

where $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{c}^X$ and $\bar{\mathbf{y}} = \mathbf{y} - \mathbf{c}^Y$, see details in [5, 9]. The minimization of functional F is equivalent to imposing the following m constraints

$$\mathbf{A}(\mathbf{x}^i - \mathbf{c}^X) = \mathbf{y}^i - \mathbf{c}^Y, \quad i = 1, \dots, m, \tag{4}$$

being the unknowns the coefficients of the $n \times n$ matrix

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix}.$$

These constraints can be expressed in matrix form as

$$\mathbf{A}\bar{\mathbf{X}} = \bar{\mathbf{Y}}, \tag{5}$$

where

$$\bar{\mathbf{X}} := \begin{pmatrix} x_1^1 - c_1^X & \dots & x_1^m - c_1^X \\ \vdots & & \vdots \\ x_n^1 - c_n^X & \dots & x_n^m - c_n^X \end{pmatrix} \quad \text{and} \quad \bar{\mathbf{Y}} := \begin{pmatrix} y_1^1 - c_1^Y & \dots & y_1^m - c_1^Y \\ \vdots & & \vdots \\ y_n^1 - c_n^Y & \dots & y_n^m - c_n^Y \end{pmatrix},$$

However, minimization of functional (3) generates flattened layers under the following two conditions:

- If the set of source points, X , determines a plane in 3D geometries (for instance a source surface mesh with planar boundary), then the minimization of functional (3) leads to a set of normal equations with singular system matrix, see [9]. In practice, singular value decomposition is used to solve the set of normal equations. In this case the inner part of the projected mesh will be planar. Hence, the offset data of the source surface mesh will be lost.
- If a given mesh is projected to an inner layer with a planar boundary by minimizing (3), then the projected mesh will always be planar, see [9].

It is important to point out that these geometrical configurations are extremely usual in CAD models. We will use the term *hyperplanar* to denote a linear variety of dimension $n - 1$ (a plane for $n = 3$ and a straight line for $n = 2$). In particular, given a hyperplanar set of points, X , we define the *homogeneous hyperplane of X* as the subspace of vectors

$$\mathbb{H} = \{\mathbf{v} \in \mathbb{R}^n \mid \langle \mathbf{n}^X, \mathbf{v} \rangle = 0\}, \tag{6}$$

where $\mathbf{n}^X \in \mathbb{R}^n$ is a unitary normal vector to X .

To solve the previous drawbacks, Knupp introduced another change of coordinates: $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X$ and $\bar{\mathbf{y}} = \mathbf{y} - \mathbf{c}^X$, see [5, 9] for details. Moreover, using these new coordinates the following functional was also defined in [5]

$$G(\mathbf{A}) := \sum_{i=1}^m \|\mathbf{y}^i - \mathbf{c}^X - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X)\|^2 = \sum_{i=1}^m \|\bar{\mathbf{y}}^i - \mathbf{A}\bar{\mathbf{x}}^i\|^2. \tag{7}$$

Therefore, we are looking for a linear mapping \mathbf{A} such that it approximately transforms, in the least-squares sense, $\bar{\mathbf{X}} = \{\bar{\mathbf{x}}^i\}_{i=1,\dots,m}$ to $\bar{\mathbf{Y}} = \{\bar{\mathbf{y}}^i\}_{i=1,\dots,m}$.

However, functional (7) also presents two important shortcomings:

- If the set of source points, X , is hyperplanar and $\mathbf{c}^Y - \mathbf{c}^X \in \mathbb{H}$, then the minimization of functional G leads to a set of normal equations with singular system matrix, see [9].
- If a non-planar surface mesh with planar boundary is projected to an inner layer which is non-parallel to the boundary of the source surface, then the projected nodes do not preserve the shape of the original surface mesh and a skewness effect is introduced, see [9].

In order to overcome the drawbacks arising from the minimization of functionals F and G , we introduced the following functional, see [9]

$$H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) := \sum_{i=1}^m \|\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)\|^2 + \|\mathbf{u}^Y - \mathbf{A}\mathbf{u}^X\|^2, \tag{8}$$

where \mathbf{u}^X and \mathbf{u}^Y belong to \mathbb{R}^n . It is important to point out that vectors \mathbf{u}^X and \mathbf{u}^Y in (8) can be properly selected in order to obtain several desired properties of functional H .

It has been proved that if the set of source points is hyperplanar it is always possible to select a vector \mathbf{u}^X such that the minimization of H leads to a set of normal equations with a full rank matrix, see [9]. However, given any arbitrary geometry no algorithm was proposed to properly define vectors \mathbf{u}^X and \mathbf{u}^Y . The main goal of the present paper is to explicitly state how to select vectors \mathbf{u}^X and \mathbf{u}^Y in order to define an automatic and robust algorithm to sweep meshes in a one-to-one volume.

3 Analysis of Functional H

In this section we present new properties of functional H that are of major importance to deduce the general node projection algorithm. First, we prove four lemmas that will allow us to relate the solutions of the minimization of functionals F and H .

Lemma 1. *If X is a hyperplanar set of points and $\mathbf{u}^X \notin \mathbb{H}$, then $\mathbb{R}^n = \text{span}(\mathbf{u}^X) \oplus \mathbb{H}$.*

Proof. In this case, the homogeneous hyperplane defined by X is a subspace of \mathbb{R}^n with dimension equal to $n-1$. Since $\mathbf{u}^X \notin \mathbb{H}$ we have that $\text{span}(\mathbf{u}^X) \cap \mathbb{H} = \{\mathbf{0}\}$. Thus, $\mathbb{R}^n = \text{span}(\mathbf{u}^X) \oplus \mathbb{H}$. \square

Lemma 2. *Let X be a hyperplanar set of points. Assume that $\mathbf{u}^X \notin \mathbb{H}$, $\mathbf{u}^Y \in \mathbb{R}^n$, and $\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)$ are given. Then, there exists a mapping $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y] : \mathbb{R}^n \rightarrow \mathbb{R}^n$ such that:*

- (i) $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y] \in \mathcal{L}(\mathbb{R}^n)$
- (ii) $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{u}^X) = \mathbf{u}^Y$
- (iii) $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{v}_{\mathbb{H}}) = \mathbf{A}\mathbf{v}_{\mathbb{H}}, \quad \forall \mathbf{v}_{\mathbb{H}} \in \mathbb{H}$.

Proof. Given $\mathbf{v} \in \mathbb{R}^n$, Lemma 1 states that \mathbb{R}^n can be represented as the direct sum of the subspaces $\text{span}(\mathbf{u}^X)$ and \mathbb{H} . Therefore, for every $\mathbf{v} \in \mathbb{R}^n$ there exist $\mathbf{v}_{\mathbb{H}} \in \mathbb{H}$ and $\lambda \in \mathbb{R}$ such that

$$\mathbf{v} = \mathbf{v}_{\mathbb{H}} + \lambda \mathbf{u}^X. \tag{9}$$

Hence, we define the image of $\mathbf{v} \in \mathbb{R}^n$ by $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]$ as

$$\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{v}) := \mathbf{A}\mathbf{v}_{\mathbb{H}} + \lambda \mathbf{u}^Y. \tag{10}$$

It is straightforward to prove that $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]$ defined in such a way is linear, and that it verifies properties (ii) and (iii). \square

To illustrate a practical application of Lemma 2 in a three-dimensional problem, we consider a set of points X located on a plane \mathbb{H} , see figure 1. Under these conditions, the numerical solution obtained by minimizing functional F maps all vectors that do not belong to the plane \mathbb{H} to the image of this plane by \mathbf{A} , i.e. the flattening effect. In this case, Lemma 2 ensures that by using the linear mapping Θ we will be able take into account the offset data by mapping parallel vectors to \mathbf{u}^X (since vector $\mathbf{u}^X \notin \mathbb{H}$) to the desired direction, provided by vector \mathbf{u}^Y . In addition, we will preserve the behavior of the linear mapping \mathbf{A} over \mathbb{H} . Specifically, any vector that belongs to the plane \mathbb{H} will be mapped according to the linear mapping \mathbf{A} . In practice, we will determine this matrix \mathbf{A} by minimizing functional F , as we will see in Proposition 1.

Lemma 3. *Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. Then,*

$$F(\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]) = F(\mathbf{A}).$$

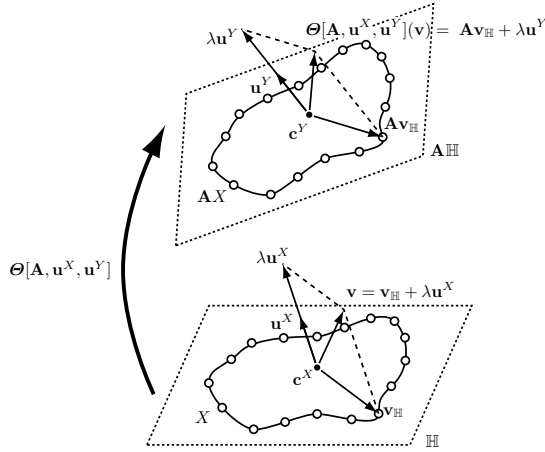


Fig. 1. Transformation of a given vector \mathbf{v} by mapping Θ when X is a planar set.

Proof. Since X is hyperplanar, $\mathbf{x}^i - \mathbf{c}^X \in \mathbb{H}$, for $i = 1, \dots, m$. Therefore, by the third property of Lemma 2, $\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{x}^i - \mathbf{c}^X) = \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)$, for $i = 1, \dots, m$. Finally, according to the definition of the functional F

$$\begin{aligned} F(\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]) &= \sum_{i=1}^m \|\mathbf{y}^i - \mathbf{c}^Y - \Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y](\mathbf{x}^i - \mathbf{c}^X)\|^2 \\ &= \sum_{i=1}^m \|\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)\|^2 = F(\mathbf{A}). \end{aligned}$$

□

Lemma 4. Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. Then,

$$H(\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y]; \mathbf{u}^X, \mathbf{u}^Y) = F(\Theta[\mathbf{A}, \mathbf{u}^X, \mathbf{u}^Y])$$

Proof. This result follows from the definitions of functionals F and H , and Lemma 2. □

Proposition 1. Let X be a hyperplanar set of points, and assume that $\mathbf{u}^X \notin \mathbb{H}$ and $\mathbf{u}^Y \in \mathbb{R}^n$. If $\mathbf{A}^F \in \mathcal{L}(\mathbb{R}^n)$ and $\mathbf{A}^H \in \mathcal{L}(\mathbb{R}^n)$ are such that

$$\begin{aligned} F(\mathbf{A}^F) &= \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} F(\mathbf{A}), \\ H(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) &= \min_{\mathbf{A} \in \mathcal{L}(\mathbb{R}^n)} H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y), \end{aligned}$$

then: $\Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y] = \mathbf{A}^H$.

Proof. We have already proved, see details in [9], that the minimization of functional H is equivalent to solving n uncoupled overdetermined linear systems. Thus, the minimization of functional H has one and only one solution. In addition, we define $R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) := \|\mathbf{u}^Y - \mathbf{A}\mathbf{u}^X\|^2$. Hence,

$$H(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) = F(\mathbf{A}) + R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y).$$

We consider the following sequence of equalities and inequalities:

$$\begin{aligned} F(\mathbf{A}^F) &\leq F(\mathbf{A}^H) && \text{since } \mathbf{A}^F \text{ minimizes } F \\ &\leq F(\mathbf{A}^H) + R(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) && \text{since } R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) \geq 0 \text{ for every } \mathbf{A} \\ &= H(\mathbf{A}^H; \mathbf{u}^X, \mathbf{u}^Y) && \text{by definition of } R(\mathbf{A}; \mathbf{u}^X, \mathbf{u}^Y) \text{ and } H \\ &\leq H(\Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]; \mathbf{u}^X, \mathbf{u}^Y) && \text{since } \mathbf{A}^H \text{ minimizes } H \\ &= F(\Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]) && \text{by Lemma 4} \\ &= F(\mathbf{A}^F). && \text{by Lemma 3.} \end{aligned}$$

Note that the first and the last terms are the same. Thus, all the inequalities are in fact equalities. From the previous sequence of equalities we prove that $\Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$ and \mathbf{A}^H minimize the functional H . Since the minimization of H has a unique solution we have that $\Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y] = \mathbf{A}^H$, and the proposition holds. \square

To summarize, based on the four previous lemmas, Proposition 1 states a strong relationship between the optimal solutions of functional F and H , namely \mathbf{A}^F and \mathbf{A}^H . This relationship is the keystone for the general algorithm to minimize functional H . It states that when X is a hyperplanar set of points we can obtain the unique solution of the minimization of H by means of one of the optimal solutions of the minimization of F . Specifically, \mathbf{A}^H can be obtained as the linear transformation that maps \mathbf{u}^X to \mathbf{u}^Y , and any vector $\mathbf{v}_{\mathbb{H}} \in \mathbb{H}$ to $\mathbf{A}\mathbf{v}_{\mathbb{H}}$, see figure 1 for a 3D interpretation. Hence, the two remaining tasks are: 1. to automatically select the two vector parameters \mathbf{u}^X and \mathbf{u}^Y according to the given geometry; and 2. to determine how to compute the linear transformation Θ , see section 5.

4 Preserving Offset Data

In this section we introduce several definitions and results in order to formalize some desirable properties of node projection algorithms. The key issue is the definition of a *measure* of the *normal vector* to a given loop of nodes. Recall that in projection algorithms the inner layers are described by a loop of nodes. That is, there is not an underlying surface carrying any additional information. Moreover, in a wide range of applications the loops of nodes are not planar. Therefore the normal vector to this kind of loops is not defined. However,

given a loop of nodes we will define a pseudo-normal vector and we will relate it to the preservation of the shape of the inner part of the projected mesh, the offset data.

Definition 1 (Loop). *Given a set of points $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$, a loop is the closed poly-line constructed by joining \mathbf{x}^i with \mathbf{x}^{i+1} for $i = 1, \dots, m$. We consider that $\mathbf{x}^{m+1} \equiv \mathbf{x}^1$.*

In several applications it is necessary to sweep a non-simple connected surface along the extrusion path. These surfaces are defined by one outer boundary and as many inner boundaries as holes they have. Therefore, we need to consider sets of points composed by several loops. Specifically, one counter-clockwise oriented loop corresponding to the outer boundary, and several clockwise oriented loops corresponding to the inner holes.

Definition 2 (Multi-loop). *A set of points $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ is a multi-loop if it is organized in p loops X_1, \dots, X_p .*

Definition 3 (Pseudo-area). *Given a vector $\mathbf{c} \in \mathbb{R}^3$, the pseudo-area of a loop $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ is*

$$\mathbf{a} := \sum_{i=1}^m (\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}).$$

The pseudo-area of a multi-loop $X = X_1 \cup \dots \cup X_p$ organized in p loops is

$$\mathbf{a} := \mathbf{a}_1 + \dots + \mathbf{a}_p,$$

where $\mathbf{a}_1, \dots, \mathbf{a}_p$ are the pseudo-areas of loops X_1, \dots, X_p , respectively.

Note that $\|(\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c})\|$ is the double of the area of the triangle $\mathbf{x}^i \mathbf{x}^{i+1} \mathbf{c}$, see figure 2. Moreover, if X is a planar multi-loop, then the pseudo-area \mathbf{a} is equal to the area enclosed by X .

In order to prove that pseudo-area is well defined, the next proposition proves that the pseudo-area vector does not depend on the selected $\mathbf{c} \in \mathbb{R}^3$. Moreover, it is invariant under translations, and its norm is also invariant under orthogonal transformations.

Proposition 2 (Invariance of pseudo-area). *Let $X = \{\mathbf{x}^i\}_{i=1,\dots,m} \subset \mathbb{R}^3$ be a set of points. The pseudo-area vector verifies:*

(i) *Given $\mathbf{c} \in \mathbb{R}^3$ then*

$$\mathbf{a} = \sum_{i=1}^m (\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}) = \sum_{i=1}^m \mathbf{x}^i \times \mathbf{x}^{i+1}.$$

(ii) *Given $\mathbf{t} \in \mathbb{R}^3$ the pseudo-area of X is equal to the pseudo-area of $X + \mathbf{t} = \{\mathbf{x}^i + \mathbf{t}\}_{i=1,\dots,m}$.*

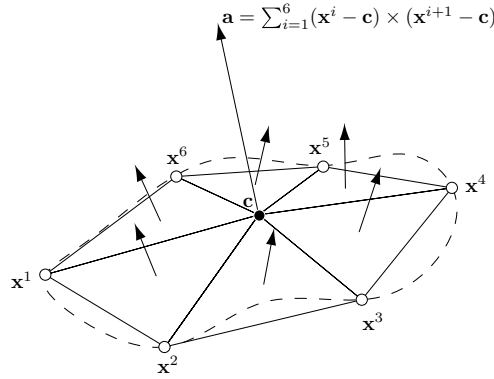


Fig. 2. Geometrical interpretation of the pseudo-area vector.

(iii) Given an orthogonal transformation \mathbf{N} , then the pseudo-area of $\mathbf{N}X = \{\mathbf{N}\mathbf{x}^i\}_{i=1,\dots,m}$ is $\mathbf{N}\mathbf{a}$, where \mathbf{a} is the pseudo-area of X .

Proof. Given $\mathbf{c} \in \mathbb{R}^3$, and taking into account that X is a loop, then

$$\begin{aligned} \mathbf{a} &= \sum_{i=1}^m (\mathbf{x}^i - \mathbf{c}) \times (\mathbf{x}^{i+1} - \mathbf{c}) \\ &= \sum_{i=1}^m \mathbf{x}^i \times \mathbf{x}^{i+1} + \sum_{i=1}^m \mathbf{c} \times (\mathbf{x}^i - \mathbf{x}^{i+1}) + \sum_{i=1}^m \mathbf{c} \times \mathbf{c} \\ &= \sum_{i=1}^m \mathbf{x}^i \times \mathbf{x}^{i+1} + \sum_{i=1}^m \mathbf{c} \times (\mathbf{x}^i - \mathbf{x}^{i+1}) = \sum_{i=1}^m \mathbf{x}^i \times \mathbf{x}^{i+1}. \end{aligned}$$

Given $\mathbf{t} \in \mathbb{R}^3$, the second property is a direct consequence of property (i) applied to $\mathbf{c} = -\mathbf{t}$.

By property (i) and taking into account that \mathbf{N} is orthogonal we have that

$$\mathbf{a}_{\mathbf{N}X} = \sum_{i=1}^m \mathbf{N}\mathbf{x}^i \times \mathbf{N}\mathbf{x}^{i+1} = \sum_{i=1}^m \mathbf{N}(\mathbf{x}^i \times \mathbf{x}^{i+1}) = \mathbf{N}\mathbf{a}.$$

□

At this point, given a loop X , we have proved that the norm of the pseudo-area vector is a geometrical invariant associated to the loop. Furthermore, it only depends on the ordering and the relative geometrical location of the points.

Proposition 3 (Projected area). *If a multi-loop X is projected on an orthogonal plane to its pseudo-area vector \mathbf{a} , then the obtained polygon has area equal to $\|\mathbf{a}\|$.*

Proof. By definition of pseudo-area of a multi-loop, it suffices to prove the result for a single loop. Given a loop X , the projection of the points $\{\mathbf{x}^i\}_{i=1,\dots,m}$ on the orthogonal plane to \mathbf{a} are the points $\mathbf{x}_{\mathbf{a}\perp}^i := \mathbf{x}^i - \mathbf{x}_{\mathbf{a}}$, where

$$\mathbf{x}_{\mathbf{a}}^i := \frac{\langle \mathbf{x}^i, \mathbf{a} \rangle}{\langle \mathbf{a}, \mathbf{a} \rangle} \mathbf{a}.$$

Hence, we have that $\mathbf{x}^i = \mathbf{x}_{\mathbf{a}}^i + \mathbf{x}_{\mathbf{a}\perp}^i$ for $i = 1, \dots, m$. The pseudo-area of X is

$$\begin{aligned} \mathbf{a} &= \sum_{i=1}^m \mathbf{x}^i \times \mathbf{x}^{i+1} = \sum_{i=1}^m (\mathbf{x}_{\mathbf{a}}^i + \mathbf{x}_{\mathbf{a}\perp}^i) \times (\mathbf{x}_{\mathbf{a}}^{i+1} + \mathbf{x}_{\mathbf{a}\perp}^{i+1}) \\ &= \sum_{i=1}^m \mathbf{x}_{\mathbf{a}}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1} + \mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}}^{i+1} + \mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1} \end{aligned}$$

Note that $\mathbf{x}_{\mathbf{a}}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1}$ and $\mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}}^{i+1}$ are orthogonal to \mathbf{a} . Thus, its sum is also orthogonal. Furthermore, $\mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1}$ is parallel to \mathbf{a} . Therefore $\mathbf{x}_{\mathbf{a}}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1} + \mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}}^{i+1}$ cannot contribute to the parallel component to \mathbf{a} and it has to be $\mathbf{0}$. Hence, we have that

$$\mathbf{a} = \sum_{i=1}^m \mathbf{x}_{\mathbf{a}\perp}^i \times \mathbf{x}_{\mathbf{a}\perp}^{i+1},$$

which is the area of the polygon obtained from the projection of the loop X on the orthogonal plane to \mathbf{a} . □

Hence, the view of the loop from the direction of the pseudo-area has an area equal to the norm of the pseudo-area of X . Therefore, we can interpret the direction of the pseudo-area as a *normal* vector to the loop.

Definition 4 (Pseudo-normal). *The pseudo-normal of a multi-loop X is the unitary vector*

$$\mathbf{n}_{pseudo}^X := \mathbf{a} / \|\mathbf{a}\|,$$

where \mathbf{a} is the pseudo-area of X .

Note that if X is a planar loop, the pseudo-normal \mathbf{n}_{pseudo}^X is equal to the unitary normal \mathbf{n}^X to X .

The pseudo-normal provides a kind of *normal* when there is no underlying surface, only the loop of points. All the information along the direction of the pseudo-normal is understood as *offset data*. We claim that a good node projection procedure has to preserve data along the pseudo-normal. In particular, the flattening and skewness effects previously explained are due to a deficient preservation of offset data. In addition, we describe two new undesired effects related to unsatisfactory preservation of offset data:

- **Offset scaling.** Offset data of projected meshes is scaled along the sweep direction. This effect appears when we project from a non-planar loop to

another non-planar loop of different thickness. It is related to the minimization of functional F , and to an incorrect election of \mathbf{u}^X and \mathbf{u}^Y when minimizing functional H , see first example in section 6.

- **Flipping.** Offset data is projected inversely to the expected orientation. It appears when a loop is curved towards one direction and it is projected to another loop that is curved on the opposite direction. It is related to the minimization of functional F , and can also appear due to incorrect elections of \mathbf{u}^X and \mathbf{u}^Y when minimizing functional H , see second example in section 6.

Summarizing, in order to avoid flattening, skewness, offset scaling, and flipping effects we have to obtain affine mappings that preserve the length, direction and orientation of offset data.

5 Algorithm Implementation

In this section we detail the algorithm that we have developed in order to properly select the parameters \mathbf{u}^X and \mathbf{u}^Y and obtain the affine projection. The basic idea is that we can efficiently use the minimization of functional F to minimize H . The key issue is to realize that $\mathbf{A}^H = \Theta[\mathbf{A}^F, \mathbf{u}^X, \mathbf{u}^Y]$ when X is hyperplanar, see Proposition 1. Therefore, the optimal solution of functional H can be computed from the optimal solution of functional F if a proper criterion to select the vectors \mathbf{u}^X and \mathbf{u}^Y is defined.

The general algorithm, for hyperplanar and non-hyperplanar set of points X , is decomposed in two steps. First, the optimal solution \mathbf{A}^F and its singular value decomposition are computed. Second, a criterion to select the vectors \mathbf{u}^X and \mathbf{u}^Y is defined. In addition, if the set of points X and/or Y are hyperplanar (a planar source and/or target surfaces in 3D applications) a geometrical interpretation of the chosen vectors \mathbf{u}^X and \mathbf{u}^Y is also presented.

5.1 The Optimal Solution of Functional F and its Singular Value Decomposition

In order to minimize functional F we compute the minimum norm solution of Equation (5). To this end, we use the singular value decomposition of the system matrix

$$\overline{\mathbf{X}}^T = \overline{\mathbf{U}} \overline{\mathbf{W}} \overline{\mathbf{V}}^T, \tag{11}$$

where $\overline{\mathbf{U}}$ is an $m \times n$ matrix with orthogonal columns, $\overline{\mathbf{W}}$ is a $n \times n$ diagonal matrix with positive or zero elements (the singular values)

$$\overline{\mathbf{W}} := \begin{pmatrix} \overline{w}_1 & & \\ & \ddots & \\ & & \overline{w}_n \end{pmatrix},$$

such that $\bar{w}_1 \geq \bar{w}_2 \geq \dots \geq \bar{w}_{n-1} \geq \bar{w}_n$, and $\bar{\mathbf{V}}$ is an $n \times n$ orthogonal matrix. We denote by $\bar{\mathbf{v}}_i \in \mathbb{R}^n$, for $i = 1, \dots, n$, the columns of matrix $\bar{\mathbf{V}}$.

Taking into account this decomposition, we compute the minimum norm solution, \mathbf{A}^F , as

$$\mathbf{A}^F = \bar{\mathbf{Y}} \bar{\mathbf{U}} \bar{\mathbf{W}}^+ \bar{\mathbf{V}}^T, \tag{12}$$

where

$$\bar{\mathbf{W}}^+ := \begin{pmatrix} \bar{w}_1^+ & & \\ & \ddots & \\ & & \bar{w}_n^+ \end{pmatrix} \quad \text{and} \quad \bar{w}_i^+ = \begin{cases} 0 & \text{if } \bar{w}_i = 0 \\ \frac{1}{\bar{w}_i} & \text{if } \bar{w}_i \neq 0 \end{cases} \quad \text{for } i = 1, \dots, n.$$

Once we have computed the optimal solution \mathbf{A}^F according to (12), we compute its singular value decomposition

$$\mathbf{A}^F = \mathbf{U} \mathbf{W} \mathbf{V}^T, \tag{13}$$

where \mathbf{U} and \mathbf{V} are two $n \times n$ orthogonal matrices, and \mathbf{W} is a $n \times n$ diagonal matrix with positive or zero elements (the singular values)

$$\mathbf{W} := \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{pmatrix},$$

such that $w_1 \geq w_2 \geq \dots \geq w_{n-1} \geq w_n$. We denote by $\mathbf{u}_i \in \mathbb{R}^n$ and $\mathbf{v}_i \in \mathbb{R}^n$, for $i = 1, \dots, n$, the columns of matrices \mathbf{U} and \mathbf{V} respectively.

Remark 1. Let \mathbf{M} be an $m \times n$ matrix, and $\mathbf{M} = \mathbf{U}_M \mathbf{W}_M \mathbf{V}_M^T$ its singular value decomposition. On one hand, the columns of the orthogonal matrix \mathbf{V}_M with an associated singular value equal to zero span $\text{Ker } \mathbf{M}$. On the other hand, the columns of the orthogonal matrix \mathbf{U}_M with an associated positive singular value span $\text{Range } \mathbf{M}$, see references [10, 11].

5.2 Selection of Vectors \mathbf{u}^X and \mathbf{u}^Y

From Equation (11) we realize that when the set of points X is hyperplanar the diagonal matrix $\bar{\mathbf{W}}$ has a null singular value: $\bar{w}_n = 0$. In this case, the singular value decomposition of the optimal solution \mathbf{A}^F will also have a null singular value: $w_n = 0$. Therefore, to properly choose \mathbf{u}^X and \mathbf{u}^Y we have to analyze the $\text{Ker } \mathbf{A}^F$ and the $\text{Range } \mathbf{A}^F$.

Lemma 5. *If $\dim \text{Ker } \mathbf{A}^F = 1$, then $\text{Ker } \mathbf{A}^F = \text{span}(\mathbf{v}_n)$.*

Proof. Since $\dim \text{Ker } \mathbf{A}^F = 1$ and $w_1 \geq w_2 \geq \dots \geq w_{n-1} \geq w_n \geq 0$ we have that $w_i > 0$ for $i = 1, \dots, n - 1$ and $w_n = 0$. To finalize, by Remark 1 we know that $\text{Ker } \mathbf{A}^F = \text{span}(\mathbf{v}_n)$. □

Table 1. Algorithm to obtain the affine projection.

<p>STEP 1. Compute the optimal solution \mathbf{A}^F according to Equation (12). STEP 2. Compute the SVD of \mathbf{A}^F according to Equation (13). STEP 3. Set the values of \mathbf{u}^X and \mathbf{u}^Y : 3.a If $w_i > 0$, for $i = 1, \dots, n$. Set $\mathbf{u}^X = \mathbf{n}_{pseudo}^X$ and $\mathbf{u}^Y = \mathbf{n}_{pseudo}^Y$. 3.b If $w_i > 0$, for $i = 1, \dots, n - 1$, and $w_n = 0$. Set $\mathbf{u}^X = \mathbf{v}_n$ and $\mathbf{u}^Y = \mathbf{u}_n$. 3.c If $w_i \geq 0$, for $i = 1, \dots, n - 2$ and $w_{n-1} = w_n = 0$. Degenerated case not applicable to real situations. Stop the algorithm. STEP 4. For any $\bar{\mathbf{x}} \in \mathbb{R}^n$ compute the linear part of the affine projection as</p> $\mathbf{A}(\bar{\mathbf{x}}) = \mathbf{A}^F(\bar{\mathbf{x}} - \langle \bar{\mathbf{x}}, \mathbf{u}^X \rangle \mathbf{u}^X) + \langle \bar{\mathbf{x}}, \mathbf{u}^X \rangle \mathbf{u}^Y.$ <p>STEP 5. Compute the desired affine mapping according to Equation (2)</p> $\varphi(\mathbf{x}) := \mathbf{A}(\mathbf{x} - \mathbf{c}^X) + \mathbf{c}^Y.$
--

Lemma 6. *If $\dim \text{Ker } \mathbf{A}^F = 1$, then $(\text{Range } \mathbf{A}^F)^\perp = \text{span}(\mathbf{u}_n)$, where $^\perp$ denotes orthogonality.*

Proof. Since $\dim \text{Ker } \mathbf{A}^F = 1$ and $w_1 \geq w_2 \geq \dots \geq w_{n-1} \geq w_n \geq 0$ we have that $w_i > 0$ for $i = 1, \dots, n - 1$, and $w_n = 0$. Taking into account Remark 1, we know that $\text{Range } \mathbf{A}^F = \text{span}(\mathbf{u}_1, \dots, \mathbf{u}_{n-1})$. To finalize, since \mathbf{U} is orthogonal we have that $\langle \mathbf{u}_n, \mathbf{u}_i \rangle = 0$, for $i = 1, \dots, n - 1$. Therefore $(\text{Range } \mathbf{A}^F)^\perp = \text{span}(\mathbf{u}_n)$. □

Lemmas 5 and 6 define the desired criterion to select vectors \mathbf{u}^X and \mathbf{u}^Y . That is, to obtain the optimal solution \mathbf{A}^H , we first find the optimal solution \mathbf{A}^F , and based on its singular value decomposition we select the vectors \mathbf{u}^X and \mathbf{u}^Y . Our proposed algorithm is composed of five steps and it is summarized in Table 1.

In the fourth step we obtain the linear part of the affine mapping as

$$\mathbf{A}^F(\bar{\mathbf{x}} - \langle \bar{\mathbf{x}}, \mathbf{u}^X \rangle \mathbf{u}^X) + \langle \bar{\mathbf{x}}, \mathbf{u}^X \rangle \mathbf{u}^Y.$$

In the case that X is a hyperplanar set we can decompose, by Lemma 2, $\bar{\mathbf{x}}_{\mathbb{H}} = \bar{\mathbf{x}}_{\mathbb{H}} + \lambda \mathbf{u}^X$. Therefore, the obtained linear transformation maps $\bar{\mathbf{x}}_{\mathbb{H}}$ to $\mathbf{A}^F \bar{\mathbf{x}}_{\mathbb{H}}$ and $\lambda \mathbf{u}^X$ to $\lambda \mathbf{u}^Y$. Hence, by Proposition 1 we know that this linear mapping is the optimal solution of the minimization of functional H , obtained by means of minimizing F .

5.3 Geometrical Interpretation

Finally, we will prove two additional results that provide a geometrical interpretation to the obtained selection of vectors \mathbf{u}^X and \mathbf{u}^Y in our algorithm.

Table 2. Selection of vectors \mathbf{u}^X and \mathbf{u}^Y according to the sets X and Y .

	Y hyperplanar	Y non-hyperplanar
X hyperplanar	$\dim \text{Ker } \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{n}^X = \mathbf{n}_{pseudo}^X$ $\mathbf{u}^Y = \mathbf{n}^Y = \mathbf{n}_{pseudo}^Y$	$\dim \text{Ker } \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{n}^X = \mathbf{n}_{pseudo}^X$ $\mathbf{u}^Y = \mathbf{u}_n$
X non-hyperplanar	$\dim \text{Ker } \mathbf{A}^F = 1$ $\mathbf{u}^X = \mathbf{v}_n$ $\mathbf{u}^Y = \mathbf{n}^Y = \mathbf{n}_{pseudo}^Y$	$\dim \text{Ker } \mathbf{A}^F = 0$ $\mathbf{u}^X = \mathbf{n}_{pseudo}^X$ $\mathbf{u}^Y = \mathbf{n}_{pseudo}^Y$

Specifically, Lemma 8 states that if X is a hyperplanar set of points, then our algorithm selects \mathbf{u}^X as the unitary normal vector to X : $\mathbf{u}^X = \mathbf{v}_n = \mathbf{n}^X$, which is in fact the natural choice. Moreover, Lemma 9 states that if Y is a hyperplanar set of points, then our algorithm selects \mathbf{u}^Y as the unitary normal vector to Y : $\mathbf{u}^Y = \mathbf{u}_n = \mathbf{n}^Y$, which is also the obvious choice. Table 2 presents the geometrical interpretation of the proposed selection of vectors \mathbf{u}^X and \mathbf{u}^Y .

Lemma 7. *If X is a hyperplanar set of points and \mathbf{n}^X is an unitary normal vector to X , then $\text{Ker } \overline{\mathbf{X}} = \text{span}(\overline{\mathbf{v}}_n) = \text{span}(\mathbf{n}^X)$.*

Proof. Since X is hyperplanar then $\text{Rank } \overline{\mathbf{X}} = n - 1$, see [9]. That is, $\dim \text{Ker } \overline{\mathbf{X}} = 1$. Therefore, $\text{Ker } \overline{\mathbf{X}} = \text{span}(\overline{\mathbf{v}}_n)$, see Remark 1. Since \mathbf{n}^X is an unitary normal vector we have that $\overline{\mathbf{X}}\mathbf{n}^X = \mathbf{0}$, see equation (6). Hence, $\mathbf{n}^X \in \text{Ker } \overline{\mathbf{X}}$. Thus, $\text{span}(\mathbf{n}^X) = \text{Ker } \overline{\mathbf{X}} = \text{span}(\overline{\mathbf{v}}_n)$. \square

Lemma 8. *Let X be a hyperplanar set of points and \mathbf{A}^F the optimal solution of functional F computed according to Equation (12). If \mathbf{n}^X is an unitary normal vector to X and $\dim \text{Ker } \mathbf{A}^F = 1$, then*

$$\text{Ker } \mathbf{A}^F = \text{Ker } \overline{\mathbf{X}} = \text{span}(\overline{\mathbf{v}}_n) = \text{span}(\mathbf{v}_n) = \text{span}(\mathbf{n}^X).$$

Proof. Since $\overline{\mathbf{V}}$ is an orthogonal matrix we have that $\overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = (0 \cdots 0 \ 1)^T$. Moreover, since X is hyperplanar and $\overline{w}_1 \geq \overline{w}_2 \geq \cdots \geq \overline{w}_{n-1} \geq \overline{w}_n \geq 0$, we have that $\overline{w}_n = 0$. Therefore, $\overline{\mathbf{W}} + \overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = \overline{\mathbf{W}} + (0 \cdots 0 \ 1)^T = \mathbf{0}$. Hence

$$\mathbf{A}^F \overline{\mathbf{v}}_n = \overline{\mathbf{Y}} \overline{\mathbf{U}} \overline{\mathbf{W}} + \overline{\mathbf{V}}^T \overline{\mathbf{v}}_n = \mathbf{0}.$$

That is, $\overline{\mathbf{v}}_n \in \text{Ker } \mathbf{A}^F$. Since $\dim \text{Ker } \mathbf{A}^F = 1$ we have that $\text{Ker } \mathbf{A}^F = \text{span}(\overline{\mathbf{v}}_n)$. To finalize, we only have to apply Lemmas 5 and 8. \square

Lemma 9. *If Y is a hyperplanar set of points, $\dim \text{Ker } \mathbf{A}^F = 1$, and \mathbf{n}^Y is an unitary normal vector to Y , then $(\text{Rank } \mathbf{A}^F)^\perp = \text{span}(\mathbf{u}_n) = \text{span}(\mathbf{n}^Y)$.*

Proof. First, since Y is hyperplanar then $\bar{\mathbf{Y}}^T \mathbf{n}^Y = \mathbf{0}$, or equivalently

$$(\mathbf{n}^Y)^T \bar{\mathbf{Y}} = \mathbf{0}^T. \tag{14}$$

Next, we will prove that $(\mathbf{n}^Y)^T \mathbf{U} \mathbf{W} \mathbf{V}^T = \mathbf{0}^T$

$$\begin{aligned} (\mathbf{n}^Y)^T \mathbf{A}^F &= (\mathbf{n}^Y)^T \mathbf{U} \mathbf{W} \mathbf{V}^T && \text{by Equation 13} \\ &= (\mathbf{n}^Y)^T \bar{\mathbf{Y}} \bar{\mathbf{U}} \bar{\mathbf{W}} + \bar{\mathbf{V}}^T && \text{by Equation 12} \\ &= \mathbf{0}^T && \text{by Equation 14.} \end{aligned}$$

Since \mathbf{V} is orthogonal it is invertible. Thus

$$(\mathbf{n}^Y)^T \mathbf{U} \mathbf{W} = \mathbf{0}^T,$$

which is equivalent to the following set of conditions

$$\begin{aligned} (\mathbf{n}^Y)^T \mathbf{u}_1 w_1 &= 0 \\ &\vdots \\ (\mathbf{n}^Y)^T \mathbf{u}_{n-1} w_{n-1} &= 0 \\ (\mathbf{n}^Y)^T \mathbf{u}_n w_n &= 0. \end{aligned}$$

Since $\dim \text{Ker } \mathbf{A}^F = 1$, then $w_i > 0$, for $i = 1, \dots, n-1$, and $w_n=0$. Therefore, \mathbf{n}^Y is orthogonal to $\mathbf{u}_1, \dots, \mathbf{u}_{n-1}$ (the first $n - 1$ columns of matrix \mathbf{U}). To finalize, using Lemma 6, we have that $\text{span}(\mathbf{u}_n) = (\text{Range } \mathbf{A}^F)^\perp = \text{span}(\mathbf{n}^Y)$. □

6 Numerical Examples

Two examples are presented to assess several aspects and advantages of the proposed algorithm. To highlight the analyzed capabilities in both cases we have selected two extremely simple geometries and we have discretized them with a coarse mesh, as it is suggested in [14]. In both examples we first project the source surface onto the target surface [12]. Second, we obtain a structured mesh on the linking sides using a transfinite interpolation algorithm (TFI) [1]. Third, we compute an initial inner node position using a weighted projection from the cap surfaces [3, 5, 9]. Finally, a boundary error correction is added to compute the final location [7, 8]. In addition, both examples are computed using two strategies: 1. projecting only from the cap surfaces, and 2. starting from the cap surfaces, compute the position of the new layer from the previous one in an *advancing front manner*. Note that in these examples we analyze the capability of the projection algorithm to reproduce the

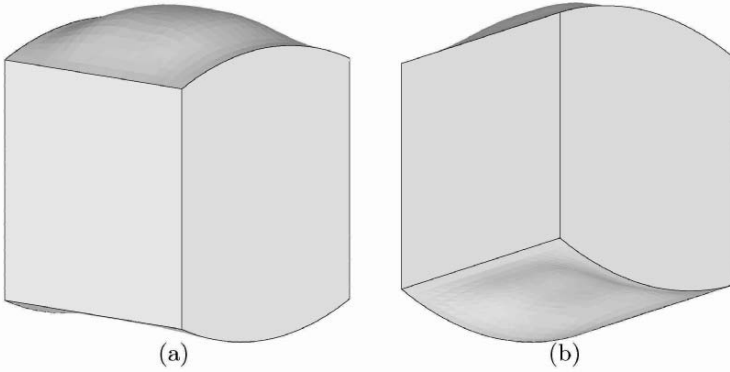


Fig. 3. Geometry of the first example: (a) top view; (b) bottom view

shape of the inner part of the projected mesh. That is, the flattening and the flipping effects. Other issues such as the application to skewed and twisted sweep paths, layers defined by non-affine or non-convex boundaries, or the skewness effect introduced by functional G have been already addressed, see [9, 12] for details. In order to measure the quality of the hexahedral mesh we use the hexahedron shape metric, f_{shape} , defined in [13]. Note that f_{shape} is a normalized measure. Therefore, it always lies in the range $[0, 1]$.

The goal of the first example is to illustrate that, using the developed algorithm, the offset scaling and flattening effects introduced by the minimization of functional F can be avoided. That is, we will show that the obtained affine mapping preserves the shape of the source mesh. Hence, it generates a hexahedral mesh with less low-quality elements than the minimization of functional F . The one-to-one volume is defined by two non-planar surfaces, see figure 3. The boundary loops of both surfaces are symmetric, being the boundary loop of the middle inner layer on the symmetry plane, see figure 4(a). However, the surfaces are not symmetric. Note that they have a non-planar inner part curved in the same orientation.

Figures 4(b) and 4(c) show the central cross-section of the obtained meshes minimizing functionals F and using the proposed algorithm, respectively. For this geometry, as the sweeping process advances from one layer to the next one, the boundary loops become flatter and flatter until a planar boundary loop is reached in the middle of the sweep path. Therefore, the flattening and offset scaling effects produced by the minimization of functional F appear. That is, in each projection the inner shape of the projected mesh becomes more planar. When the planar loop in the middle of the geometry is reached, see figure 4(a), a planar projected mesh is obtained and the offset data of the cap surfaces is completely lost. Nevertheless, the proposed algorithm also imposes that the optimal solution has to map $\mathbf{u}^{\mathbf{X}}$ to $\mathbf{u}^{\mathbf{Y}}$. According to our selection of these vectors, we take into account the offset information of the

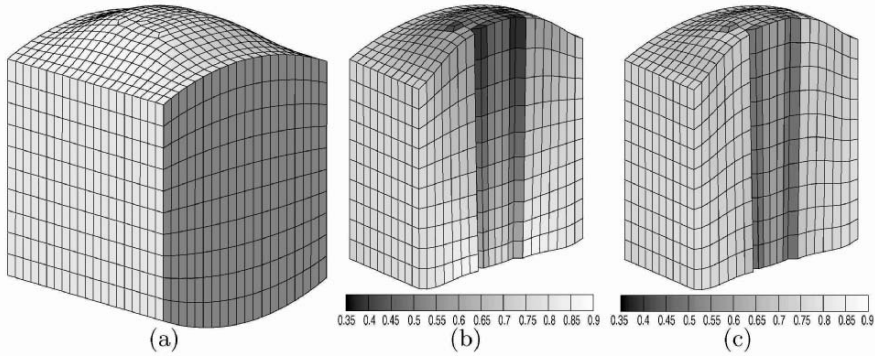


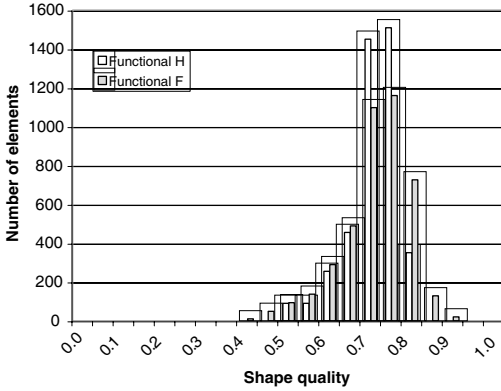
Fig. 4. (a) Surface mesh; (b) central cross-section of the obtained mesh minimizing F ; (c) central cross-section of the obtained using the proposed algorithm.

boundary loops, and in each projection, the location of the inner nodes of the new layer resemble the shape of the cap meshes.

It is important to point out that in this particular case, all the boundary loops are affine. That is, given two boundary loops an affine mapping exists than exactly maps one onto the other. Therefore, functional F becomes null in each projection and the boundary error correction will not improve the initial mesh since it will not be triggered. In other words, the boundary correction will not contribute since there is not error in the projection of the boundaries. On the contrary, the developed algorithm also imposes that the computed affine mapping has to map \mathbf{u}^X to \mathbf{u}^Y . Therefore, in this example it does not exactly map the boundary loops in the pseudo-normal direction, and the boundary error correction will be triggered. Hence, we capture the shape of the source surface and we obtain curved inner layers of elements, see figure 4(c), while the minimization of functional F delivers more planar layers, see figure 4(b). The meshes presented in figures 4(b) and 4(c) are obtained using a layer-by-layer projection procedure. No significant differences have been observed if the projection is computed only from the source and the target surfaces.

Figure 5 shows an histogram of the element quality. Note that using the proposed algorithm we are able to increase the minimum quality value, $\min(f_{shape})$. However, the minimization of F generates elements with a higher value of $\max(f_{shape})$. The general behavior, which we have also observed in other examples, is that the proposed algorithm tends to increase the minimum quality value and to concentrate the quality of the elements around the mean value.

The goal of the second example is to illustrate that the flipping effect introduced by the minimization of functional F can be avoided using the proposed algorithm. In this example we discretize an extrusion volume defined by two surfaces with a rectangular boundary and with non-planar inner part, see



	F	H
$\min(f_{shape})$	0.38	0.47
$\max(f_{shape})$	0.89	0.77
\bar{f}_{shape}	0.69	0.68
$\sigma(f_{shape})$	0.08	0.06

Fig. 5. Mesh quality analysis for the first example. Distribution of the elements according to its quality and statistical values of the quality of the elements.

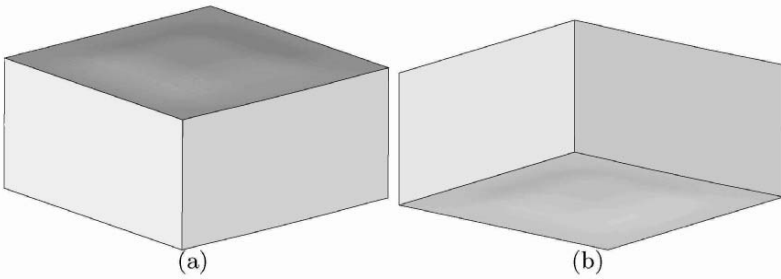


Fig. 6. Geometry of the second example: (a) top view; (b) bottom view

figure 6. These cap surfaces are parallel, and their boundaries are orthogonal to the sweep direction. We start the sweep process by meshing the linking sides. However, due to a bad parameterization of one linking side, the loops of nodes that define the inner layers are not completely planar. Moreover, the loops of nodes are alternatively curved towards and opposed to the sweep direction.

Under these conditions the minimization of functional F introduces the flipping effect and unacceptable flat hexahedral elements with zero volume are obtained, see figure 7(a). Note that since the loops of nodes are affine, the boundary error cannot correct this drawback. However, according to the proposed algorithm we are able to detect the proper direction of vectors \mathbf{u}^X and \mathbf{u}^Y . Hence, the flipping effect due to the minimization of functional F is avoided and a high quality mesh is generated, see figure 7(b). Similarly to the previous example, figures 7(a) and 7(b) are obtained projecting from one layer to the next one in an advancing front manner. No significant differences

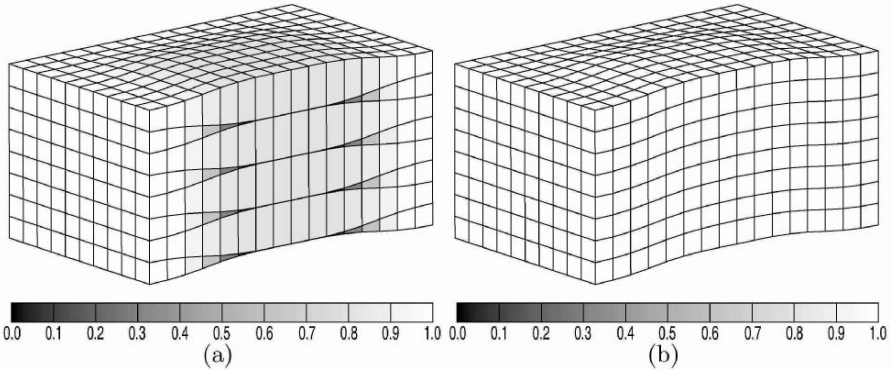


Fig. 7. Central cross-section of the obtained mesh: (a) minimizing F ; (b) using the proposed algorithm.

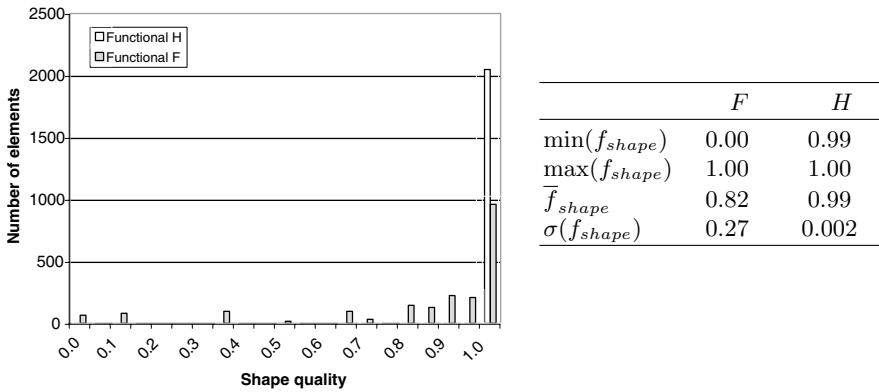


Fig. 8. Mesh quality analysis for the second example. Distribution of the elements according to its quality and statistical values of the quality of the elements.

have been observed if the projections are computed only from the source and the target surfaces.

Figure 8 presents the distribution of the elements according their quality. Note that the presented algorithm generates a mesh such that the quality of the elements verifies $f_{shape} \in [0.95, 1.0]$.

7 Conclusions

In this paper we have proposed and detailed a node projection algorithm to obtain hexahedral meshes in one-to-one sweep geometries. We show that this algorithm, in conjunction with the boundary error procedure, is of major

importance to preserve non-planar shape of the cap surfaces in the inner layers of the hexahedral mesh. Moreover, we claim that the presented algorithm has two additional advantages. First, it provides better node location than the minimization of functionals F and G . Second, since it takes into account the offset data of the cap surfaces (via the vectors \mathbf{u}^X and \mathbf{u}^Y), it triggers the boundary correction procedure when the boundary loops of the layers are affine. To summarize, using this algorithm we are able to overcome flattening, skewness, offset scaling, and flipping effects introduced by the minimization of functional F .

References

1. Thompson JF, Soni B, Weatherill N *Handbook of Grid Generation*. CRC Press, 1999.
2. Owen SJ (1998) A survey of unstructured mesh generation technology. In: 7th Int Meshing Roundtable 239–267.
3. Blacker T (1996) The Cooper Tool. In: 5th Int Meshing Roundtable 13–30.
4. Mingwu L, Benzley SE (1996) A multiple source and target sweeping method for generating all-hexahedral finite element meshes. In: 5th Int Meshing Roundtable 217–225.
5. Knupp PM (1998) Next-generation sweep tool: a method for generating all-hex meshes on two-and-one-half dimensional geometries. In: 7th Int Meshing Roundtable 505–513.
6. Staten ML, Canann SA, Owen SJ (1999) BMSweep: Locating interior nodes during sweeping. *Eng Comput* 15:212–218.
7. Scott MA, Earp MA, Benzley SE, Stephenson MB (2004) Adaptive Sweeping Techniques. In: 14th Int Meshing Roundtable 417–432.
8. White DR, Saigal S, Owen SJ (2004) CCSweep: automatic decomposition of multi-sweep volumes. *Eng Comput* 20:222–236.
9. Roca X., Sarrate J. Huerta A (2005) A new least-squares approximation of affine mappings for sweep algorithms. In: 14th Int Meshing Roundtable 433–448.
10. Gill PE, Murray W, Wright MH (1991) *Numerical Linear Algebra and Optimization*. Addison-Wesley, Edwood City.
11. Lawson C, Hanson R (1974) *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs.
12. Roca X, Sarrate J, Huerta A (2004) Surface mesh projection for hexahedral mesh generation by sweeping. In: 13th Int Meshing Roundtable 169–179.
13. Knupp PM (2004) Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elem Anal Des* 39:217241.
14. Tautges TJ, White DR and Leland RW (2004) Twelve ways to fool the masses when describing mesh generation performance. In: 13th Int Meshing Roundtable 181–190.

Delaunay Meshing

On Refinement of Constrained Delaunay Tetrahedralizations

Hang Si

Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany
si@wias-berlin.de

Summary. This paper discusses the problem of refining constrained Delaunay tetrahedralizations (CDTs) into good quality meshes suitable for adaptive numerical simulations. A practical algorithm which extends the basic Delaunay refinement scheme is proposed. It generates an isotropic mesh corresponding to a sizing function which can be either user-specified or automatically derived from the geometric data. Analysis shows that the algorithm is able to produce provable-good meshes, i.e., most output tetrahedra have their circumradius-to-shortest edge ratios bounded, except those in the neighborhood of small input angles. Good mesh conformity can be obtained for smoothly changing sizing information. The algorithm has been implemented. Various examples are provided to illustrate the theoretical aspects and practical performance of the algorithm.

1 Introduction

Given a three dimensional mesh domain Ω represented by a piecewise linear discretization of its boundary, i.e., $\partial\Omega$ is a set of vertices together with a set of non-crossing segments and facets. The *constrained Delaunay tetrahedralizations* (CDT) \mathcal{T} of $\partial\Omega$ is a tetrahedralization of its vertices and every segment or facet of $\partial\Omega$ is represented as a union of faces of \mathcal{T} . CDTs are useful structures that they not only respect the boundaries of mesh domains but also retain many nice properties of Delaunay tetrahedralizations [19]. It is known that the CDT of a given $\partial\Omega$ does not always exist in three dimensions [1]. By slightly refining $\partial\Omega$ with few additional points, the existence of a CDT is guaranteed [14]. Provably-good algorithms for efficiently constructing CDTs have been proposed [16, 18, 31]. A robust software implementation is publicly available [32].

Generally, CDTs are not well suited for numerical simulations. The mesh quality of CDTs is usually bad, e.g., there are elements which are very skinny or flat and vertices having a big number of connected edges. Numerical methods such as finite element and finite volume methods have special demands on their meshes. In order to obtain accurate results, the mesh elements must

be “well-shaped”, e.g., having small aspect ratio. To reduce the interpolation error, the largest angle of elements should be bounded [3, 17]. For capturing the details of the solution field, the mesh must have smaller size in the region where the solution or its gradient changes rapidly. While in order to reduce the CPU time, the mesh must be as sparse as possible in the rest of the region. In adaptive simulation which the physical problem is solved iteratively, the desired mesh size in a single loop is usually obtained from the solution of the previous iteration through an error estimator. It is convenient to introduce a *sizing function* (or control space [6, 10]) to specify the desired size feature of the problem. For example, the function specifies the (isotropic or anisotropic) mesh size at any point in the domain.

The general *adaptive mesh refinement* problem can be described as follows: given an arbitrary boundary constrained tetrahedralization \mathcal{T} and a sizing distribution function H , find a set of additional points (so-called *Steiner points*) and update \mathcal{T} with these points, such that the resulting mesh only has well-shaped elements and the mesh size conforms to H . In this paper, we study a special case of the general problem by assuming that \mathcal{T} is a CDT. We refer to our problem as *CDT refinement*. The purpose is to develop an efficient method that transfers any CDT into a good-quality mesh for adaptive numerical simulations (see Fig. 1 for an example).

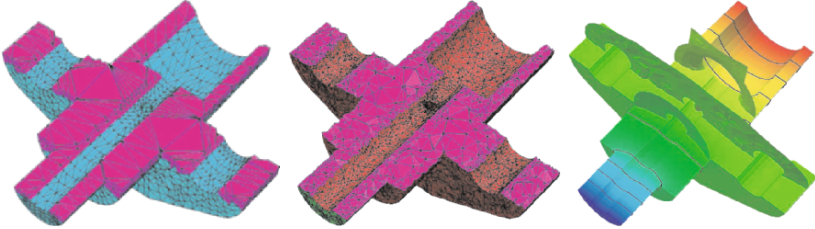


Fig. 1. A CDT of a flange (left), a refined mesh (middle), and the finite volume solution (pdelib [33]) of a static heat equation (right). The refined mesh (20K nodes, 76K tets) was obtained within 3 seconds (on Intel(R) Xeon(TM) CPU 3.60GHz).

On refining CDTs, several merits of the CDT can be exploited. The boundaries of the input domain are respected and refined simultaneously, there are no needs to recover them during the refinement and extract them later. Moreover, the exterior boundaries of the mesh domain are never overrefined. Having the properties close to Delaunay tetrahedralizations, the searching for the nearest feature of any point is local, no additional data structure is needed. It is efficient to classify critical input features (e.g., small input angles) for refining or protecting. However, CDT refinement does not guarantee to remove all badly-shaped elements. To obtain a high-quality computational efficient mesh, after the refinement a mesh smoothing (or mesh optimization) step is necessary.

The Delaunay refinement methods pioneered by Chew [4, 5], Ruppert [8], and Shewchuk [13] are well known for having theoretical guarantees on the quality of mesh elements and producing nicely graded meshes, i.e., the mesh edges are short at small input features and gradually increasing to bigger ones. Moreover, the resulting mesh is conforming Delaunay, which means the edges (or faces) of the dual Voronoi diagram are orthogonal to the faces (or edges) of the mesh. This is a very useful property for finite volume meshes [11, 25]. In three dimensions, only one class of badly-shaped tetrahedra called *slivers* (a sliver has no short edges but nearly zero volume) can survive. The mesh smoothing is essentially used to remove slivers [20, 23, 24]. The main limitation against the elegance of the basic scheme is that no input angle should be smaller than 90° . This condition is not likely to be satisfied in most of the realistic problems. Much work [15, 28, 21, 22] has gone into removing the restrictions. However, most of these methods do not take an arbitrary sizing functions into account.

The algorithm of Miller, et al. [11] finds a well-spaced point set conforming to the domain boundary by sphere-packing, then triangulate the point set by the Delaunay criterion. Recently, Oudot, et al [30] designed a volume meshing algorithm which greedily samples the interior and the boundary of the domain using a similar Delaunay refinement scheme. Both algorithms support user-defined sizing functions. While these methods are not designed for refining CDTs. In stead of that, the boundaries are unknown on input and have to be enforced by the refinement.

Another class of methods [7, 9] which is popularly used for mesh refining works in two parts: (1) point generation using the sizing information, and (2) point insertion by the constrained Delaunay criterion. In part (2), some points are filtered due to the saturation of the near points. This approach is able to quickly generate a number of Steiner points well conforming to the given sizing function and can be parallelized easily. From the theoretical point of view, this approach does not guarantee mesh quality. It heuristically relies on mesh optimization.

In this paper, a practical algorithm that builds on many previous work [7, 8, 13, 30] is proposed. This algorithm, referred to as *constrained Delaunay refinement*, generates an isotropic mesh corresponding to a sizing function H which can be either user-specified or automatically derived. The CDT is refined incrementally by appropriately inserting points into it. At each step, a new point v is generated by the basic Delaunay refinement scheme, v is inserted only if the local mesh is sparse according to H . The process terminates when no new point can be inserted. This algorithm inherits the theoretical guarantees of the basic Delaunay refinement [8, 13]. It generates provably-good quality mesh inside the domain. Those remaining low quality elements are located in the neighborhood of small input angles. This algorithm has been implemented. Practical experiments show that it works both robustly and efficiently. The results validate our claim on the quality of the output tetrahedra. Good mesh conformity is observed for smooth sizing functions.

The remainder of this paper is organized as follows. The proposed algorithm is described in section 2. Section 3 provides theoretical analysis regarding this algorithm. Two approaches for specifying a sizing function are discussed in section 4. Various refinement examples as well as numerical results are presented in section 5. Conclusions are stated in section 6.

2 Constrained Delaunay Refinement

In this section, the algorithm for refining CDTs is presented. It behaves like the basic three dimensional Delaunay refinement algorithm of Shewchuk [13], i.e., finds the badly-shaped tetrahedra and eliminates them by inserting their circumcenters. However, the insertion of circumcenters is restricted by the local mesh sizing information specified on input. We refer to this algorithm as *constrained Delaunay refinement*.

2.1 Definitions

The input of this algorithm is a CDT \mathcal{T} of a *piecewise linear complex* X [11]. The boundaries of X (segments and facets) are represented as a union of *subsegments* and *subfaces* in \mathcal{T} (see Fig. 2). Any tetrahedron τ in \mathcal{T} is *constrained Delaunay*, i.e., the circumsphere of τ encloses no vertex of X that is visible from the inside of τ , the visibility is blocked by the boundaries of X [16]. If τ is constrained Delaunay, it may not be Delaunay. While if τ has no face which is subface, it is Delaunay.

Let $H(p) > 0$ be a *sizing function* defined at any point p in X that specifies the desired lengths of edges connecting at p . H is *isotropic* if the edge length does not vary with respect to the directions at the point, otherwise, it is *anisotropic*. In the scope of this paper, we assume H is isotropic. An ideal sizing function is defined analytically at any point of X . In most cases, H is given discretely at some points in X , the size of other points is obtained by interpolation. We will further discuss the sizing function in section 3.

The *radius-edge ratio* of a tetrahedron τ is the ratio of the circumradius to the shortest edge of τ . If τ has a large radius-edge ratio, then it must be badly-shaped, e.g., it is skinny or flat. However, it is not vice versa. The slivers can

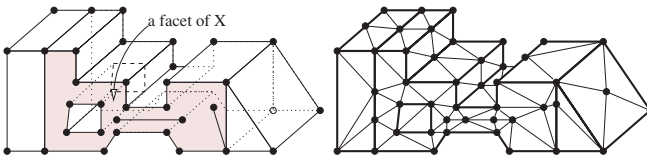


Fig. 2. **Left:** a piecewise linear complex (PLC). The shaded area highlights one of its facets. **Right:** a constrained Delaunay tetrahedralization (CDT) of the left PLC. The surface mesh of the CDT consists of subsegments and subfaces.

have fairly small radius-edge ratio although they are very flat. Nevertheless, radius-edge ratio is effective in classifying the shapes of tetrahedra.

In three dimensions, a subsegment or subface can have infinitely many circumspheres. However, its smallest circumsphere (i.e., the diametric circumsphere) is unique. In the scope of this paper, we tacitly use the term circumsphere to mean the unique one. A subsegment or a subface is said to be *encroached* if a vertex lies inside or on its circumsphere.

2.2 The Algorithm

Given a CDT \mathcal{T} to be refined, a sizing function H , a radius-edge ratio bound B , and two parameters α_1, α_2 . The algorithm incrementally adds Steiner points into \mathcal{T} and updates \mathcal{T} into a refined mesh.

At each step, a Steiner point v is generated by the basic Delaunay refinement scheme, i.e., v is found by the following three *point generating rules*.

- R1 if a subsegment s is encroached, then v is the midpoint of s ;
- R2 if a subface f is encroached, then v is the circumcenter of f . However, if v encroaches upon some subsegments, reject v , use R1 to find a v on one of the encroached subsegments.
- R3 if a tetrahedron t satisfies one of the two cases:
 - (1) t has radius-edge ratio greater than B , or,
 - (2) there is a corner p of t , such that $\alpha_1 H(p) < r$, where r is the radius of the circumsphere of t ,
 then v is the circumcenter of t . However, if v encroaches upon any subsegment of subface, reject v , use R1 or R2 to find a v on one of the encroached subsegments or subfaces.

Once the point v is found, the *point accepting rule* decides whether or not v can be inserted into the mesh. Let P be a set of vertices collected as follows:

- If v is found by R1, P has two endpoints of s .
- If v is found by R2, P has the endpoints of subfaces which v is intended to split.
- If v is found by R3, P has the endpoints of tets which v is intended to split.

Then v can be inserted if $\alpha_2 H(p) < |v - p|$, for all $p \in P$, where $|\cdot|$ is the Euclidean distance. Otherwise, v is not inserted.

If v passes the point accepting rule, it is inserted into the current mesh and the local mesh of v is retriangulated according to the Delaunay criterion.

Remark. R3-1 tests if t has bad quality, and R3-2 checks the H -conformity of the corners of t . R3-1 has priority higher than R3-2, that is, R3-2 is triggered when all tets has radius-edge ratio larger than B .

In the point accepting rule, if v is found by R1 or R2, only the endpoints of the subsegment or subfaces of the same facet where v lies on have the right to

accept or reject v . It appears that v can be too close to some existing vertices in terms of H , i.e., there exists a point $p \notin P$, such that $\alpha_2 H(p) > |v - p|$. We will show in the next section that the distance $|v - p|$ is bounded in terms of α_2 and H .

3 Analysis

The central idea of the algorithm is - only inserts a Steiner point when the local mesh of the point is sparse. The sparseness is indicated by the values of the sizing function at its adjacent vertices. In isotropic case, one can assume each vertex p of the mesh is surrounded by two virtual balls, one *sparse ball* with radius $\alpha_1 H(p)$, and one *protect ball* with radius $\alpha_2 H(p)$. The space outside the sparse ball of p is *sparse* from the viewpoint of p , while the space inside the protect ball of p is free of Steiner points. Notice that when $\alpha_1 \rightarrow +\infty$ (i.e., no sparse space) and $\alpha_2 \rightarrow 0$ (i.e., no protect ball), it is the basic Delaunay refinement algorithm [13].

In the following, we provide conditions on the sizing function H , and the parameters α_1 , α_2 , and B to ensure the theoretical guarantees of this algorithm. Specifically, we will show:

- The termination of the algorithm only depends on α_2 .
- The mesh quality is governed by both B and α_2 .
- The properties of H will influence the mesh conformity.
- The mesh size can be adjusted by α_1 .

For each output vertex v , its *parent* p_v is defined as follows: if v is an input vertex, p_v is the closest output vertex to v ; if v is an inserted vertex, let p_v be the closest vertex to v immediately after v is inserted, if there are several such vertices, choose the one which is the most recently inserted. Notice that p_v may not be the closest output vertex to v .

Given a PLC X , the *local feature size* [8] $lfs(p)$ at any point p is the radius of the smallest ball centered at p that intersects two nonincident features of X (where each of two features might be a vertex, segment or facet). $lfs()$ is defined for all points in X , it satisfies a 1-Lipschitz condition, i.e., for any two points u and v in X , $lfs(v) \leq lfs(u) + |u - v|$.

We use the definition of *input angle* from Cheng et al. [21]. Simply saying, an input angle of the PLC X is any angle that formed by two incident segments, or a segment and a facet, or the dihedral angle formed by two incident facets.

Lemma 1 shows that in the output mesh, the length of the shortest edge of each vertex is bounded.

Lemma 1. *Let v be a vertex of the output mesh, let p be the vertex closest to v , then $|v - p| \geq \min\{\alpha_2 H(v), C\alpha_2 H(p_v), lfs(v)\}$, where $C = \sin\theta_m/\sqrt{2}$, and θ_m is the smallest acute input angle.*

Proof. We prove this lemma by enumerating all cases of the presence of v and p and deriving the bounds on each of them.

Assume v is an input vertex, then $|v - p| = |v - p_v| \geq \alpha_2 H(v)$. Now assume p is an input vertex, then $|v - p| \geq lfs(v)$.

In the following, we examine the cases which both v and p are inserted vertices. The notation $p \prec v$ means p is inserted before v .

Assume v is found by R1. Let s be the segment on which v lies.

- (1) Assume p is found by R1. Let s' be the segment on which p lies.
 - If s and s' are coincident, if $p \prec v$, then $|v - p| = |v - p_v| \geq \alpha_2 H(p_v)$, else, $|v - p| \geq \alpha_2 H(v)$.
 - If s and s' are disjoint, then $|v - p| \geq lfs(v)$.
 - If s and s' share a common input vertex e , let θ be the angle formed by s and s' , $\theta < 90^\circ$ (since p is the closest vertex to v), then $|v - p| \geq |v - e| \sin \theta \geq |v - p_v| \sin \theta \geq \alpha_2 H(p_v) \sin \theta_m$ (see Fig. 3 (a)).
- (2) Assume p is found by R2. Let f be the facet on which p lies.
 - If s and f are disjoint, then $|v - p| \geq lfs(v)$.
 - If s and f share at one common input vertex e , let θ be the angle formed by s and line segment eq , $\theta < 90^\circ$ (see Fig. 3 (b)), then $|v - p| \geq |v - e| \sin \theta \geq |v - p_v| \sin \theta \geq \alpha_2 H(p_v) \sin \theta_m$.
 - If s belongs to f (see Fig. 3 (c)). Suppose $p \prec v$, then p does not encroach upon the segment v splits, hence $|v - p| > |v - p_v|$, so this case is not possible (since p_v is closer to v than p is). The remaining case is $v \prec p$, then $|v - p| \geq \alpha_2 H(v)$.
- (3) Assume p is found by R3. Similar to the last case in (2) (Fig. 3 (c)), the only possible case is $v \prec p$, then $|v - p| \geq \alpha_2 H(v)$.

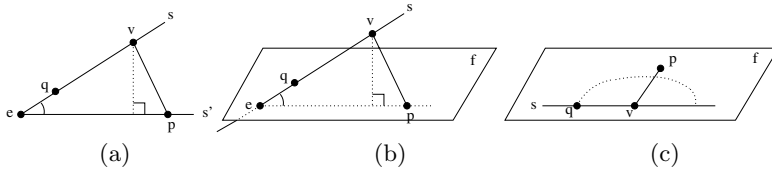


Fig. 3. Suppose v is found by R1, vp is the shortest edge connected at v . q illustrates the possible location of the parent of v .

Assume v is found by R2. Let f be the facet on which v lies:

- (4) Assume p is found by R1. Let s be the segment on which p lies.
 - If s and f are disjoint, then $|v - p| \geq lfs(v)$.
 - If s and f intersect at one input vertex e . Let θ be the input angle formed by s and f , $\theta < 90^\circ$ (refer to Fig. 3 (b), switch the positions of v and p). Then $|v - p| \geq |v - e| \sin \theta \geq |v - p_v| \sin \theta \geq \alpha_2 H(p_v) \sin \theta_m$.
 - If s belongs to f (refer to Fig. 3 (c), switch the positions of v and p). If

$p \prec v$, then $|v - p| \geq |v - p_v| \geq \alpha_2 H(p_v)$, else, let $q \in s$ be either an input vertex or $q \prec v$ (see below), $|v - p| \geq |v - q|/\sqrt{2} \geq |v - p_v|/\sqrt{2} \geq \alpha_2 H(p_v)/\sqrt{2}$.

Now we show that such q must exist. It can be found by the following iterative process: initialize $i := 0, q_0 := p$; (i) let q_{i+1} be the endpoint of the subsegment split by q_i which is closer to v ; if q_{i+1} is an input vertex or $q_{i+1} \prec v$, then $q := q_{i+1}$ and return; else, $i := i + 1$ and goto (i). The iterative process will terminate since R1 has priority higher than R2.

- (5) Assume p is found by R2. Let f' be the facet on which p lies. If f and f' are coincident. If $p \prec v$, then $|v - p| \geq |v - p_v| \geq \alpha_2 H(p_v)$, else, $|v - p| \geq \alpha_2 H(v)$. If f and f' are disjoint, then $|v - p| \geq lf s(v)$. If f and f' intersect at an input vertex e (refer to Fig: 3 (b)), let θ be the input angle formed by the two facets at e , $\theta < 90^\circ$, $|v - p| \geq |v - e| \sin \theta \geq |v - p_v| \sin \theta \geq \alpha_2 H(p_v) \sin \theta_m$. If f' and f intersect at a common segment s , let θ be the input dihedral angle formed by f and f' , $\theta < 90^\circ$, let q be the vertex on s which is the closest one to v , $|v - q| \geq \alpha_2 H(p_v)/\sqrt{2}$ (using the same arguments in case (4)), then $|v - p| \geq |v - q| \sin \theta \geq \alpha_2 H(p_v) \sin \theta/\sqrt{2}$.

- (6) Assume p is found by R3. If $p \prec v$, let $q \in f$ be either an input vertex or $p \prec v$ (such q can be found by using the similar iterative process in case (4) and the fact R2 has higher priority than R3), then $|v - p| \geq |v - q|/\sqrt{2} \geq |v - p_v|/\sqrt{2} \geq \alpha_2 H(p_v)/\sqrt{2}$, else, $|v - p| \geq \alpha_2 H(v)$.

Assume v is found by R3. If $p \prec v$, then $|v - p| = |v - p_v| \geq \alpha_2 H(p_v)$, else, we have the following cases:

- (7) Assume p is found by R1. Let s be the segment on which p lies, similar to case (4), let $q \in s$ be either an input vertex or $q \prec v$, then $|v - p| \geq |v - q|/\sqrt{2} \geq |v - p_v|/\sqrt{2} \geq \alpha_2 H(p_v)/\sqrt{2}$.
- (8) Assume p is found by R2. Let f be the facet on which p lies, similar to case (6), let $q \in f$ be either an input vertex or $q \prec v$, then $|v - p| \geq |v - q|/\sqrt{2} \geq |v - p_v|/\sqrt{2} \geq \alpha_2 H(p_v)/\sqrt{2}$.
- (9) Assume p is found by R3, then $|v - p| \geq \alpha_2 H(v)$.

In the worst case, which is in case (5), $C = \sin \theta_m/\sqrt{2}$. ■

Theorem 1 which guarantees the termination of the algorithm is a directly outcome from the above Lemma.

Theorem 1. *The algorithm terminates if $\alpha_2 > 0$.* ■

Next, we consider the output mesh quality. Our goal is to show that the algorithm is able to create a mesh with most of the tetrahedra have their

radius-edge ratio bounded, only few poor quality tetrahedra remain in well defined locations.

We say a tetrahedron is *skinny* if its radius-edge ratio is smaller than B . A vertex is *sharp* if there are two segments or a segment and a facet sharing at it form an acute angle; a segment is *sharp* if it contains a sharp vertex or there are two facets sharing it form an acute dihedral angle; a facet is *sharp* if it contains a sharp segment or there is another segment or facet adjacent to it forming an acute angle or acute dihedral angle.

Theorem 2. *Suppose the quality bound B is larger than 2. Then there exists a constant D , such that when $\alpha_2 = D$, most of the output tetrahedra have a radius-edge ratio smaller than B . The circumcenter of any skinny tetrahedron is within distance $\sqrt{2}\alpha_2 H(p)$, where p is a sharp vertex or a vertex inserted on a sharp segment or a sharp facet.*

Proof. If there is no acute input angle and $B > 2$, the basic Delaunay refinement algorithm guarantees that the distance of any output vertex v to its nearest neighbor is at least $\frac{lfs(v)}{D_S+1}$, where $D_S > 1$ is a fixed constant (Theorem 6 in [13]). The theorem can be proved if D is chosen sufficiently small such that the inequality

$$D < \frac{lfs(v)}{H(v)(D_S + 1)}$$

is hold for any output vertex v , i.e., the protect ball of v is always empty and no later generated vertex will be rejected.

Assume there are acute input angles. The theorem can be proved by the following procedure. At initialization, choose D_0 to be an arbitrary positive value, and run the constrained Delaunay refinement algorithm with $\alpha_2 := D_0$. In the output mesh, there may have skinny tets. Let t be a remaining poor quality tet, c_t be its circumcenter. t can be categorized into one of the four sets listed below:

- Φ_1 , c_t lies outside the mesh and does not encroach upon any segment or subface.
- Φ_2 , c_t lies inside the mesh and does not encroach upon any segment or subface.
- Φ_3 , c_t encroaches upon a segment (or a subface) which is non-sharp.
- Φ_4 , c_t encroaches upon a sharp segment (or a sharp subface).

Consider a tet $t \in \Phi_1$, there exists at least one segment s (or one subface f) which is encroached by a corner of t , i.e., s (or f) is non-conforming Delaunay, and s (or f) is not split because its circumcenter c_s is rejected by the point accepting rule, i.e., c_s lies inside at least one of the protecting balls of its corners.

We show one special case that such segment exists, other cases can be shown similarly. Assume one face abc of t lies on a facet F (see Fig. 4), bc be the longest edge of abc . Then the circumcircle C of abc must intersect some

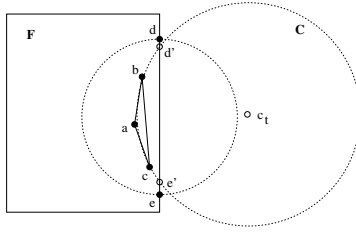


Fig. 4. $t \in \Phi_1$, one special case of the existence of an encroached segment. In the above figure, abc is a face of t and lies inside a facet F , de is the boundary segment of F which is encroached by a .

boundary segment de of F . Moreover, both d and e lie outside or on C (since abc is a Delaunay surface on F). Let the intersections of C and de be d' and e' , then $\angle dae \geq \angle d'ae' \geq 90^\circ$. Hence segment de is encroached by a .

t will be eliminated if s (or f) is split by the constrained Delaunay refinement. This can be achieved by shrinking the protect balls of the endpoints of s (or f) such that its circumcenter lies outside all of them. It is possible to choose a sufficiently small $D_1 > 0$, such that all tets of Φ_1 can be removed by running the algorithm with $\alpha_2 := D_1$. The newly inserted vertices may create new poor quality tets which can be classified into Φ_2 , Φ_3 , and Φ_4 .

Consider a tet $t \in \Phi_2$, c_t is rejected by some protect balls of existing vertices at the neighborhood of t . t can be eliminated by shrinking these protect balls such that c_t lies outside all of them. It is possible to choose a sufficiently small D_2 , $0 < D_2 \leq D_1$, such that no $t \in \Phi_2$ can survive after running the algorithm with $\alpha_2 := D_2$. There are possibly remaining poor quality tets of Φ_3 and Φ_4 .

Consider a tet $t \in \Phi_3$, the circumcenter of s (or f) is rejected by lying inside some protect balls of its endpoints. s (or f) will be split by shrinking these protect balls. Consequently, either t gets eliminated during the split of s (or f), or c_t does not encroach any segment or surface and is accepted for insertion, or t becomes a tet of Φ_2 . It is possible to choose a sufficiently small D_3 , $0 < D_3 \leq D_2$, such that no $t \in \Phi_2 \cup \Phi_3$ can survive after running the algorithm with $\alpha_2 := D_3$. Now the possibly remaining poor quality tets can only belong to Φ_4 .

If $t \in \Phi_4$, the circumcenter c_s of s (or f) is rejected by lying inside some protect balls of its endpoints (see Fig. 5). Let p be such a vertex, then, $|c_t - c_s| < |c_s - p| < \alpha_2 H(p)$. Hence $|c_t - p| < \sqrt{2}\alpha_2 H(p)$. ■

Next, we consider the mesh conformity with respect to the sizing function H . For each vertex v , let $S(v)$ and $L(v)$ denotes the lengths of the shortest edge and the longest edge among all edges containing v , respectively. We are interesting the values $\frac{S(v)}{H(v)}$ and $\frac{L(v)}{H(v)}$. Theorem 3 gives bounds on those output vertices at where the local mesh quality is met.

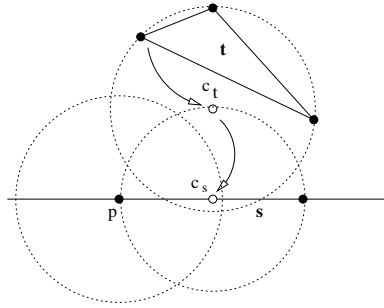


Fig. 5. $t \in \Phi_4$, the circumcenter c_t of t encroaches upon the circumsphere of segment s . The circumcenter c_s of s lies inside the protecting ball of one of the endpoint p .

Theorem 3. *Suppose all tets containing v have their radius-edge ratio bounded, and the rule R3-2 is not applicable on any of them, then:*

- (i) $\frac{S(v)}{H(v)} \geq \min\{\alpha_2, C\alpha_2 \frac{H(p_v)}{H(v)}, \frac{lfs(v)}{H(v)}\}$.
- (ii) $\frac{L(v)}{H(v)} \leq 2\alpha_1$;

Proof. The first claim follows directly from Lemma 1. Let t be a tet which contains v and has the longest edge of length $L(v)$, let r be the circumradius of t , then: $\frac{L(v)}{2} \leq r \leq \alpha_1 H(v) \implies \frac{L(v)}{H(v)} \leq 2\alpha_1$. ■

When the local mesh quality is satisfied, and the mesh is saturated, Theorem 3 shows that the mesh conformity at each vertex v is related to H , α_1 , α_2 and lfs . Specifically,

- H , α_2 , and lfs together decide the lower bound of the mesh conformity.
- The term $\frac{H(p_v)}{H(v)}$ indicates that H should not vary too much in v 's neighborhood, e.g., H is 1-Lipschitz. The term $\frac{lfs(v)}{H(v)}$ indicates that H is constrained by lfs which is dependent upon the boundary of the CDT.
- α_2 plays a contradictory role between mesh quality and mesh conformity. It needs to be small in order to guarantee the mesh quality. While it is desired to be as large as possible for the good mesh conformity.
- α_1 limits the length of the longest output edge connected at v . It directly controls the result mesh size, i.e., the smaller it is, the bigger the mesh size will be.

4 Specify Sizing Functions

Our algorithm needs a sizing function H which is defined over the mesh domain and specifies the local mesh size, e.g., the desired edge length or element density. The data of a sizing function can be based on either a priori known information or a posteriori error estimation.

This section describes two approaches for specifying a sizing function. If no sizing function is given by the user, it can be automatically derived by using the boundary data of the CDT. Alternatively, a background mesh whose vertices contain the size information can be supplied along with the CDT.

4.1 Sizing Function Derived from CDT

It is possible that an appropriate sizing function may not be available in advance. For example, on the first time to create a mesh for simulation, there is not much mesh size information available¹. In such case, H is derived from the CDT \mathcal{T} of a PLC X by the following approach:

- if p is not a Steiner point, then $H(p) := lfs(p)$;
- else $H(p)$ is interpolated from its adjacent vertices by the Shepard interpolation, where the weights are set to be the second inverse power of distances, i.e.,

$$H(p) := \frac{\sum_{i=1}^n |p - v_i|^{-2} H(v_i)}{\sum_{i=1}^n |p - v_i|^{-2}}.$$

where v_i is a vertex connecting to p in current mesh.

The use of Shepard interpolation [2] has the effect that the closest node has the biggest influence on the size of the Steiner point. An example is shown in Fig. 6.

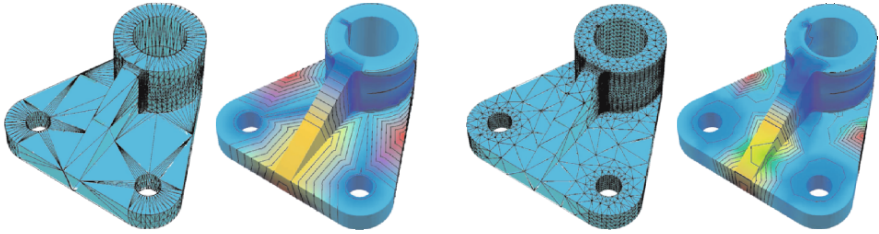


Fig. 6. Sizing function derived from local feature sizes and Shepard interpolation. The left two pictures show a CDT and the local feature sizes at its vertices. On the right, the refined mesh and the obtained sizing function are shown.

The above approach is simple enough in practice. If p is a vertex of the initial CDT, $lfs(p)$ can be efficiently computed by searching locally the smallest distance of the nearest vertex, subsegment, and subface. Another feature is that the sizing function is computed on the fly, i.e., each Steiner point gets its size after it is inserted. There is no need to store H . The mesh quality

¹Although some pre-knowledge of the problem may help for deciding the size of the mesh on its boundary. However, it is generally hard to decide the appropriate mesh size inside the volume in advance.

can be improved by choosing an appropriate value for α_2 . There are similar approaches [7, 9, 29] but they require additional data structures, e.g., a background mesh or a KD-tree.

4.2 Use of a Background Mesh

If H is known in advance, the most popular and flexible way for specifying H is through a background mesh whose vertices or elements encode the information about the desired mesh size. The background mesh can be any grid structure (such as uniform grid or Octree) or an unstructured mesh (such as CDT).

We use an unstructured mesh as the background mesh. Hence it can be the initial CDT or a tetrahedral mesh obtained at the previous iteration in an adaptive process. At any point p of the current mesh, $H(p)$ can be obtained by means of interpolation in the background mesh:

- locate p in a tetrahedron t which contains p ;
- compute $H(p)$ as the P^1 interpolation of the sizes $H(p_i)$ at the vertices p_i of t .

5 Examples

The algorithm has been integrated in TetGen – a quality tetrahedral mesh generator [32]. The input can be either a PLC or a CDT. A sizing function H can be optionally specified using a background mesh. If H is not available, it will be automatically generated by the approach discussed in section 4.1. Parameters B , α_1 , and α_2 are all adjustable by command line options. Each of them has a default value ($B = 2.0$, $\alpha_1 = \sqrt{2}$, $\alpha_2 = 0.5$) if it is not specified on input.

The next two examples (Figs. 7 and 8) were built for analysis purpose. We study the effects of using different combinations of the parameters (B , α_1 , and α_2) and compare the results according to the theoretical analysis of section 3.

Figure 7 are serials of meshes created by various combinations of (B, α_2) . The sizing functions are automatically derived. Remaining poor quality tets are plotted for selected meshes. The mesh size of each mesh is given in the form $n_v/n_t/n_b$, where n_v – the number of nodes, n_t – the number of tets, and n_b – the number of remaining poor quality tets. The results validate our claim (Theorem 2) on the mesh quality, i.e., for an appropriate α_2 , most of tets have bounded radius-edge ratio, poor quality tets are all close to small input angles. Notice that few slivers may remain in the volume, they can be removed by a mesh smoothing step.

Figure 8 show three meshes of a unit cube obtained by specifying different sizing functions H through background meshes. The H s used in these meshes are piecewise smooth functions. The parameters are chosen as follows: $B =$

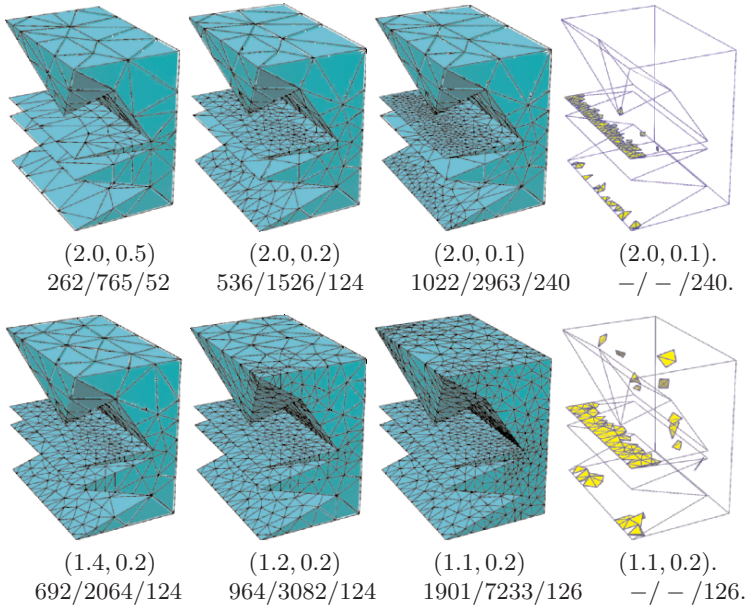


Fig. 7. Given a CDT with small input angles, the above pictures are different meshes refined by using various combinations of parameters (B, α_2) and automatically deriving sizing functions.

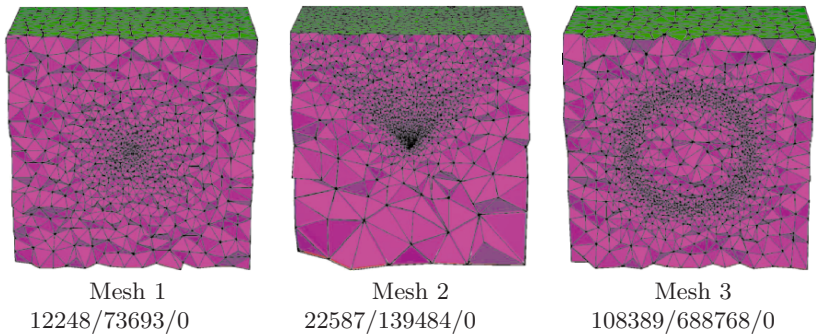


Fig. 8. Meshes created by specifying different sizing functions.

2.0, $\alpha_1 = \sqrt{2}$, and $\alpha_2 = 0.05$. Table 1 lists the statistics of the ratios $\frac{S(v)}{H(v)}$ and $\frac{L(v)}{H(v)}$ (defined in Theorem 3) at mesh vertices. The statistics show that the meshes are well conformed to the corresponding H_s .

In the following, some selected examples which illustrate the practicability of this algorithm are presented (Fig. 10 to Fig. 12). The complexity of the input geometries² challenges both the robustness and efficiency of the algo-

²Available from Inria's large repository: <http://www-rocq1.inria.fr/gamma>

Table 1. Statistics of the ratios $S_v = \frac{S(v)}{H(v)}$ and $L_v = \frac{L(v)}{H(v)}$ (defined in Theorem 3) at mesh vertices of the meshes shown in Fig. 8.

	Mesh 1		Mesh 2		Mesh 3	
	S_v	L_v	S_v	L_v	S_v	L_v
< 0.5	0	0	0	0	0	0
$0.5 - 1/\sqrt{2}$	58	0	0	0	0	0
$1/\sqrt{2} - 1$	3221	1	283	0	0	0
$1 - \sqrt{2}$	15062	113	10778	14	1927	49
$\sqrt{2} - 2$	4246	3867	1187	1044	94186	12594
$2 - 2\sqrt{2}$	0	18606	0	11190	12276	95746
$> 2\sqrt{2}$	0	0	0	0	0	0

Table 2. Statistics of mesh sizes and CPU times (examples in Fig. 9 to Fig. 11). Tested on Intel(R) Xeon(TM) CPU 3.60GHz, 2G Memory.

	Input CDT		Quality Mesh		CPU time (sec.)
	n_v	n_t	n_v	n_t	
747	6,364	19,626	300,712	1,783,953	36.80
Peugeot	7,689	20,689	474,678	2,706,469	84.91
Engine	59,233	190,768	410,288	1,506,073	135.06

rithm. Table 2 shows the statistics on the size of input and resulting meshes, the CPU time for refining CDTs. Figure 9 are radius-edge ratio histograms of the resulting meshes.

6 Conclusion and Discussion

In this paper, the problem of refining constrained Delaunay tetrahedralizations is raised in the context of adaptive numerical simulations. A practical algorithm is proposed. This algorithm inherits the simplicity and elegance of the basic Delaunay refinement scheme. While it has no restriction on the angles of the inputs and it takes a user-specified sizing function into account. The algorithm refines a CDT by fast distributing additional points (Steiner points) in its sparse area. Theoretical analysis shows that the mesh quality can be provably-good, good mesh conformity can be obtained for smooth sizing functions.

There are possibilities to improve the mesh quality and reduce the mesh size. Notice that the the circumcenter used in basic Delaunay refinement scheme might not be the best position for a Steiner point. Alternatively, the off-center [27] may be considered.

Although this algorithm is proposed for refining CDTs, it can be used to refine any boundary constrained tetrahedralizations (non-CDTs) as well. Notice that the point generating rules and the point accepting rule do not rely on the type of inputs. A CDT is eligible for an efficient implementation

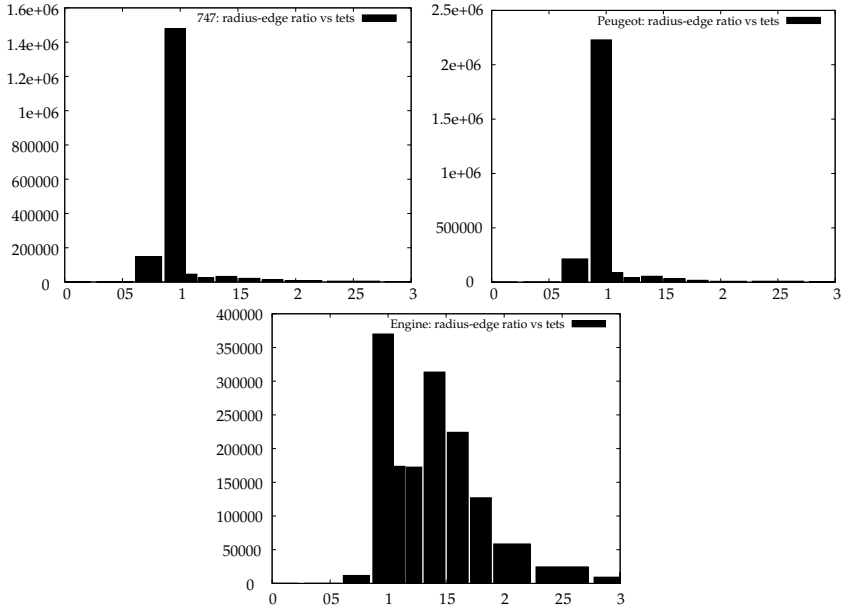


Fig. 9. Radius-edge ratio histograms (examples in Fig. 10 to Fig. 12).

of these rules. It is not the case for arbitrary non-CDTs that required features may not be found locally. While a background grid (such as Octree) may be used.

It would be interesting to adapt the algorithm for anisotropic H . For this purpose, the usual ways of measuring edge lengths, updating locally Delaunay property around Steiner points, and the interpolation of H must be extended by anisotropic manners [10].

Acknowledgements

This work is supported by the “pdelib” project of Weierstrass Institute for Applied Analysis and Stochastics. Thanks to Dr. Klaus Gärtner for many helpful discussions. Thanks to the reviewers for the valuable suggestions and comments.

References

1. Schönhardt E (1928) Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Mathematische Annalen* 98:309–312
2. Shepard D (1968) A Two-Dimensional Interpolation Function for Irregularly Space Data. In: *Proc. 23th Nat. Conf. ACM*, 517–524.

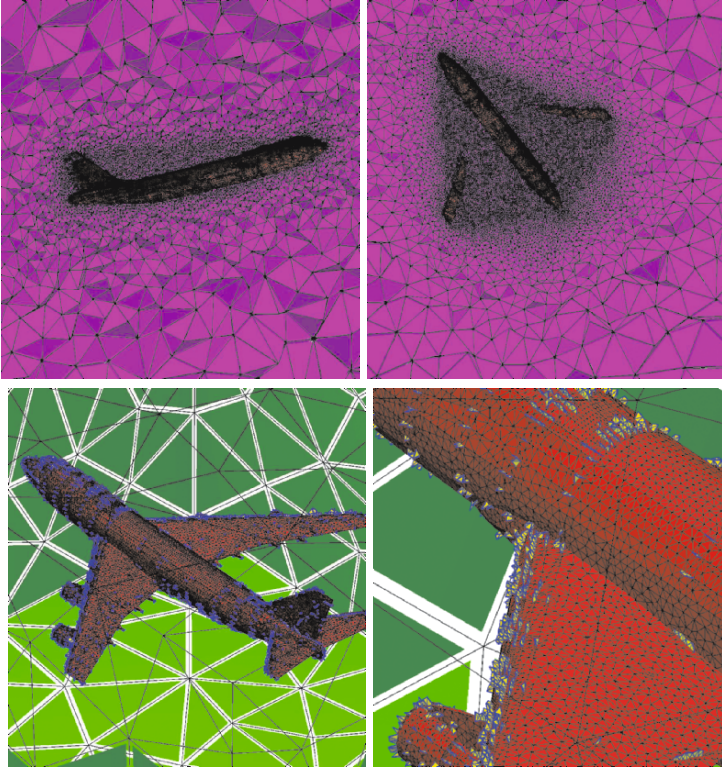


Fig. 10. 747: The geometry: a plane in the middle of a bounding box, the internal region of the plane is not meshed. H is specified on the CDT of the input PLC. The size is simply chosen such that H is small at the surface of the plane and big on the bounding box. Parameters: $B = 2.0$, $\alpha_1 = \sqrt{2}$, and $\alpha_2 = 1/\sqrt{2}$. **Top left and right**, two different views of the resulting mesh. **Bottom left**, an overview of the remaining poor quality tets in the mesh (which are all close to the surface mesh of the plane). **Bottom right**, a detailed look of the bad tets on one of the wings.

3. Babuška I, Aziz, A K (1976) On the Angle Condition in the Finite Element Method. *SIAM Journal on Numerical Analysis* 13(2): 214–226.
4. Chew P L (1989) Guaranteed-Quality Triangular Meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University.
5. Chew P L (1997) Guaranteed-Quality Delaunay Meshing in 3D. In: Proc. 19th Annu. Sympos. Comput. Geom., 274–280.
6. George P L (1991) Automatic Mesh Generation. Application to Finite Element Methods. Wiley.
7. Weatherill N P, Hassan O (1994) Efficient Three-Dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints. *Int. J. Numer. Meth. Engng* 37: 2005–2039.
8. Ruppert J (1995) A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *J Algorithms* 18(3): 548–585

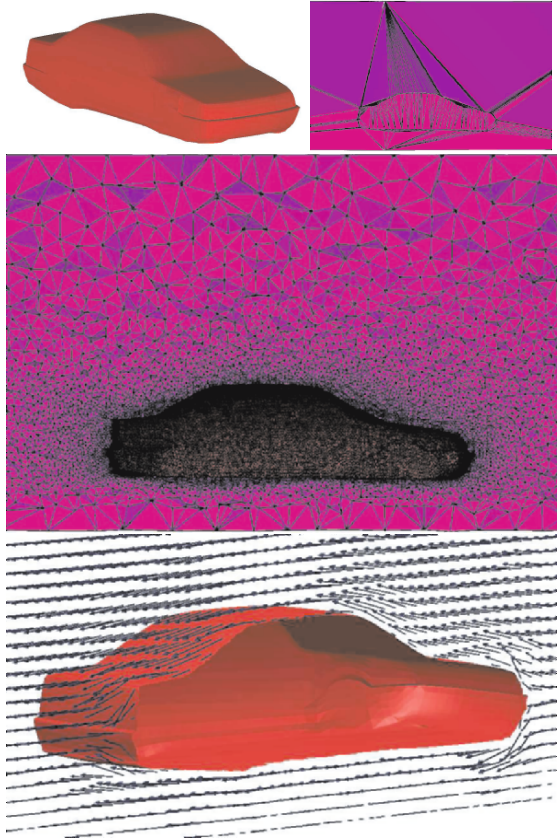


Fig. 11. Peugeot: Test case of incompressible Navier Stokes simulation (solved by AcuSolve [34]). Parameters: $B = 2$, $\alpha_1 = \sqrt{2}$, and $\alpha_2 = 1/\sqrt{2}$. **Top left**, the geometry - a car body inside a channel, the internal of the car is not meshed. **Top right**, the background mesh (a CDT) used for specifying H which is small on the car surface and big on the wall of the channel. **Middle**, a view of the resulting mesh. **Bottom**, the resulting velocity field.

9. Frey P J, Borouchaki H, George P L (1996) Delaunay Tetrahedralization Using an Advancing Front Approach. In: Proc. of 5th Intl. Meshing Roundtable.
10. Frey P J, George P L (2000) Mesh Generation. Application to Finite Elements. Hermès, Paris.
11. Miller G L, Talmor D, Teng S H, Walkington N, Wang H (1996) Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening. In: Proc. of 5th Intl. Meshing Roundtable.
12. Freitag L, Ollivier-Gooch C (1996) A Comparison of Tetrahedral Mesh Improvement Techniques. In: Proc. 5th Intl. Meshing Roundtable.
13. Shewchuk J R (1998) Tetrahedral Mesh Generation by Delaunay Refinement. In: Proc. 14th Annu. Sympos. Comput. Geom.

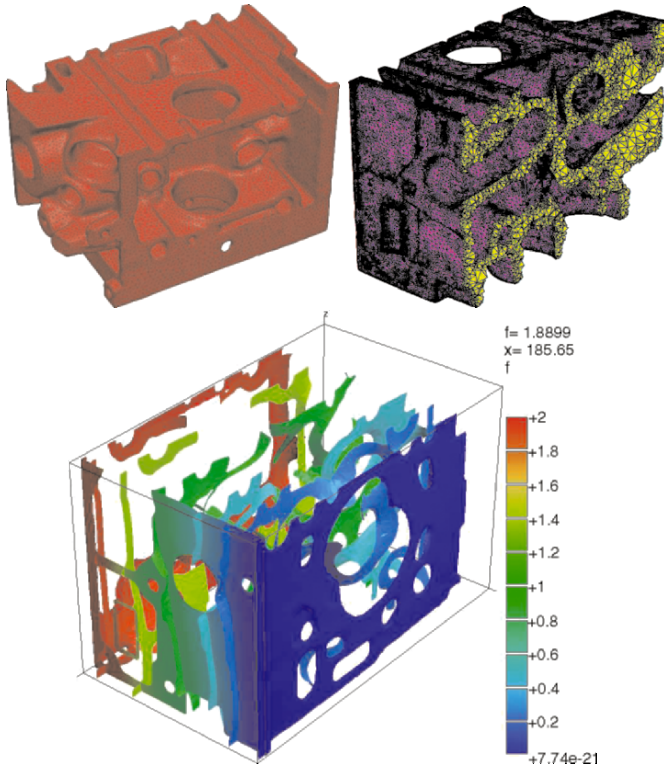


Fig. 12. Engine: The sizing function is automatically derived from the input CDT. Parameters: $B = 2.0$, $\alpha_1 = \infty$, $\alpha_2 = 1/\sqrt{2}$. **Top left**, the input CDT. **Top right**, a view of the resulting mesh. **Bottom**, the solution of a static heat equation (solved by pdelib [33]).

14. Shewchuk J R (1998) Tetrahedral Mesh Generation by Delaunay Refinement. In: Proc. 14th Annu. Sympos. Comput. Geom.
15. Shewchuk J R (2000) Mesh Generation for Domains with Small Angles. In: Proc 16th Annu. ACM Sympos. Comput. Geom.
16. Shewchuk J R (2002) Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery. In: Proc. 11th Intl. Meshing Roundtable.
17. Shewchuk J R (2002) What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In: Proc. 11th Intl. Meshing Roundtable.
18. Shewchuk J R (2003) Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips. In: Proc. 19th Annu. Sympos. Comput. Geom.
19. Shewchuk J R (2005) General-Dimensional Constrained Delaunay and Constrained Regular Triangulations. To appear in Discrete & Computational Geometry.
20. Cheng S W, Dey, T K, Edelsbrunner H, Facello M A, Teng S H (1999) Sliver Exudation. J. ACM, 47, 883–904.

21. Cheng S W, Dey T K, Ramos E A, Ray T (2004) Quality Meshing for Polyhedra with Small Angles. In: Proc. 20th Annu. ACM Sympos. Comput. Geom.
22. Cheng S W, Dey T K, Ray T (2005) Weighted Delaunay Refinement for Polyhedra with Small Angles. In: Proc. 14th Intl. Meshing Roundtable.
23. Edelsbrunner H, Li X Y, Miller G, Stathopoulos A, Talmor D, Teng S H, Üngör A, Walkington N (2000) Smoothing Cleans up Slivers. In: Proc. 32th Annu. ACM Sympos. Theory of Computing, 273–277.
24. Edelsbrunner H, Guoy D. (2002) An Experimental Study of Sliver Exudation. *Engineering with Computers*, 18, 299–240.
25. Fuhrmann J, Langmach H (2001) Stability and Existence of Solutions of Time-Implicit Finite Volume Schemes for Viscous Nonlinear Conservation Laws. *Appl. Numer. Math.*, 37(1-2): 201–230.
26. Du Q, Wang D. (2003) Tetrahedral Mesh Generation and Optimization Based on Centroidal Voronoi Tessellations. *Int. J. Numer. Meth. Engng*, 56, 1355–1373.
27. Üngör A (2004) Off-centers: A New Type of Steiner Points for Computing Size-Optimal Guaranteed-Quality Delaunay Triangulations. In: Proc. LATIN, 152–161.
28. Pav S, Walkington N (2004) A Robust 3D Delaunay Refinement Algorithm. In: Proc. 13th Intl. Meshing Roundtable.
29. Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M (2005) Variational Tetrahedral Meshing. *ACM Transactions on Graphics*, 24(3): 617–625.
30. Oudot S, Rineau L, Yvinec M (2005) Meshing Volumes Bounded by Smooth Surfaces. In: Proc. 14th Intl. Meshing Roundtable.
31. Si H, Gärtner K (2005) Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations. In: Proc. 14th Intl. Meshing Roundtable.
32. TetGen, a Quality Tetrahedral Mesh Generator. <http://tetgen.berlios.de>.
33. pdelib, Tool Box for Solving Partial Differential Equations. <http://www.wias-berlin.de/software/pdelib>
34. AcuSolve, a Finite Element Incompressible Flow Solver. <http://www.acusim.com>.

Smooth Delaunay–Voronoi Dual Meshes for Co-Volume Integration Schemes

Igor Sazonov, Oubay Hassan, Kenneth Morgan and Nigel P. Weatherill

Civil & Computational Engineering Centre, University of Wales Swansea,
Singleton Park, Swansea, SA2 8PP, U.K.
i.sazonov@swansea.ac.uk

1 Introduction

Yee's scheme for the solution of the Maxwell equations [1] and the MAC algorithm for the solution of the Navier–Stokes equations [2] are examples of co-volume solution techniques. Co-volume methods, which are staggered in both time and space, exhibit a high degree of computational efficiency, in terms of both CPU and memory requirements compared to, for example, a finite element time domain method (FETD). The co-volume method for electromagnetic (EM) waves has the additional advantage of preserving the energy and, hence, maintaining the amplitude of plane waves. It also better approximates the field near sharp edges, vertices and wire structures, without the need to reduce the element size. Initially proposed for structured grids, Yee's scheme can be generalized for unstructured meshes and this will enable its application to industrially complex geometries [3].

Despite the fact that real progress has been achieved in unstructured mesh generation methods since late 80s, co-volume schemes have not generally proved to be effective for simulations involving domains of complex shape. This is due to the difficulties encountered when attempting to generate the high quality meshes that satisfying the mesh requirements necessary for co-volume methods. In this work, we concentrate on EM wave scattering simulations and identify the necessary mesh criteria required for a co-volume scheme. We also describe several approaches for generating two-dimensional and three-dimensional meshes satisfying these criteria. Numerical examples on the scattering of EM waves show the efficiency and accuracy that can be achieved with a co-volume method utilising the proposed meshing scheme.

2 Mesh Requirements

For co-volume integration schemes to be implemented on unstructured meshes, the triangulation has to satisfy a number of criteria. For EM simulations, since

the schemes are staggered in space, co-volume methods need two mutually orthogonal meshes for the electric and magnetic fields. The dual Delaunay–Voronoi diagram is the obvious choice. In 2D, corresponding edges of the Voronoi and Delaunay meshes are mutually orthogonal. In 3D, every edge of the Voronoi diagram is orthogonal to the corresponding face of the Delaunay triangulation and vice versa.

Both meshes must be sufficiently smooth, i.e. the edges in both meshes must not be too short or too long. The stability property of an explicit implementation determines the time step in terms of the shortest edge in both meshes. Thus, all the edges should have length $\mathcal{O}(\delta)$, where δ is the recommended element size based upon the characteristic wavelength, λ , of the problem. The typical value used in many simulations is $\delta = \lambda/15$.

In addition, from the view point of accuracy, the Delaunay mesh should not contain bad elements, where, in this context, an element is defined as bad if its circumcentre is located outside the element.

A brief explanation of the main requirements can be found in the Appendix.

3 Mesh Quality Criteria

The criteria employed to determine the suitability of the mesh have to reflect the requirements of the numerical solution procedure. The simple formula

$$Q = \beta \frac{\min\{l_D, l_V\}}{\langle l_D \rangle} \quad (1)$$

is very useful to estimate the quality of the mesh built for wave scattering purposes. Here, $\beta = \sqrt{8}$ is a normalization coefficient, which gives $Q = 1$ for an ideal mesh (see Section 5), l_D is the Delaunay edge length, l_V is the Voronoi edge length and $\langle l_D \rangle$ is the average Delaunay edge length. The minimum is taken over all sides of both meshes.

If the mesh contains some bad elements, then the fraction,

$$r_{\text{bad}} = \frac{N_e^{\text{bad}}}{N_e}, \quad (2)$$

is also a criterion which may be used to compare different meshes. Here N_e is the total number of elements in the mesh and N_e^{bad} is the number of bad elements.

A sufficient condition to ensure the smoothness of the Voronoi diagram, is to ensure that every element has its circumcentre inside the element and located as far as possible from its edges or faces. In this case, the measure

$$q_e = 3 \frac{h_e}{R_e} \equiv 3 \cos \left(\max_{i=1,\dots,4} \alpha_i \right) \quad (3)$$

is employed to estimate the quality of each individual element. In this expression, R_e is the circumradius, $h_e = \min_{i=1,\dots,4} h_i^e$ is the signed shortest distance from the circumcentre to an element edge or face (set negative if the circumcentre lies outside the element). For a perfect tetrahedral element $q_e = 1$; $q_e < 0$ for a bad element. This quality parameter q_e is also related to the maximal vertex angle α_i of an element. In 2D the vertex angle is simply the angle between the two edges connected to this vertex. In 3D the analogous tetrahedron vertex angle has a more complicated definition. The angle α_i of vertex P_i is defined as the angle under which a circumcircle of the face op-

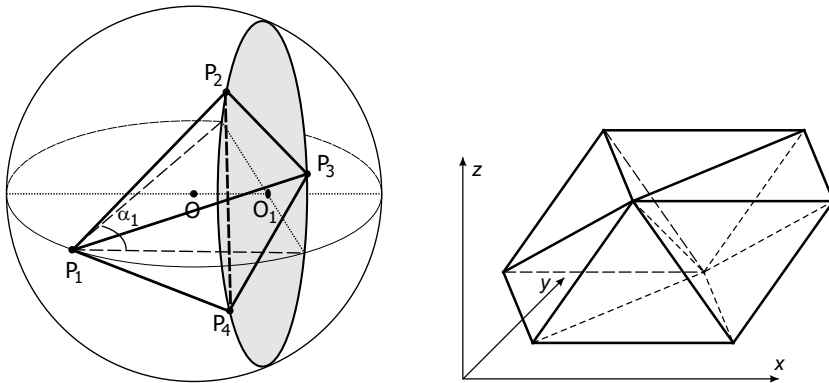


Fig. 1. On the definition of 3D element angle (left). Six elements of 3D ideal mesh form a parallelepiped tiling the space (right).

posite to P_i is seen from P_i in the plane passing through P_i and centre of element’s circumsphere, O , and face’s circumcircle, O_i , as seen in figure 1. The segment OO_i has length h_i . A good quality element, with $q_e > 0$, will have all acute vertex angles. It can be shown that, if $q_e > 0$ for all elements of the mesh, then the mesh is guaranteed to be Delaunay satisfying [4].

It can be shown that the necessary condition to ensure a good quality tetrahedral element, $q_e > 0$, with circumcentre located inside the element, is that the four triangular faces of the element should have acute angles [4]. Therefore, it is important to ensure that the initial boundary triangulation consists of triangles with all acute angles.

If all the tetrahedron vertex angles defined above are acute, some dihedral angles can be right or obtuse. A tetrahedron with all acute dihedral angles can have its circumcentre located outside its volume and hence can have an obtuse vertex angle.

Notice also, that although the tetrahedron vertex angle and the solid angle for the same vertex are not related uniquely, nevertheless the solid angle is

less/equal/greater than 2π , if the vertex angle is acute/right/obtuse, respectively.

4 Traditional Meshing Methods

Traditional unstructured mesh generation methods, such as the advancing front technique (AFT) [5] and the Delaunay triangulation [6], as well as their combinations [7], are not designed to guarantee the creation of a mesh meeting the requirements set out above, even in the two-dimensional case. These methods generate meshes in which the element edge length is normally acceptable, but the corresponding Voronoi diagram may often be highly irregular, with some very short Voronoi edges. Thus, these methods cannot guarantee the regularity of the edge lengths of the dual mesh and the absence of bad elements, even in 2D. Mesh improvement methods, based on swapping, reconnection [8] and smoothing, improve the quality of 2D meshes. Nevertheless, a significant number of very short Voronoi edges and bad elements, located mainly near the domain boundary, remain in the final mesh [9]. In 3D, the advancing front technique (AFT) can produce meshes with about 60% of bad elements and these are unsuitable for a co-volume solution scheme.

A promising approach is the construction of the centroidal Voronoi tessellation (CVT) and its dual Delaunay mesh. The CVT relocates the generated nodes to be at the mass centroids of the corresponding Voronoi cells with respect to a given density function [10]. A new Voronoi tessellation of the relocated nodes is produced. This process, which is called Lloyd's algorithm [11], can be repeated until all nodes are close enough to the corresponding centroids. Lloyd's algorithm needs an initial mesh, which can be generated by any method. In addition to relocating the nodes, the CVT scheme changes the mesh topology. Although the quality of final mesh is much higher than the quality of the initial mesh, it is normally not suitable for the successful application of co-volume solution schemes: in 3D, the share of bad elements is usually around 10% of the total elements, although this can be reduced to 3–5% for specially prepared initial meshes.

An alternative approach is the stitching method [9]. In this approach, the problem of triangulation is split into a set of relatively simple problems of local triangulation. Firstly, in the vicinity of boundaries, body fitted local meshes are built with properties close to those regarded as being ideal. An ideal mesh is employed, away from boundaries, to fill the remaining part of the domain. These mesh fragments are then combined, to form a consistent mesh, with the outer layer of the near boundary elements stitched to a region of ideal mesh by a special procedure, in which the high compliance of mesh fragments is used. This will result in high quality meshes compared to those built by other methods [9].

5 Ideal Mesh

In 2D, a mesh of equilateral triangles is an ideal mesh. The index of every node, i.e. the number of nodes connected to a node, is 6, the Voronoi edge length $l_V = l_D/\sqrt{3} \approx 0.56 l_D$, the element quality is $q_e = 1$ and all the vertex angles are 60° . A 3D analogue of this ideal mesh consists of equal non-perfect tetrahedra, each face of which is an isosceles triangle with one side of length l_D^{long} and two shorter sides of length $l_D^{\text{short}} = (\sqrt{3}/2)l_D^{\text{long}}$. Six such tetrahedra form a parallelepiped tiling the space, as illustrated in Figure 1. It can be shown that this configuration maximises the minimum Voronoi edge for a fixed element size. All Voronoi edges have the same length $l_V \approx 0.38 \delta$ where $\delta \equiv \langle l_D \rangle = (3l_D^{\text{long}} + 4l_D^{\text{short}})/7 \approx 0.92l_D^{\text{long}}$. This configuration guarantees that the element quality is $q_e \approx 0.95$ and that every node has an index of 14. All vertices have an acute angle, 71.5° , and hence, the circumcentre is located inside each element. However, certain dihedral angles are equal to 90° , which is larger than the value $\alpha = \arccos(1/3) = 70.5^\circ$ for the perfect tetrahedron. The 2D and 3D ideal meshes satisfy the requirements listed in Section 2, but do not necessarily fit the boundaries of an arbitrary domain to be triangulated.

6 Near Boundary Triangulation

If the domain boundary is smooth enough i.e. if its radius of curvature is much greater than δ , then building the first few layers of a body fitted mesh is an elementary task in 2D: the near boundary mesh has the same topology as the 2D ideal mesh. A well tuned 2D advancing front method is capable of producing a high quality near boundary mesh of this type. Modifications to the method, which improve the mesh quality if the boundary curvature is not small, are described in [9].

The analogous 3D problem is not elementary, even for the problem of generating high quality tetrahedral elements near a plane boundary discretised with a 2D ideal mesh. Using the 3D advancing front technique, which is regarded to be an effective method for placing points, we can build a perfect tetrahedron on every boundary triangle (Figure 2a). The first layer of new points forms a hexagonal structure which cannot be connected to form triangles which are all acute. This is illustrated in Figure 3. Hence, this method applied directly, which generating nearly twice the number of boundary points, cannot even produce a mesh of the desired form for the first layer.

Analyzing the structure of the 3D ideal mesh, it can be concluded that the best location of a new point is above the edge shared by two conjugate surface triangles (Figure 2b). Starting from a plane boundary with an ideal 2D triangulation, the boundary triangles are grouped into pairs sharing the same edge. A New points are located above these edges. The optimal position for the points is at a distance of 0.684 of the boundary edge length from the

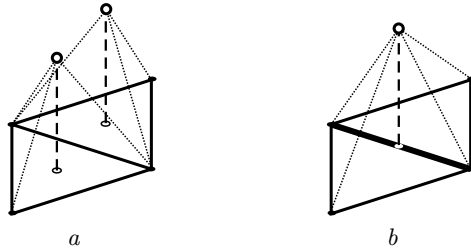


Fig. 2. (a) Placing a new point above centroid of boundary triangles as in classical AFT. (b) Placing a new point above a midpoint of an edge (bold line) shared by two boundary triangles. Open circles indicate new points, dotted lines show new edges of formed nearby boundary tetrahedra.

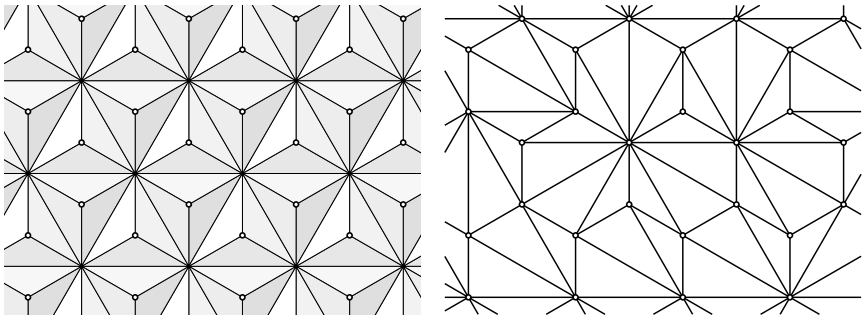


Fig. 3. Perfect tetrahedra built at every boundary triangle (view from above the boundary) (left). Delaunay connection of their apices (right)

plane. This minimises the worst element quality at $q_e = 0.795$. This procedure ensures that the number of new points similar to that of the initial boundary points. This means that the triangulation can be continued layer by layer to form a 3D mesh with the same topology as the 3D ideal mesh. This process is demonstrated in Figure 4.

In the general case, the boundary triangulation does not have the topology of 2D ideal mesh on a plane with all nodal indexes equal to 6. Therefore, when we group all the boundary faces into pairs, some of the generated triangles will remain ungrouped (Figure 5). For such an ungrouped triangle, point in the next layer must be located above its circumcentre, as in the standard advancing front technique. If the topology of the boundary mesh is good enough, i.e. if the index of most nodes is 6, and a relatively small number of nodes have index 5 and 7, then the number of uncoupled triangles is relatively small. Hence, the number of points in the the next layer will only slightly exceed the number of boundary points. All the points are connected by the Delaunay method and a new layer is formed. This can be viewed as a sophisticated

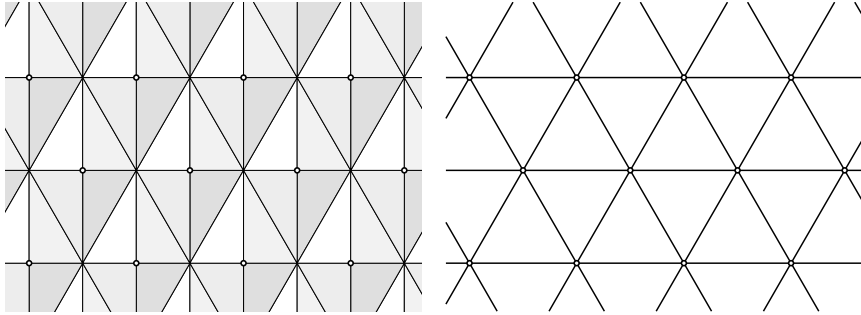


Fig. 4. Tetrahedra built if nodes are placed above edges (left). Boundary triangulation (solid) and Delaunay connection of nodes (dashed) (right).

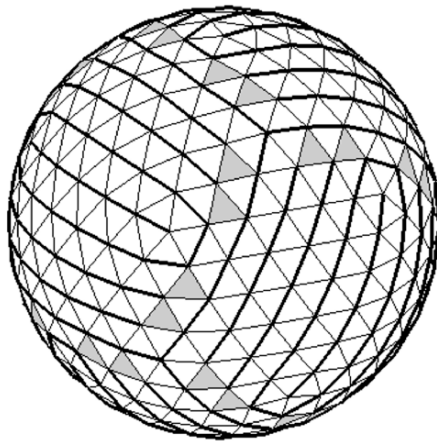


Fig. 5. Set of boundary faces is split by non-intersecting pairs of triangles sharing this same edge. Those edges are indicated by a bold lines. Non-coupled triangles are indicated by grey.

version of the 3D advancing front method for placing points, coupled with the Delaunay method for connecting points.

An example of a mesh produced by the basic AFT for a domain with a slightly curved boundary, and with a nearly ideal boundary mesh, is shown in Figure 6. Here only tetrahedra having boundary faces are shown. The colour indicates the tetrahedron quality: with white to red corresponding to quality q_e from 1 to zero and from blue to black corresponds to the quality q_e from zero to -3 . It can be observed that a significant number of bad elements remains.

Figure 7 shows several views of the first layer of a mesh generated using the proposed method. The colour–quality indication is the same as that used in

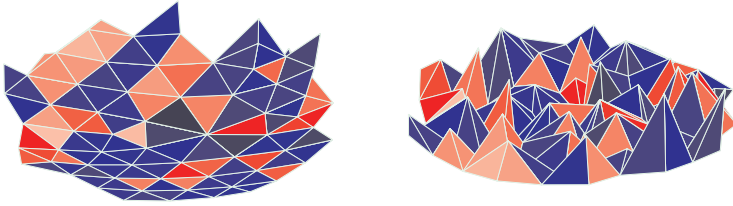


Fig. 6. Fragment of triangulation by the classical AFT. Elements having boundary face are shown only. View from the outside (left) and from the domain (right).

Figure 6. It should be noted that the near boundary triangulation is the most demanding part of the stitching method. Furthermore, acute near boundary triangulation will automatically solve the problem of boundary recovery, which is an essential part of any Delaunay triangulation.

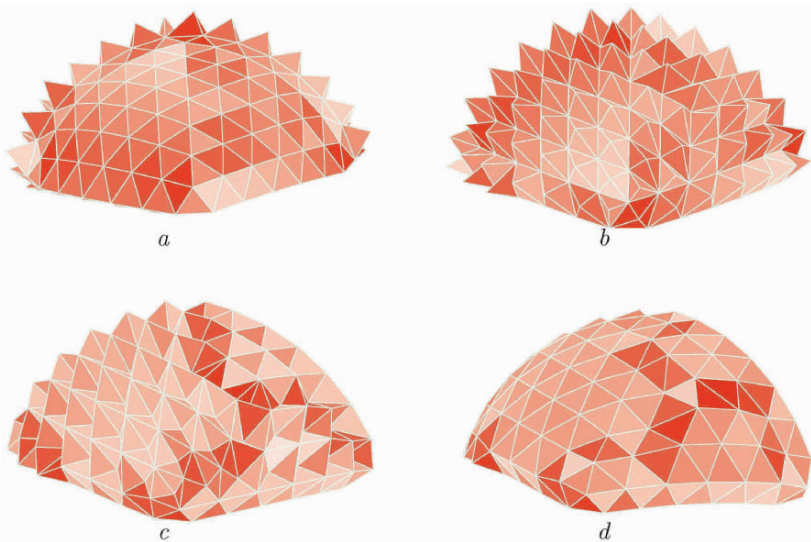


Fig. 7. Fragment of the triangulation by the new method. View from the outside (a) and from the domain (b–c), elements having boundary face (a,b), also elements having boundary edge (c), also elements having boundary node (d)

Steps in creating a new layer:

1. Split all boundary (or frontal) triangles by pair of triangles sharing the same edge, minimizing the number of single triangles.
2. Locate new points above midpoint of the shared edges and above centroid of single triangle.
3. Connect all the points by Delaunay method.

7 Example of a Domain Triangulation.

A simple domain, made of two spherical surfaces, is triangulated using the procedure described above. The size of elements used increases towards the outer sphere. A cut through the triangulation is shown in Figure 8. The colour-quality indication is the same as in Figures 6 and 7. The mesh contains some bad elements, with $q_e \approx -0.017$, but they form just 0.12% of the elements only, i.e. 256 among $N_e = 218,816$. The shortest Voronoi edge is 0.05δ , 87% of the internal nodes have the ideal index of 14, and the rest of the nodes have indices between 10 and 16. To compare the quality of the generated mesh with that obtained by other methods, the same 3D domain and the same boundary triangulation were used to generate meshes using an advancing front method, a Delaunay triangulation and a coupled Delaunay and CVT scheme. Various mesh quality criteria were computed: the global quality criteria, Q defined in equation (1); the percentage of bad elements in the mesh r_{bad} ; the individual

Table 1. Comparison for different meshing methods

Method	Q	r_{bad}	q_e		$3R_e^{\text{in}}/R_e$		nodal index	
			min	mean	min	mean	min	max
Advancing Front	$4 \cdot 10^{-7}$	67%	-3	-0.49	$3 \cdot 10^{-4}$	0.65	4	37
Delaunay	$2.4 \cdot 10^{-6}$	50%	-2.2	-0.07	$5 \cdot 10^{-3}$	0.72	8	24
Delaunay+CVT	$1.0 \cdot 10^{-5}$	9.8%	-1.2	0.39	0.08	0.88	9	21
Present method	$4.8 \cdot 10^{-2}$	0.12%	-0.02	0.65	0.74	0.92	10	16

element quality q_e , defined in equation (3); a standard criterion of the ratio an element's inradius R_e^{in} to its circumradius. To estimate topological properties of the obtained meshes, the range of the nodal index for internal nodes was also determined. The results are displayed in Table 1. It can be seen that the new method gives much improved mesh quality compared to all other methods.

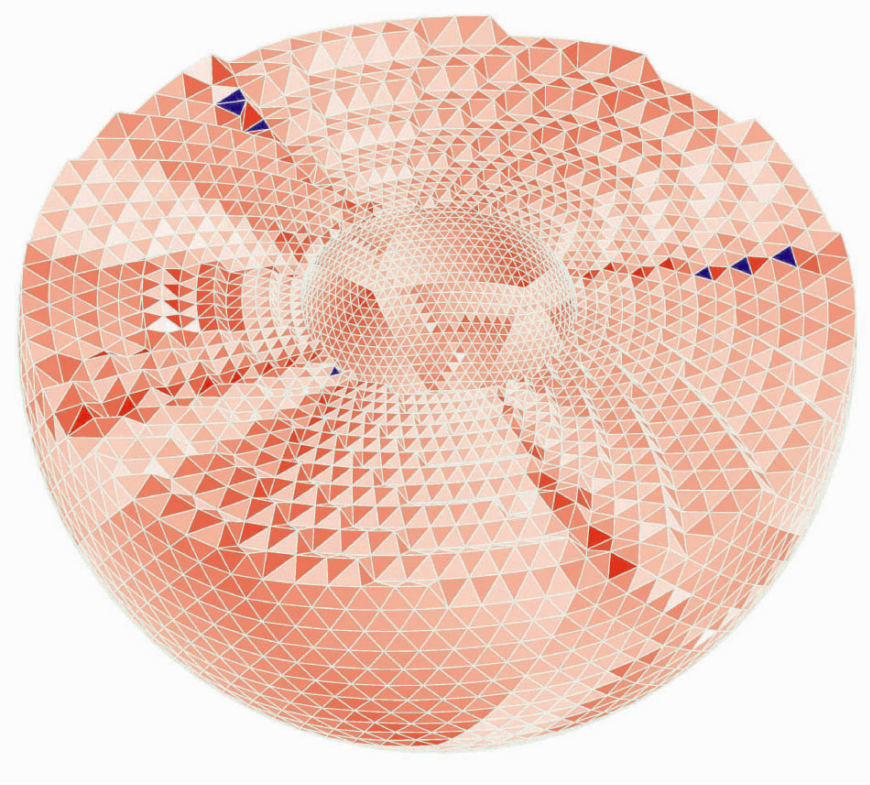


Fig. 8. Cut of the triangulation of a spherical layer.

8 Numerical Results

This mesh has been used with a co-volume integration scheme for Maxwell's equations. The unknowns are the projections of the electric field onto the Delaunay edges and the projections of the magnetic field onto the Voronoi edges. The solution is advanced in time using a staggered explicit scheme. The wave frequency is such that the diameter of the sphere is 2λ . The size of the elements near the scattering PEC sphere is approximately $\lambda/15$. Near the external boundary, the element size is $\lambda/6$. The distance between the boundaries is large enough to neglect the reflection of wave from the external boundary during first three cycles. The number of time steps per cycle is 141. The computation time per time step is 0.1 s, which is ten times faster than the corresponding time for a conventional finite element time domain method on the same mesh. The computed radar cross section (RCS) distribution is

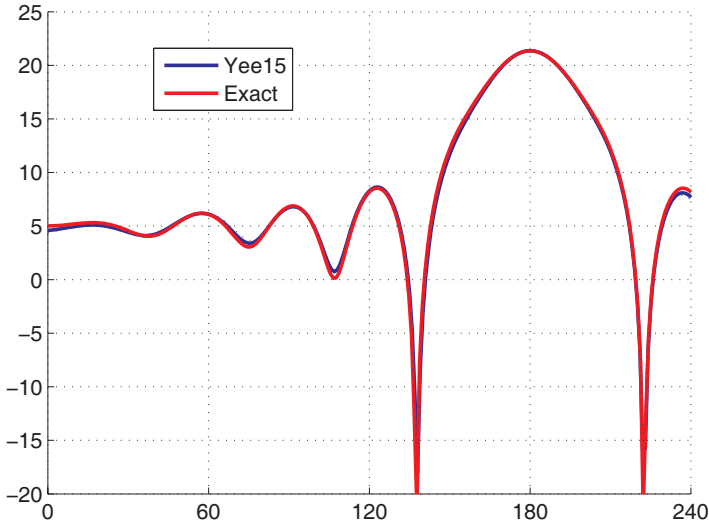


Fig. 9. Radar Cross Section exact (red) and computed by the co-volume method (blue).

compared with the exact distribution in Figure 9 and the maximum difference at any location is less than one tenth of a dB.

9 Conclusion

A new method of placing the points in 3D triangulation is proposed. The method produces a high quality near boundary triangulation. It has been demonstrated that the quality of the resulting mesh is such co-volume integration schemes can be successfully implemented. An example that demonstrates that the problem of building a mesh for a 3D domain for use with a co-volume solution scheme is solvable. Work on the generalization of the method for more complicated domains is currently in progress.

References

1. Yee K (1996) Numerical solution of initial boundary value problem involving Maxwell's equation in isotropic media, *IEEE Trans Antennas and Propagation* 14:302–307
2. Churbanov A (2001) An unified algorithm to predict compressible and incompressible flows. In: *CD Proceedings of the ECCOMAS Computational Fluid Dynamics Conference*, Swansea, Wales, UK

3. Morgan K, Hassan O, Peraire J (1996) A time domain unstructured grid approach to the simulation of electromagnetic scattering in piecewise homogeneous media, *Computer Methods in Applied Mechanics and Engineering* 134:17–36
4. Bern M, Chew L.P, Eppstein D, and Ruppert J (1995) Dihedral bounds for mesh generation in high dimensions. In: 6th ACM-SIAM Symposium on Discrete Algorithms, 189–196.
5. Peraire J, Vahdati M, Morgan K, Zienkiewicz O (1987) Adaptive remeshing for compressible flow computations, *Journal of Computational Physics* 72:449-466
6. George P, Borouchaki H (1998) *Delaunay Triangulation and Meshing. Application to Finite Elements*, Hermès, Paris
7. Marcum D, Weatherill N (1995) Aerospace applications of solution adaptive finite element analysis, *Computer Aided Geometric Design*, Elsevier, 12:709–731
8. Frey W, Field D (1991) Mesh relaxation: a new technique for improving triangulation, *Int J Numer Meth in Engineering* 31:1121–1133
9. Sazonov I, Wang D, Hassan O, Morgan K, Weatherill N (2006) A stitching method for the generation of unstructured meshes for use with co-volume solution techniques, *Computer Methods in Applied Mechanics and Engineering* 195/13-16:1826–1845
10. Du Q, Faber V, Gunzburger M (1999) Centroidal Voronoï Tessellations: Applications and Algorithms, *SIAM Review* 41/4:637–676
11. Lloyd S (1982) Least square quantization in PCM. *IEEE Trans Infor Theory* 28:129–137
12. Taflove A, Hagness S.C (2000) *Computational electrodynamics: the finite-difference time domain method*. 2nd ed., Artech House, Boston.

Appendix: Yee’s scheme generalized to unstructured meshes

The Yee algorithm is based on: Ampere’s law

$$\frac{\partial}{\partial t} \int_A \mathbf{E} d\mathbf{A} = \varepsilon \oint_{\partial A} \mathbf{H} dl \quad (4)$$

and Faraday’s law

$$\frac{\partial}{\partial t} \int_A \mathbf{H} d\mathbf{A} = -\mu \oint_{\partial A} \mathbf{E} dl \quad (5)$$

applied to a surface A and its boundary ∂A . Here \mathbf{E} and \mathbf{H} are the electric and magnetic fields, respectively; $d\mathbf{A}$ is an element of surface area directed normal to the surface, dl is an element of the contour length directed tangent to the contour; ε and μ are the electric and magnetic permeability.

If a dual Delaunay–Voronoi diagram is built for the domain of integration, then equations (4) and (5) can be approximated as

$$\frac{E_i^n - E_i^{n-1}}{\Delta t} A_i^V = \varepsilon \sum_{k=1}^{M_i^V} H_{j_i,k}^{n+0.5} l_{j_i,k}^V, \quad i = 1, \dots, N_s^D \quad (6)$$

$$\frac{H_j^{n+0.5} - H_j^{n-0.5}}{\Delta t} A_j^D = -\mu \sum_{k=1}^3 E_{i_{j,k}}^n l_{i_{j,k}}^D, \quad j = 1, \dots, N_s^V \quad (7)$$

where E_i^n is the projection of the electric field onto the i th Delaunay side at the instant $n\Delta t$; $H_j^{n+0.5}$ is the projection of the magnetic field onto the j th Voronoi side at the instant $(n + 0.5)\Delta t$; l_i^D and A_i^V denote the length of the i th Delaunay side and the area of the corresponding Voronoi face respectively; l_j^V and A_j^D are the length of the j th Voronoi side and the area of the corresponding Delaunay face respectively; $j_{i,k}, k = 1, \dots, M_i^V$ are sides of a Voronoi face corresponding to the i th Delaunay edge (Figure 10a); $i_{j,k}, k = 1, 2, 3$ are sides of a Delaunay triangle face corresponding to the j th Voronoi edge (Figure 10b); N_s^D and N_s^e are the numbers of Delaunay and Voronoi sides in the mesh.

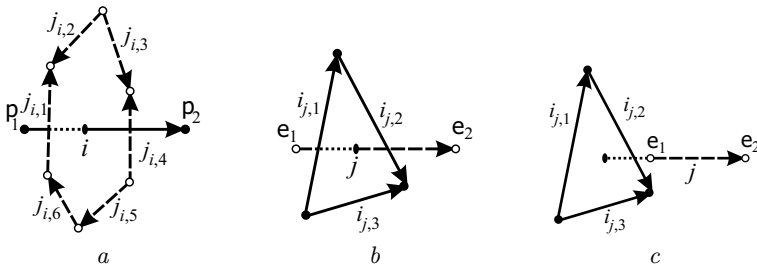


Fig. 10. The i th Delaunay side connecting nodes p_1 - p_2 and correspondent Voronoi face formed by Voronoi sides $j_{i,1}, \dots, j_{i,6}$ (a). The j th Voronoi side connecting circumcentra e_1 - e_2 and correspondent Delaunay face formed by Delaunay sides $i_{j,1}, i_{j,2}, i_{j,3}$ (b-c). In (c) the Voronoi side does not intersect the correspondent Delaunay face.

Equations (6) and (7) form an explicit procedure for advancing the electric field from time t^n to $t^{(n+1)}$ and the magnetic field from time $t^{(n-0.5)}$ to $t^{(n+0.5)}$. For a structured grid this scheme is if $c\Delta t < l/\sqrt{3}$ where $c = 1/\sqrt{\epsilon\mu}$ is the light speed, and l is an edge length. For an unstructured tetrahedral mesh, there is no such simple criterion but computations show that we can use the following relation

$$c\Delta t < S_f \min_{i,j} \{l_i^V, l_j^D\} \quad (8)$$

where S_f is a safety factor.

In the co-volume scheme, the values of electric and magnetic field are taken at the intersection point of the edge and the correspondent face. If an element has its circumcentre outside its volume then the field will lie outside the edge connecting the two corresponding circumcentres (Figure 10c). In this case, the approximation of the integral cannot guarantee even the first order accuracy.

A Study on Delaunay Terminal Edge Method

Maria-Cecilia Rivara

Department of Computer Science, Universidad de Chile
mcrivara@dcc.uchile.cl

Summary. The Delaunay terminal edge algorithm for triangulation improvement proceeds by iterative Lepp selection of a point M which is midpoint of a Delaunay terminal edge in the mesh. The longest edge bisection of the associated terminal triangles (sharing the terminal edge) can be seen as a first step in the Delaunay insertion of M . The method was introduced as a generalization of non-Delaunay longest edge algorithms but formal termination proof had not been stated until now. In this paper termination is proved and several geometric aspects of the algorithm behavior are studied.

1 Introduction

Edge based refinement methods for 2 and 3 dimensional triangulations have been studied and used for developing adaptive software for PDEs since beginning of the eighties, e.g. Bank [5], Rivara et al [6, 7, 18, 8, 13], Nambiar et al, [11], Muthukrishnan et al [10], Morin et al, [9]. In particular, the methods based on the longest edge bisection of triangles guarantee that refined meshes of geometrical quality analogous to the input mesh are produced in 2-dimensions [6, 7].

More recently, methods that produce a sequence of improved constrained Delaunay triangulations (CDT) have been developed to deal with the quality triangulation problem of a planar straight line graph D . The combination of edge refinement and Delaunay insertion has been described by George and Borouchaki, [4, 3] and Rivara and her collaborators, [13, 15, 16, 17, 18, 22]. Strong mesh improvement properties for iterative Delaunay refinement based on inserting the circumcentre of triangles to be refined have been established by Chew, [2], Ruppert [14], and Shewchuk [21]. In particular, under appropriate conditions on D , the method is guaranteed to produce optimal-size meshes with the minimum angle larger than a specified angle tolerance. Applications of this form of refinement have been described by Weatherill et al [25, 26] and Baker, [19]. Baker also published a comparison of edge based and circumcenter

based refinement, [20]. An algorithm based on off-center insertions have been recently presented by Üngör [28].

The Delaunay terminal edge algorithm for triangulation improvement proceeds by iterative selection of a point M which is midpoint of a Delaunay terminal edge (a longest edge for both triangles that share this edge) in the mesh. This method combines the use of the longest edge propagating path associated to a bad quality processing triangle to determine a terminal edge in the current mesh, with the Delaunay insertion of the midpoint of this terminal edge. Note that the longest edge bisection of the associated terminal triangles (sharing the terminal edge) can be seen as a first step in the Delaunay insertion of M .

The Lepp Delaunay terminal edge method was introduced in a rather intuitive basis as a generalization of previous longest edge algorithms in [13, 18, 31]. This was supported by the improvement properties of both the longest edge bisection of triangles and the Delaunay algorithm, and by the result presented in Theorem 1 in next section. Later in [17] we discussed some geometrical properties including some (rare) potential loop cases for angle tolerance greater than 22° and its management. However, while empirical studies show that the method behaves analogously to the circumcircle method in 2-dimensions [17, 18, 22], formal proofs on algorithm termination and on optimal size property have not been established due to the difficulty of the analysis. Recently in [29] we have presented some geometrical improvement properties of an isolated insertion of a terminal edge midpoint M in the mesh. In this paper we improve and extend these results and prove algorithm termination.

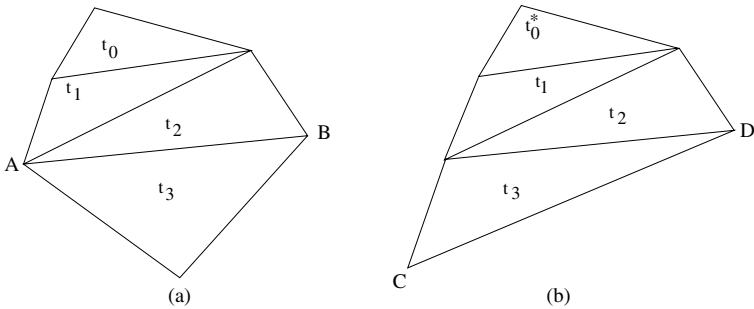


Fig. 1. (a) AB is an interior terminal edge shared by terminal triangles (t_2, t_3) associated to $\text{Lepp}(t_0) = \{t_0, t_1, t_2\}$; (b) CD is a boundary terminal edge with unique terminal triangle t_2 associated to $\text{Lepp}(t_0^* = \{t_0^*, t_1, t_2, t_3\})$.

2 Concepts and Preliminary Results

2.1 The Longest Edge Propagating Path (Lepp) and Terminal Edge Triangles

The following concepts were introduced and used in references [13, 17, 18]. An edge E is called a terminal edge in triangulation τ if E is the longest edge of every triangle that shares E , while the triangles that share E are called terminal triangles. Note that in 2-dimensions either E is shared by two terminal triangles t_1, t_2 if E is an interior edge, or E is shared by a single terminal triangle t_1 if E is a boundary (constrained) edge. See Figure 1 where edge AB is an interior terminal edge shared by two terminal triangles t_2, t_3 , while edge CD is a boundary terminal edge with associated terminal triangle t_3 .

For any triangle t_0 in τ , the longest edge propagating path of t_0 , called $Lepp(t_0)$, is the ordered sequence $\{t_j\}_{j=0}^{N+1}$, where t_j is the neighbor triangle on a longest edge of t_{j-1} , and longest-edge(t_j) > longest-edge(t_{j-1}), for $j=1, \dots, N$. Edge $E = \text{longest-edge}(t_{N+1}) = \text{longest-edge}(t_N)$ is a terminal edge in τ and this condition determines N . Consequently either E is shared by a couple of terminal triangles (t_N, t_{N+1}) if E is an interior edge in τ , or E is shared by a unique terminal triangle t_N with boundary (constrained) longest edge. See Figure 1 for an illustration of these ideas.

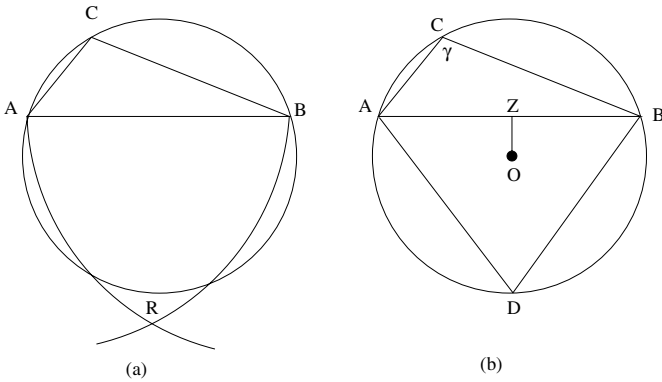


Fig. 2. R is the geometrical place of the fourth vertex D for Delaunay terminal triangles ABC, ABD; (b) R reduces to one point when $\gamma = 2\pi/3$ (triangle ADB equilateral).

For a Delaunay mesh, an unconstrained terminal edge imposes the following constraint on the largest angles of the associated terminal triangles [13, 18]:

Theorem 1. For any pair of Delaunay terminal-triangles t_1, t_2 sharing a non-constrained terminal edge, largest angle(t_i) $\leq 2\pi/3$ for $i = 1, 2$.

Proof For any Delaunay terminal triangles BAC of longest AB (see Figure 2(a)), the third vertex D of the neighbor terminal triangle ABD must be situated in the exterior of circumcircle CC(BAC) and inside the circles of center A,B and radius $|A - B|$. This defines a geometrical place R for D which reduces to one point when angle $BCA = 2\pi/3$ where $OZ = r/2$ (see Figure 2(b)), implying that $R = \phi$ when angle $BCA > 2\pi/3$.

2.2 The Longest Edge Bisection of Triangles

For an arbitrary triangle, t , the longest edge bisection of t is the splitting of t into two triangles t_A, t_B by joining the midpoint of a longest edge to the opposite vertex as shown in Figure 3 where $|B - C| \leq |C - A| \leq |B - A|$. Using the notation of Figure 3, the following theorem presents some simple properties of a the first longest edge bisection of any t .

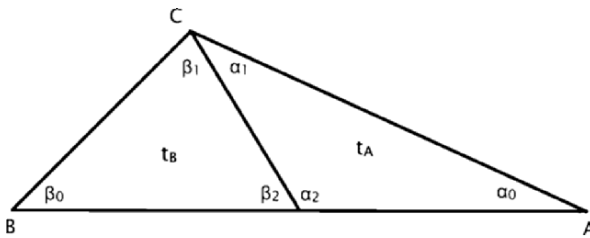


Fig. 3. notation for longest edge bisection.

- Theorem 2.** a) $\alpha_1 \geq \alpha_0/2$
 b) If t is obtuse, $\alpha_1 \geq \alpha_0$
 c) $\beta_2 \geq 3\alpha_0/2$
 d) if t is obtuse, $\beta_2 \geq 2\alpha_0$
 e) if t is acute, $\beta_1 > \alpha_0$ if $\alpha_0 < \pi/6$
 f) t_B is acute if $\alpha_0 > \arcsin(1/3) = 19.5^\circ$ (sufficient, but not necessary.)

Assertion a) follows from the following strong result due to Rosenberg and Stenger [27]: For any triangle t^* obtained in the iterative longest edge bisection process, the smallest angle of t^* is greater than or equal to $\alpha_0/2$. Assertion (b) was proved in [12]. Assertions c) and d) are obtained by noting that $\beta_2 = \alpha_1 + \alpha_0$. Assertions (e) and (f) were proved in [29].

2.3 Delaunay Insertion of Point M

It is well known that among all triangulations, the Delaunay triangulation maximizes the minimum angle. In fact, a much stronger statement holds [30].

Among all triangulations with the same smallest angle, the Delaunay triangulation maximizes the second smallest angle, and so on. In particular, we can associate an angle vector to any triangulation, that is the increasing sequence of angles $(\alpha_1, \alpha_2, \dots, \alpha_{3m})$ (where m is the total number of triangles in any triangulation of the associated vertex set). That is, $\alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_{3m}$. We say $(\alpha_1, \alpha_2, \dots, \alpha_{3m}) > (\beta_1, \beta_2, \dots, \beta_{3m})$ if there exists an i such that $\alpha_i > \beta_i$ and for all $j < i, \alpha_j = \beta_j$. This is the standard lexicographic order on vectors. So, given 2 triangulations, we can compare their angle vectors using this order [30].

Theorem 3. *Among all triangulations of a given point set, the Delaunay triangulation has the largest angle vector.*

Consequently if we call T_S the non-Delaunay triangulation obtained by simple insertion of point M (longest edge bisection of both terminal triangles), and T_D to the Delaunay triangulation obtained by Delaunay insertion of M , then T_D is better than T_S in sense of Theorem 3.

3 An Algorithmic Description of the Lepp Delaunay Terminal Edge Method

We are interested in improvement of angles in the mesh, so we introduce a minimum angle tolerance θ_{tol} . Given these criteria, we will refer to any triangle t such that a smallest angle of t is less than θ_{tol} as a bad triangle.

The algorithm can be simply described as follows: iteratively, each bad triangle t_{bad} in the current triangulation is eliminated by finding Lepp(t_{bad}), a pair of terminal triangles t_1, t_2 , and associated terminal edge l . If non-constrained edges are involved, then the midpoint M of l is Delaunay inserted in the mesh. Otherwise the constrained point insertion criterion of § 3.1 is used. The process is repeated until t_{bad} is destroyed in the mesh, and the algorithm finishes when the minimum angle in the mesh is greater than or equal to an angle tolerance θ_{tol} . The algorithm including point selection strategies for constrained terminal triangles is given below. A restriction on the size of θ_{tol} will be presented in § 3.1.

Lepp-Delaunay-Terminal-Edge Algorithm

```

Input = a CDT,  $\tau$ , and angle tolerance  $\theta_{tol}$ 
Find  $S_{bad}$  = the set of bad triangles with respect to  $\theta_{tol}$ 
for each  $t$  in  $S_{bad}$  do
  while  $t$  remains in  $\tau$  do
    Find Lepp ( $t_{bad}$ ), terminal triangles  $t_1, t_2$  and terminal edge  $l$ . Triangle
     $t_2$  can be null for boundary  $l$ .
    Select Point ( $P, t_1, t_2, l$ )
    Perform constrained Delaunay insertion of  $P$  into  $\tau$ 
    Update  $S_{bad}$ 

```



```

    end while
end for

Select Point (P,  $t_{term1}$ ,  $t_{term2}$ ,  $l_{term}$ )
if (second longest edge of  $t_{term1}$  is not constrained and second longest edge
of  $t_{term2}$  is not constrained) or  $l_{term}$  is constrained then
    Select P = midpoint of  $l_{term}$  and return
else
    for j = 1,2 do
        if  $t_{termj}$  is not null and has constrained second longest edge  $l^*$  then
            if ( $\alpha_0 < 30^\circ$  or  $\alpha_1 < 30^\circ$ ) then
                Select P = midpoint of  $l^*$  and return
            else
                Select P = midpoint of  $l_{term}$ 
            end if
        end if
    end for
end if
end if

```

3.1 Potential Loop Conditions to be Avoided

When the second longest edge CA is a constrained edge, the swapping of this edge is forbidden. In such a case, the insertion of point M would imply that the later processing of bad quality triangle MAC would introduce triangle MAM_1 (see Figure 4(a)) similar to triangle ABC implying an infinite loop situation. To avoid this behavior we introduce the following additional operation:

Constrained edge point insertion: If CA is a constrained edge and $\alpha_0 < 30^\circ$ or $\alpha_1 < 30^\circ$ then insert midpoint M_1 of edge CA.

Lemma 1. *If triangle BM_1A (see Figure 4(b)) has largest angle greater than $2\pi/3$, then M is not inserted in the mesh by processing triangle M_1BA .*

Proof See Theorem 1□

The condition ($\alpha_0 < 30^\circ$ or $\alpha_1 < 30^\circ$) has been introduced to avoid the undesirable boundary loop condition, as well as to avoid the insertion of unnecessary vertices in the boundary. We have chosen 30° value because the algorithm behaves well in practice for $\theta_{tol} = 30^\circ$ [13, 17, 18]

For simplicity we will omit consideration of a rare special loop case discussed in [17], where a triangle MAM_1 similar to a bad-quality triangle ABC can be also obtained for a non-constrained edge CA. This happens when quadrilaterals BEAC and ADCM (see Figure 4(a)) are terminal quadrilaterals (where edges BA and CA are terminal edges respectively) together with some non-frequent conditions on neighbor constrained items. A necessary but not sufficient condition on the triangle ABC for this to happen is that angle $BMC \geq \pi/3$ which implies $\alpha_0 \geq \alpha_{limit} = \arctan \frac{\sqrt{15}-\sqrt{3}}{3+\sqrt{5}} > 22^\circ$ for obtuse

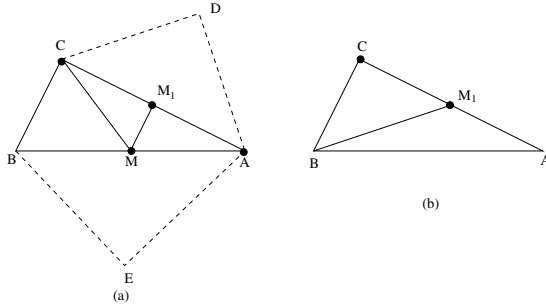


Fig. 4. (a) Over constrained edge CA, the insertion of M and M_1 produces triangle MAM_1 similar to triangle BAC; (b) Insertion of M_1 avoids this situation.

triangle BAC [17]. This loop case can be avoided by adding some extra conditions to the algorithm. However, to simplify the analysis we will restrict the angle tolerance to α_{limit} .

4 Characterization of Delaunay Terminal Triangles

In this section we present a characterization of Delaunay terminal triangles based on fixing the second longest edge CA and choosing the smallest angle at vertex A.

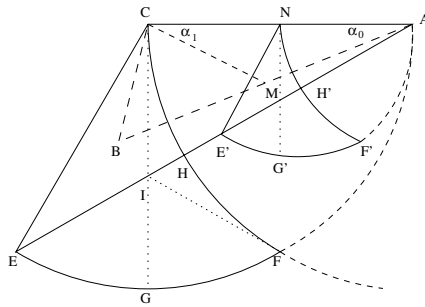


Fig. 5. Regions EFC and $E'F'C'$ are geometrical places for vertex B and midpoint M for a terminal triangle BAC with respective smallest and largest angle of vertices A and C.

Figure 5 shows the possible locations for vertex B and the midpoint M. This supports an analysis of properties of the longest edge sons of triangle BAC as parametrized by B. Since CB is a shortest edge and BA is a longest edge, the following two conditions hold: Condition (1) constrains B to lie inside the circular arc EFA of centre C and radius $|C - A|$. Consequently,

M lies inside the circular arc $CF'A$ of centre $N = (C + A)/2$ and radius $|C - A|/2$. Condition (2) constrains B to lie outside the circular arc CF of centre A and radius $|C - A|$, and so M lies outside circular arc NF' of centre A , radius $|C - A|/2$. The line CE makes an angle of 120° with CA ; this constraint is a direct application of Theorem 1 for an unconstrained Delaunay terminal triangle. Some of the properties of the triangles of the diagram are summarized in the table given below. We denote the smallest and largest angle of any triangle t by $\theta_{min}(t)$ and $\theta_{max}(t)$ respectively.

B is in/on	property
edge CE	$\theta_{max}(t) = 120^\circ$
arc EF	t is an isosceles triangle with smallest edge equal to second longest edge
arc CF	t is an isosceles triangles with longest edge equal to second longest edge
edge CG	t is a right triangle, $\alpha_0 = \alpha_1$, $\theta_{min}(t_A) = \alpha_0 = \alpha_1$
interior of region CEG	obtuse triangles, $\alpha_1 > \alpha_0$, $\theta_{min}(t_A) = \alpha_0$
interior of region CGF	acute triangles with $\alpha_1 < \alpha_0$, $\theta_{min}(t_A) = \alpha_1$

5 Angle Size Bounds

In this section we use the characterization of the preceding section to obtain a theorem that improves the bounds of Theorem 2 on α_1 for acute Delaunay terminal triangles. We firstly present a study on the distribution of the ordered pair of angles (α_0, α_1) for unconstrained Delaunay terminal triangles and we develop better bounds on α_1 for acute triangles. The study on the distribution of (α_0, α_1) is summarized in Figure 6 which is a relabelled version of Figure 5.

Note that the segments EH and UW respectively correspond to the terminal triangles with smallest angles equal to 30° and $\alpha_{limit} \approx 22,2^\circ$. Note that for $\alpha_0 = 30^\circ$, the angle α_1 decreases from 60° to approximately 23.79° along EH, while for $\alpha_0 = \alpha_{limit}$ the α_1 angle decreases approximately from 37.75° to 19.25° along UW. Remember that segment line CG identifies right terminal triangles with $\alpha_0 = \alpha_1$. Note that the ratio $\alpha_1/\alpha_0 \geq 1$ for obtuse triangles (B in CEG) as expected according to part (b) of Theorem 2; while for acute triangles, the ratio α_1/α_0 increases from 0.5 to 1 both along arc F to C, and along arc F to G. These properties and continuity reasoning allows to state the following lemma:

Lemma 2. *For acute Delaunay terminal triangles, there exist fixed constants C_1, C_2 ($C_1 \approx 0.79, C_2 \approx 0.886$) such that:*

- a) *For smallest angle $\alpha_0 \leq 30^\circ$ (B in region CIH), $\alpha_1 \geq C_1\alpha_0$.*
- b) *For smallest angle $\alpha_0 \leq \alpha_{limit}$ (B in region CVW), $\alpha_1 \geq C_2\alpha_0$.*

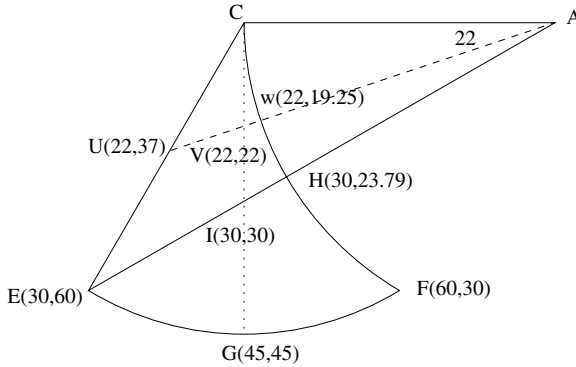


Fig. 6. Distribution of angles (α_0, α_1) for a terminal triangle with B in CEF.

c) The ratio α_1/α_0 approaches 1.0 both when α_0 decreases, and when BAC becomes a right triangle.

d) Using the notation of Figure 3, $\beta_2 \geq (1 + C_1)\alpha_0$ for $\alpha_0 \leq 30^\circ$.

Proof for (d) note that $\beta_2 = \alpha_1 + \alpha_0$ □

6 Bounds on the Distance from M to Previous Vertices

In this section we bound the distance from M to previous vertices in the mesh for a bad quality terminal triangle BAC. We use both the properties of the longest edge bisection of a Delaunay terminal triangle BAC and the constraint on the empty circumcircle.

Note that the circumcenter O of an obtuse (acute) triangle is situated in the exterior (the interior) of the triangle, as illustrated in Figure 7. Furthermore for any non constrained Delaunay obtuse triangle t, the distance d from the circumcenter O to the longest edge BA (see Figure 7(a)) satisfies that $0 < d < r/2$, where r is the circumradius. We will consider the limit cases $d = r/2$ and $d = 0$, which respectively correspond to largest angles equal to $2\pi/3$ and $\pi/2$, to state bounds for obtuse and acute triangles.

6.1 Obtuse Delaunay Terminal Triangles

Consider the limit Delaunay terminal triangle BAC with largest angle equal to $2\pi/3$ and smallest angle α equal to $\alpha_{limit} = \arctan \frac{\sqrt{15}-\sqrt{3}}{3+\sqrt{5}} > 22.2^\circ$ which holds if and only if angle BMC is equal to $\pi/3$.

Consider Figure 8 and without loss of generality, assume $r = 1$. Then by using a coordinate system of center O and axes OZ and $O\tilde{Z}$, the point $C(x_C, y_C)$ satisfies the equations $y_C = mx_C + 1/2$ and $x_C^2 + y_C^2 = 1$, where $m = \tan 120^\circ = -\sqrt{3}$. By solving this system we get:

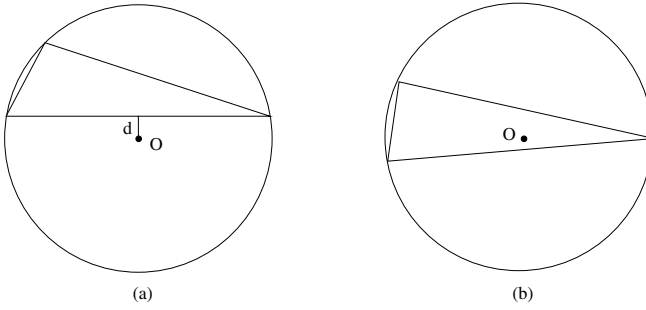


Fig. 7. (a) obtuse triangle; (b) acute triangle.

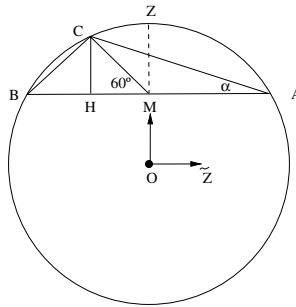


Fig. 8. BAC is a limit triangle with angle $BCA = 3\pi/2$, $\alpha = \alpha_{limit} \approx 22^\circ$, angle $BMC = \pi/3$, and $OM = r/2$.

$$x_C = \frac{\sqrt{3}-\sqrt{15}}{8}, \quad y_C = -\sqrt{3} \left(\frac{\sqrt{3}-\sqrt{15}}{8} \right) + \frac{1}{2}.$$

This implies that $\alpha_{limit} = \arctan \left(\frac{\sqrt{15}-\sqrt{3}}{3+\sqrt{5}} \right)$ (as discussed in §3.1) and $OM = MZ = 1/2$

$$BA = \sqrt{3}, \quad BM = MA = \sqrt{3}/2$$

$$HM = (\sqrt{15} - \sqrt{3})/8$$

$$BH = (5\sqrt{3} - \sqrt{15})/8$$

$$CH = \sqrt{3}(\sqrt{15} - \sqrt{3})/8$$

$$BC = \sqrt{3}\sqrt{3 - \sqrt{5}}/2$$

$$MC = (\sqrt{15} - \sqrt{3})/4$$

Note also that any previous neighbor vertex D of neighbor triangle CAD must be situated in the exterior of $CC(BAC)$, which implies $|D - M| > |Z - M|$.

Lemma 3. For any obtuse bad quality Delaunay terminal triangle,

$$|M - C| \geq C_1|B - C|, \quad \text{and} \quad |M - D| \geq C_2|B - C|$$

$$\text{where } C_1 = \frac{\sqrt{15}-\sqrt{3}}{2\sqrt{3}\sqrt{3-\sqrt{5}}} \approx 0.7 \quad \text{and} \quad C_2 = \frac{1}{\sqrt{3}\sqrt{3-\sqrt{5}}} \approx 0.66$$

Lemma 4. For $\alpha \leq \alpha_{limit}$ and $\pi/2 \leq \gamma \leq 2\pi/3$, there exist functions $\tilde{C}_1(\alpha, \gamma), \tilde{C}_2(\alpha, \gamma)$ such that

$$|M - C| = \tilde{C}_1(\alpha, \gamma)|B - C| \text{ and } |M - D| = \tilde{C}_2(\alpha, \gamma)|B - C|$$

where $\tilde{C}_1(\alpha, \gamma) \geq C_1$, and $\tilde{C}_2(\alpha, \gamma) \geq C_2$

Furthermore,

- (i) for fixed $\alpha = \tilde{\alpha}$, both $\tilde{C}_1(\tilde{\alpha}, \gamma)$ and $\tilde{C}_2(\tilde{\alpha}, \gamma)$ increase as γ decreases.
- (ii) For fixed $\gamma = \tilde{\gamma}$, both $C_1(\alpha, \tilde{\gamma})$ and $\tilde{C}_2(\alpha, \tilde{\gamma})$ increase as α decreases.

6.2 Acute Delaunay Terminal Triangles

Consider the right triangle with smallest angle equal to α_{limit} , where M coincides with the circumcenter O . Assuming $r = 1$, it holds that

$$|B - C| = 2\sin \alpha_{limit} \text{ and } |MC| = |MZ| = 1, \text{ which implies that}$$

$$|M - C| = |M - Z| = \frac{1}{2 \sin \alpha_{limit}}|B - C| \approx 1.3|B - C|.$$

Furthermore for acute isosceles triangle with smallest angle equal to α_{limit} , it holds that $|M - C| > |M - B|$ and $|M - Z| > r$. Assuming $r = 1$, after some computation we get $|B - C| < 0.4$ and $|M - B| > 0.98$ which implies $|M - C| > 2.4|B - C|$, and $|M - Z| > 2.5|B - C|$. Consequently the following theorem can be stated:

Theorem 4. For acute Delaunay terminal triangles, there exist constants $C_C > 1, C_D > 1$ such that

$$|M - C| \geq C_C|B - C| \text{ and } |M - D| \geq C_D|B - C|.$$

7 Elimination of Too-Obtuse Triangles

7.1 Elimination of Too Obtuse Triangle MAC

Let triangle $t_A = MAC$ be the obtuse triangle resulting from bisecting terminal edge BA of triangle BAC , and assume that MAC remains in the mesh after the Delaunay insertion of M . We consider the case in which the largest angle of triangle MAC is greater than $2\pi/3$, in which case we refer to MAC as a too-obtuse triangle. See Figure 9. According to Theorem 1, MAC can not become a terminal triangle in the mesh and will be necessarily eliminated by swapping of edge CA , when another terminal edge midpoint M_1 , is inserted in a neighbor terminal edge E associated to $Lepp(MAC)$. This can be performed by processing either triangle MAC or another bad triangle t^* such that $Lepp(t^*)$ intersects $Lepp(MAC)$.

Consider the case shown in Figure 9, in which the neighboring triangle of MAC on edge AC is triangle ACD and terminal edge E is either AD or CD (in Figure 9, D is denoted as D' when E is equal to CD). Then the following constraints apply to M_1 :

- (i) D must be in the exterior of $CC(MAC)$
- (ii) M_1 must be in the interior of $CC(MAC)$ in order that the swapping of edge CA applies.
- (iii) Theorem 1 applies, implying that angle $DCA \leq 2\pi/3$ (or angle $D'AC \leq 2\pi/3$), which restricts M_1 to the shadowed region of Figure 9.

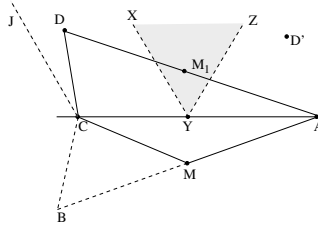


Fig. 9. DA (or $D'C$) terminal edge implies that M_1 must belong to region XYZ where angle $JCA =$ angle $XYA =$ angle $CYZ = 2\pi/3$.

The following Lemma presents some useful geometrical properties to state angle bounds on the smallest angle of neighbor terminal triangle DAC as a function of the size of angle CMA .

Lemma 5. Consider triangle MAC and the line JC such that angle $JCA=2\pi/3$ as shown in Figure 10. Then

- (a) If angle $CMA = 120^\circ$, then JC is tangent to $CC(MAC)$ at C (see Figure 10(a));
- (b) If angle $CMA > 120^\circ$, then JC intersects $CC(MAC)$ at a point G (see Figure 10(b));
- (c) If angle $CMA = 150^\circ$, then the triangle GCA is isosceles with angles $30^\circ, 30^\circ, 120^\circ$;
- (d) If angle $CMA \geq 150^\circ$, the angle $CAD \geq 30^\circ$ and the smallest angle of triangle CAD is always at D .

Remark Note that according to Theorem 2 and Lemma 2, triangle MAC is almost isosceles. This implies that for the limit case where angle $CMA = 120^\circ$, the smallest angle of triangle MAC is greater than or equal to α_{limit}

Lemma 6. Consider triangle MAC having neighbor terminal edge AD of midpoint M_1 . If angle $CMA > 120^\circ$, then M_1 belongs to region $KG'UV$ in Figure 11, where angle $KLA = 120^\circ$, $VL \perp CA$, and U is midpoint of VA . Furthermore angle $CAD \geq$ angle CAG .

Proof Part (b) of Lemma 5 applies. Thus D must be situated in the exterior of $CC(MAC)$ and to the right of line JG in Figure 10(b), which allows to find the region for M_1 □

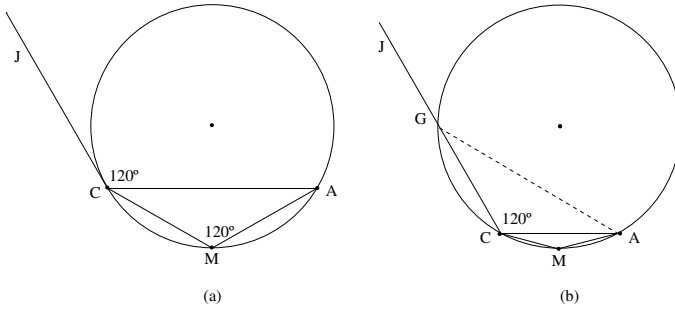


Fig. 10. (a) angle $CMA = 120^\circ$ implies JC is tangent to $CC(MAC)$ at C ; (b) angle $CMA > 120^\circ$ implies that JC intersects $CC(MAC)$ at a point G .

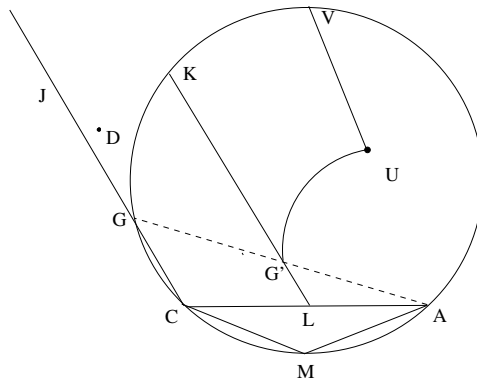


Fig. 11. Largest angle of triangle CMA impose constraints on neighbor terminal triangle DCA ; D in region $JGKV$ by above $CC(MAC)$ and to the right of JG , and M_1 in $KG'UV$.

Figure 12 shows the case in which FF' is a terminal edge. Here the constraints on M_1 also assure that points unnecessarily close to midpoint M and to previous points A, C, F, F' are not inserted.

7.2 Too-Obtuse Triangles of the Initial Mesh

Lemma 5 and Lemma 6 are indeed valid for any non-constrained too-obtuse triangle in a CDT, and consequently the results of the preceding section also apply to the triangles of the initial mesh. Note however that the remark which is below Lemma 5 does not apply, and an initial too-obtuse triangle can have a small smallest-edge corresponding to a local feature size of the geometry.

Consider now the case where a sequence of non-constrained too-obtuse triangles appear in the initial CDT. This represents an initial configuration which ends either with a constrained edge (this is mandatory if the last triangle is too-obtuse) or with a couple of terminal triangles. In any case the insertion

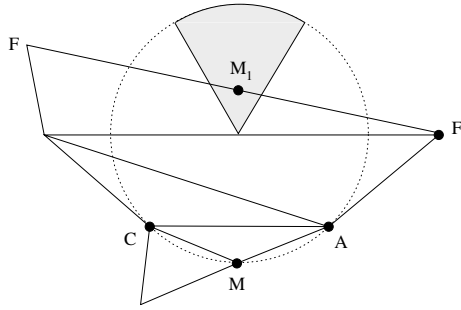


Fig. 12. FF' is terminal edge in $\text{Lepp}(CMA)$ implying that midpoint M_1 belongs to the shadowed region.

of a terminal edge midpoint M in general implies the big improvement of a subsequence of too-obtuse triangles, where the circumcircle of each triangle of the subsequence contains M .

8 Improvement Steps

The results of last section guarantee that M_1 is not inserted near to previous vertices A, C , neither too close to the midpoint of CA . Furthermore the Delaunay insertion of M_1 eliminates triangle CMA by introducing new triangles t_1, t_2 as shown in Figure 13, which are both less obtuse than triangle CMA . If M_1 is inside circumcircle $CC(MAC)$ and far from its boundary, both triangles t_1, t_2 are better than triangle MAC and even more, they can be good triangles. However if M_1 is close to the circumcircle boundary, the smallest angle(s) of vertex M_1 can be close to, or even can be worse than the smallest angle of triangle BAC (in the case that triangle BAC is acute).

In this section we prove that after a finite number of point insertions (usually one or two additional point insertions) a significant discrete improvement

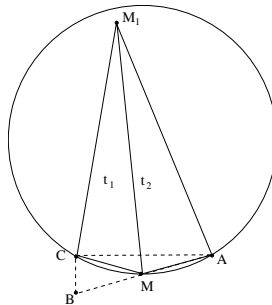


Fig. 13. M_1 close to the circumcircle introduces bad triangles t_1, t_2 .

is achieved in the smallest angles involved. Consequently slow angle improvement, neither slow angle worsening does not happen throughout the process.

We need to consider two cases:

Case 1: MM_1 is a terminal edge.

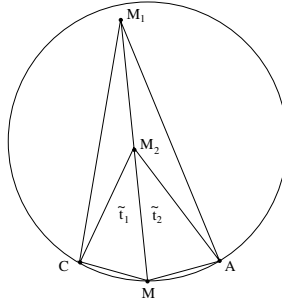


Fig. 14. M_1M is a terminal edge.

Case 2: t_1, t_2 are Lepp triangles. Here we assume that t_2 belongs to $\text{Lepp}(t_1)$, which implies that the longest edge of t_2 is larger than the longest edge of t_1 .

Thus the following result holds:

Theorem 5. *Consider that M_1 is close to the circumcircle boundary as shown in Figure 13. Then a significant discrete improvement of the smallest angles of the triangles t_1, t_2 is attained by inserting a finite number of terminal edge midpoints inside $CC(t_2)$.*

Proof Case 1. Here t_1, t_2 are terminal triangles (one obtuse and one acute). Then by Delaunay insertion of midpoint M_2 (see Figure 14), the triangles t_1, t_2 are replaced by highly improved triangle \tilde{t}_1, \tilde{t}_2 according to Theorem 2 and Lemma 2. Note also that this implies the introduction of bad too-obtuse triangles M_1CM_2 and M_1M_2A , whose elimination will be performed by edge swapping.

Case 2. Since M_1 is inside circumcircle $CC(MAC)$ and close to its boundary, this implies that both $CC(t_1)$ and $CC(t_2)$ are approximately equal to $CC(MAC)$. So both triangles t_1, t_2 are eliminated by inserting a point M_2 inside $CC(t_2)$ unless M_2 is too close to the boundary of $CC(t_2)$ but outside $CC(t_1)$.

We need to consider three cases:

- (i) M_1A is terminal edge and M_2 is midpoint of M_1A
- (ii) M_2 is midpoint of a neighbor terminal edge and is far from the boundary of $CC(t_2)$.
- (iii) M_2 is near to the boundary of $CC(t_2)$

In the case (i), the insertion of M_2 produces highly improved, more equilateral triangles (see Figure 15(a)). Note that triangle M_2MA is the best longest edge son of triangle M_1MA , for which the angle bounds of Theorem 2 and Lemma 2 apply. Note also that triangle M_1CM_2 is an obtuse longest edge son of better triangle M_1CA .

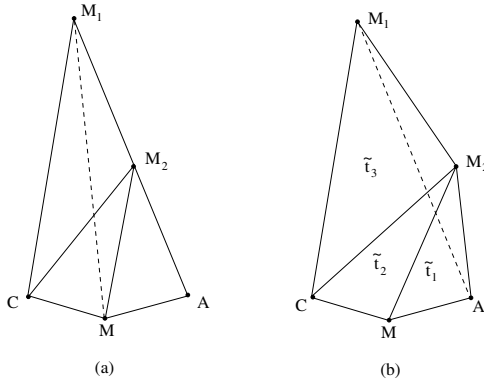


Fig. 15. (a) M_1A is a terminal edge; (b) M_2 is midpoint of neighbor terminal edge and far from the boundary of $CC(t_2)$.

For the case (ii), the insertion of M_2 produces a configuration of better triangles $\tilde{t}_1, \tilde{t}_2, \tilde{t}_3$ (see Figure 15(b)) similar to that of case (i), where $CC(M_2MA)$ contains the midpoint of edge CM_2 and probably the midpoint of edge CM_1 . Thus if any of these triangles is not acceptable, its processing will imply the insertion of either the midpoint of CM_2 and / or the midpoint of CM_1 which will produce a significant discrete improvement of the current bad angles without introducing too-obtuse angles.

For the case iii) we need to consider in turn two cases: **iii(1)** M_2 is inside $CC(t_1)$. Here the insertion of M_2 close to the boundary of $CC(t_2)$ implies the configuration of Figure 16(a), where either CM_1 or CM_2 becomes a terminal edge with midpoint M_3 , in the interior of $CC(t_1)$, whose Delaunay insertion produces significant discrete improvement of the smallest angles. Figure 16(a) illustrates the case where M_3 is midpoint of CM_2 ; note that CMM_3 is the best longest edge son of triangle CMM_2 , edge MM_2 is swapped by edge M_3A which eliminates the smallest angle at M_2 introducing highly more equilateral triangles; moreover angle $CM_1M_3 > \text{angle } CM_1M$.

iii(2) M_2 is outside $CC(t_1)$. Here the insertion of M_2 implies the configuration of Figure 16(b), where M_1M becomes a terminal edge with midpoint M_3 , whose Delaunay insertion will produce highly improved triangles and eventually one too-obtuse triangle□

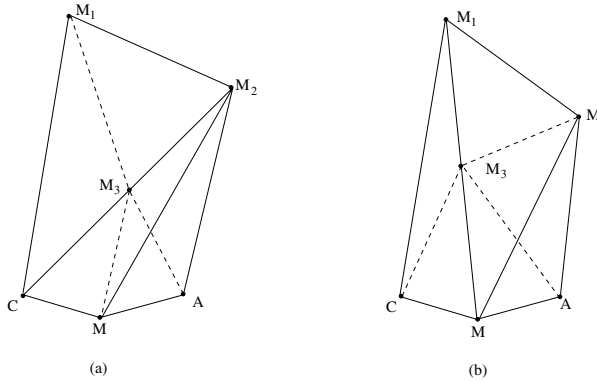


Fig. 16. M_2 close to the boundary of $CC(t_2)$ produces terminal edges inside $CC(t_2)$
 a) M_2 is inside $CC(t_1)$; (b) M_2 is outside $CC(t_1)$.

9 Algorithm Termination

In the case that an acute bad triangle t is being processed, an intermediate worst too obtuse triangle t_A is introduced and eliminated by edge swapping (t_A not necessarily belongs to an intermediate Delaunay triangulation). As discussed in § 8, this operation can introduce a bigger acute triangle that is worse than t as shown in Figure 13. We will show that only a finite chain of worsening triangles can occasionally appear in the mesh. Indeed the following more general result holds:

Lemma 7. *Only a finite sequence of worsening and / or lightly improving too-obtuse triangles can (rarely) appear in the processing mesh.*

Proof Firstly assume that there exists an infinite sequence of monotonically worst acute triangles $\{t_i\}_{i=1}^\infty$, and worst obtuse triangles $\{t_A^i\}_{i=1}^\infty$, where each circumcircle $CC(t_A^i)$ is bigger than $CC(t_i)$ and contains a point P_i (a previous point or an added point) close to the circumcircle boundary and far from t_A^i , consequently edge swapping applies producing a worst bigger acute triangle t_{i+1} , a worst bigger t_A^{i+1} and an increasing circumcircle $CC(t_A^{i+1})$, for $i = 1, 2, \dots$ However since the geometry is finite eventually one of these circumcircles will contain a boundary vertex and will intersect boundary constrained edges, which will stop the generation of bigger triangles.

An analogous reasoning can be used for a sequence of lightly improving obtuse triangles $\{t_A^i\}$, and for a sequence that combines lightly improving / worsening obtuse triangles \square

Imposing the following assumption we can prove algorithm termination:

- (I) **Geometry assumption** The PSLG geometry does not have constrained angle less than $\pi/2$ (analogously to the Ruppert condition).

Theorem 6. *Given the assumption (I) and angle tolerance $\theta_{tol} \leq \alpha_{limit}$, the algorithm terminates.*

Proof Consider a *CDT* of the input geometry data, and let t_r be a right constrained terminal triangle (with constrained right angle). Firstly we study the constrained edge point insertion criterion of § 3.1 for this kind of triangles. We need to consider two cases: (i) If the smallest angle is $\alpha_0 \geq 30^\circ$, then the midpoint of the longest edge is Delaunay inserted in the constrained mesh, which implies the introduction of two quality triangles, each one having one constrained edge. (ii) Otherwise if $\alpha_0 < 30^\circ$, the midpoint of the second longest edge is Delaunay inserted in the constrained mesh. This introduces a better right constrained triangle t_r , and a too-obtuse triangle with constrained smallest edge, which is consequently eliminated by swapping of its longest edge. The process is repeated for the new right constrained triangle either until this is a good triangle, or until its smallest angle is $\geq 30^\circ$. In the latter case, if the current t_r becomes a terminal triangle, case (i) again applies.

Finally termination is based on the angle improvement results of theorems 2, 3, 5, lemmas, 1, 2 and on Lemma 7□

10 Concluding Remarks

We have presented a geometrical characterization of unconstrained Delaunay terminal triangles. Also, for a bad Delaunay terminal triangle t , we have presented bounds on the angles and bounds on the distance from the terminal edge midpoint M to previous vertices. We have developed constraints on the next points inserted after M , which allow to consider improvement steps that use several consecutive point insertions to highly improve too obtuse triangles introduced by longest edge bisection of t . We have used these results to prove algorithm termination. In future research we expect to prove that optimal-size triangulations are obtained.

Acknowledgements

I am happy to acknowledge helpful discussions with Bruce Simpson. I thank the referees whose comments helped to improve this paper. This research has been partially supported by Proyecto Fondecyt 1040713.

References

1. M. Bern, D. Eppstein and J. Gilbert, Provably good mesh generation. *Journal Computer System Science*, 48, 1994, 384–409.
2. L.P.Chew, Guaranteed-quality triangular meshes. *Technical report TR-98-983, Computer Science Department, Cornell University, Ithaca, NY*, 1989.
3. P L George and H Borouchaki, *Delaunay Triangulation and Meshing*. Hermes, 1998.
4. H Borouchaki and P L George, Aspects of 2-D Delaunay Mesh Generation. *International Journal for Numerical Methods in Engineering*, **40**, 1997, 1957–1975.
5. R.E.Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0*. SIAM, 1998.
6. M. C. Rivara, Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *International Journal for Numerical Methods in Engineering*, **20**, 1984, 745–756.
7. M. C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *International Journal for Numerical Methods in Engineering*, **28**, 1989, 2889–2906.
8. M. C. Rivara and C. Levin. A 3d Refinement Algorithm for adaptive and multigrid Techniques. *Communications in Applied Numerical Methods*, **8**, 1992, 281–290.
9. P Morin, R H Nochetto, and K G Siebert, Convergence of Adaptive Finite Element Methods, *SIAM Review*. **44** 631–658.
10. S. N. Muthukrishnan, P. S. Shiakolos R. V. Nambiar, and K. L. Lawrence. Simple algorithm for adaptative refinement of three-dimensional finite element tetrahedral meshes. *AIAA Journal*, **33**, 1995, 928–932.
11. N. Nambiar, R. Valera, K. L. Lawrence, R. B. Morgan, and D. Amil. An algorithm for adaptive refinement of triangular finite element meshes. *International Journal for Numerical Methods in Engineering*, **36**, 1993, 499–509.
12. M. C. Rivara and G. Iribarren, The 4-triangles longest-edge partition of triangles and linear refinement algorithms, *Mathematics of Computation*, 65, 1996, 1485–1502.
13. M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, **40**, 1997, 3313–3324.
14. J Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. of Algorithms*, **18**, 1995, 548–585.
15. N. Hitschfeld and M.C. Rivara. Automatic construction of non-obtuse boundary and/or interface Delaunay triangulations for control volume methods. *International Journal for Numerical Methods in Engineering*, **55**, 2002, 803–816.
16. N. Hitschfeld, L. Villablanca, J. Krause, and M.C. Rivara. Improving the quality of meshes for the simulation of semiconductor devices using Lepp-based algorithms. *to appear. International Journal for Numerical Methods in Engineering*, 2003.
17. M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, **33**, 2001, 263–277.

18. M. C. Rivara and M. Palma. New LEPP Algorithms for Quality Polygon and Volume Triangulation: Implementation Issues and Practical Behavior. *In Trends unstructured mesh generation*, Eds: S. A. Cannan . Saigal, AMD, **220**, 1997, 1–8.
19. T.J. Baker, Automatic mesh generation for complex three dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, **5**, 1989, 161–175.
20. T. J. Baker, Triangulations, Mesh Generation and Point Placement Strategies. *Computing the Future*, ed. D Caughey, John Wiley, 61–75.
21. J R Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *First Workshop on Applied Computational Geometry*, ACM, 1996, 124–133.
22. R.B. Simpson, N. Hitschfeld and M.C. Rivara, Approximate quality mesh generation, *Engineering with computers*, **17**, 2001, 287–298.
23. M. Bern, Triangulations, In Handbook of Discrete and Computational Geometry *J. E. Goodman and J O'Rourke (eds.)*, CRC Press Boca Raton, 1997.
24. D. T. Lee and A. Lin Generalized Delaunay triangulation for planar graphs. *Disc and Comp Geom*, bf 1, 1986, 201–217.
25. N.P. Weatherill and O. Hassan, Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *IJMNE*, bf 37, 1994, 2005–2039.
26. D.L. Marcum and N.P. Weatherill, Aerospace applications of solution adaptive finite element analysis. *CAGEOD*, bf 12, 1995, 709–731
27. I.G. Rosenberg and F. Stenger, A lower bound on the angles of triangles constructed by bisecting the longest side, *Mathematics of Computation*, 29, 1975, 390–395.
28. A. Üngör, Off-centers: a new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations, *Latin 2004, LNCS 2076*, 2004, 152–161.
29. B. Simpson and M.C. Rivara, Geometrical mesh improvement properties of Delaunay terminal edge refinement, *Geometric Modeling and Processing 2006*, Pittsburgh, 2006.
30. M.de Berg, M Van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, algorithms and applications*, second edition, Springer, 2000.
31. M. C. Rivara, New mathematical tools and techniques for the refinement and / or improvement of unstructured triangulations, *Proceedings 5th International Meshing Roundtable*, Pittsburgh, 77–86, 1996.

Generalized Delaunay Mesh Refinement: From Scalar to Parallel

Andrey N. Chernikov and Nikos P. Chrisochoides

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185
{`ancher,nikos`}@`cs.wm.edu`

Summary. The contribution of the current paper is three-fold. First, we generalize the existing sequential point placement strategies for guaranteed quality Delaunay refinement: instead of a specific position for a new point, we derive a selection disk inside the circumdisk of a poor quality triangle. We prove that any point placement algorithm that inserts a point inside the selection disk of a poor quality triangle will terminate and produce a size-optimal mesh. Second, we extend our theoretical foundation for the parallel Delaunay refinement. Our new parallel algorithm can be used in conjunction with any sequential point placement strategy that chooses a point within the selection disk. Third, we implemented our algorithm in C++ for shared memory architectures and present the experimental results. Our data show that even on workstations with a few cores, which are now in common use, our implementation is significantly faster the best sequential counterpart.

1 Introduction

In this paper we address theoretical and practical aspects for the development of both scalar and parallel Delaunay mesh generation algorithm and software that satisfy the following requirements:

1. allow to construct well-shaped elements with bounded minimal angle;
2. produce graded meshes, i.e., meshes with element size specified by a user-defined function;
3. offer proofs of termination and size optimality;
4. allow to use custom point placement strategies (e.g., circumcenter, off-center, etc.);
5. replace the solution of a difficult domain decomposition problem with an easier data distribution approach without relying on the speculative execution model [10, 19];
6. offer performance improvement over the best available sequential software, even on workstations with just a few hardware cores.

We describe our solution which satisfies all of these requirements. Although the extension of the method to three dimensions is still in progress, we present a complete practical parallel two-dimensional guaranteed quality graded mesh generator. In such applications as the direct numerical simulations of turbulence in cylinder flows with very large Reynolds numbers [12] and coastal ocean modeling for predicting storm surge and beach erosion in real-time [25], three-dimensional simulations are conducted using two-dimensional meshes. In both cases, 2D mesh generation is taking place in the xy -plane and it is replicated in the z -direction in the case of cylinder flows or using bathymetric contours in the case of coastal ocean modeling applications.

The field of sequential guaranteed quality Delaunay refinement has been extensively studied, see [8, 13, 16, 20, 23] and the references therein. However, new ideas and improvements keep being introduced. One of the basic questions being studied is where to insert additional (so-called Steiner) points into an existing mesh in order to improve the quality of the elements. Ruppert's [20] and early Chew's [8] algorithms use circumcenters of poor quality triangles. Later, Chew [9] suggested to use randomized insertion of *near-circumcenter* points for three-dimensional Delaunay refinement, with the goal of avoiding slivers. Recently, Üngör [24] proposed to insert specially chosen points which he calls *off-centers*; this method allows to produce smaller meshes in practice and it was implemented in the popular sequential mesh generation software the Triangle [22]. We expect that other optimization techniques can be used to select positions for new points. Indeed, in Subsection 2.2 we give an example of a point placement strategy that in some cases allows to achieve even smaller meshes than the off-center method, albeit at significant computation cost. Since one would not like to redesign the parallel algorithm and software to accommodate each of the point placement techniques, in this paper we generalize the sequential Delaunay refinement approaches and develop a framework which allows to use custom point selection strategies. In particular, we derive a *selection disk* for the position of a new point with respect to a poor quality triangle and prove that any point placement technique with the only restriction that it selects a point inside the selection disk will terminate and produce a size-optimal guaranteed quality mesh. While the use of Chew's [9] *picking-sphere* is restricted to produce only meshes with constant density, the use of our selection disk allows to obtain graded size-optimal meshes.

The domain decomposition problem for parallel mesh generation is formulated as follows [11, 14, 15]. Given a domain $\Omega \subset \mathbb{R}^n$, construct the separators $S_{ij} \subset \mathbb{R}^{n-1}$, $S_{ij} \subset \Omega$, such that Ω is decomposed into *subdomains* Ω_i :

$$\Omega = \bigcup_{i=1}^N \Omega_i, \quad \Omega_i \cap \Omega_j = S_{ij}, \quad i, j = 1, \dots, N, \quad i \neq j,$$

while the separators do not create very small angles and other features that will force the degradation of the mesh quality. Linardakis and Chrisochoides [14, 15] described a Medial Axis Domain Decomposition Method for

two-dimensional geometries. However, the solution is based on the Medial Axis Transform which is very difficult and expensive to construct for three-dimensional geometries. Another approach is to partition and refine the mesh simultaneously [10, 19], such that when a conflict is detected between concurrently inserted points, some of the point insertions are canceled, which leads to high computation and communication costs.

In [4–7] we showed that the domain decomposition problem for the parallel Delaunay refinement can be replaced with an easier data distribution problem. We proved that an auxiliary spatial data structure, like a uniform 2D (or 3D) lattice and a quadtree (or an octree), can be constructed in such a way that the points introduced in certain regions of Ω , that correspond to separated regions from the above data structure, do not have any dependences and can be inserted concurrently.

In [5, 7] we presented the theory and the implementation for the parallel construction of guaranteed quality uniform two-dimensional meshes which use a uniform lattice as an auxiliary spatial data structure. In [4, 6] we presented the theoretical foundation for the construction of non-uniform (graded) meshes for the circumcenter point insertion method [8, 20, 23]. In this paper, we present the algorithm for the generalized point insertion method, describe our implementation and the experimental results using the off-center point insertion strategy [24].

Ruppert's sequential Delaunay refinement algorithm has quadratic worst-case running time, even though in most practical cases the time is linear with respect to the output size [20, 23]. Recently, Miller [16] proposed a Delaunay refinement algorithm which runs in optimal $\mathcal{O}(n \log n + m)$ time, where n is the size of the input, and m is the size of the output. He achieved this improvement by introducing a priority queue, where the skinny triangles are ordered by their diameter (equivalently, circumradius), and the triangles with the largest diameter are refined first. As it can be seen further in the paper, our parallel algorithm also gives priority to triangles with large circumradii, which allows to eliminate quadratic running time for pathological input geometries.

Cheng et al. [3] developed an algorithm to remove sliver tetrahedra from an existing Delaunay mesh. The algorithm pumps the weights of the vertices and flips the edges to obtain a new triangulation of the same point set. The maximum weight that can be assigned to a point is bounded by a function of the distance to the nearest point. This relates to the choice of radius for our selection disk which depends on the length of the shortest edge of a triangle.

In Section 2 we introduce the background for the sequential Delaunay refinement, define the selection disk for the insertion of Steiner points, and present the proofs of termination and size optimality to show that a Delaunay refinement algorithm which chooses points inside the selection disks of skinny triangles terminates and produces size optimal meshes. Then, in Section 3 we describe our parallel implementation and experimental results. Section 4 concludes the paper.

2 Point Insertion for Sequential Delaunay Refinement

2.1 Delaunay Refinement Background

Let the mesh $\mathcal{M} = (V, T, S)$ consist of a set $V = \{p_i\}$ of vertices, a set $T = \{t_i = \triangle(p_u p_v p_w) \mid p_u, p_v, p_w \in V\}$ of triangles, and a set $S = \{s_i = p_u p_v \mid p_u, p_v \in V\}$ of constrained segments. We will denote an edge of a triangle as $e(p_i p_j)$. The input to a planar triangular mesh generation algorithm includes a description of *domain* $\Omega \subset \mathbb{R}^2$, which is permitted to contain holes or have more than one connected component. We will use a *Planar Straight Line Graph* (PSLG) [22] to delimit Ω from the rest of the plane. Each segment in the PSLG is considered *constrained* and must appear (possibly as a union of smaller subsegments) in the final mesh. The vertices of the PSLG are a subset of the final set of vertices in the mesh.

There are two commonly used parameters that control the quality of mesh elements: an upper bound on the circumradius-to-shortest edge ratio (which is equivalent to a lower bound on a minimal angle [17]) and an upper bound on the element area. We will denote the circumradius-to-shortest edge ratio of triangle t as $\rho(t)$ and the area of triangle t as $\Delta(t)$. The former upper bound is usually fixed and given by a constant value $\bar{\rho}$, while the latter can vary and be controlled by some user-defined grading function $\bar{\Delta}(\cdot)$, which can be defined either over the set of triangles or over Ω , depending on the implementation.

Let us call the open disk corresponding to a triangle’s circumscribed circle its *circumdisk*. We will use symbols $\circ(t)$ and $r(t)$ to represent the circumdisk and the circumradius of triangle t , respectively. A mesh is said to satisfy the *Delaunay property* if the circumdisk of every triangle does not contain any of the mesh vertices [13, 23].

Delaunay mesh generation algorithms start with constructing an initial mesh, which conforms to the input PSLG, and then refine this mesh until the element quality constraints are met. In this paper, we focus on parallelizing the Delaunay refinement stage, which is usually the most memory- and computation-expensive. The general idea of Delaunay refinement is to insert additional (Steiner) points inside the circumdisks of poor quality triangles, which causes these triangles to be destroyed, until they are gradually eliminated and replaced by better quality triangles.

We will extensively use the notion of *cavity* [13] which is the set of triangles in the mesh whose circumdisks include a given point p_i . We will denote $\mathcal{C}(p_i)$ to be the cavity of p_i and $\partial\mathcal{C}(p_i)$ to be the set of edges which belong to only one triangle in $\mathcal{C}(p_i)$, i.e., external edges. For our analysis, we will use the Bowyer-Watson (B-W) point insertion algorithm [2, 26]:

$$\begin{aligned} V' &\leftarrow V \cup \{p_i\}, \\ T' &\leftarrow T \setminus \mathcal{C}(p_i) \cup \{\triangle(p_i p_j p_k) \mid e(p_j p_k) \in \partial\mathcal{C}(p_i)\}, \end{aligned} \tag{1}$$

where $\mathcal{M} = (V, T, S)$ and $\mathcal{M}' = (V', T', S')$ represent the mesh before and after the insertion of p_i , respectively.

Sequential Delaunay algorithms treat *constrained* segments differently from triangle edges [20, 23]. A vertex p is said to *encroach upon* a segment s , if it lies within the open diametral disk of s [20]. When a new point is about to be inserted and it happens to encroach upon a constrained segment s , another point is inserted in the middle of s instead [20], and a cavity of the segment’s midpoint is constructed and triangulated according to (1).

The proofs of termination and size optimality of Delaunay refinement algorithms [20, 23] explore the relationships between the insertion radius of a point and that of its parent. The *insertion radius* $R(p)$ of point p is defined as the length of the shortest edge connected to p immediately after p is inserted into the mesh [23]. The *parent* \hat{p} of point p is the vertex which is “responsible” for the insertion of p [23]. In particular, if p is inserted inside the circumdisk of a poor quality triangle, \hat{p} is the most recently inserted vertex of the shortest edge of that triangle. If p is a midpoint of an encroached segment, \hat{p} is the point (possibly rejected for insertion) that encroaches upon that segment. If p is an input vertex, it has no parent. In addition, the proofs require that $\bar{\rho} \geq \sqrt{2}$.

The local feature size function $\text{lfs} : \mathbb{R}^2 \rightarrow \mathbb{R}$ for a given point p is equal to the radius of the smallest disk centered at p that intersects two non-incident vertices or segments of PSLG \mathcal{X} [20]. $\text{lfs}(p)$ satisfies the Lipchitz condition:

Lemma 1 (Lemma 1 in Ruppert [20], Lemma 2 in Shewchuk [23]). *Given any PSLG \mathcal{X} and any two points p_i and p_j in the plane, the following inequality holds:*

$$\text{lfs}(p_i) \leq \text{lfs}(p_j) + \|p_i - p_j\| \tag{2}$$

2.2 Delaunay Refinement Using Selection Disks

Traditionally, Steiner points are selected in the circumcenters of poor quality triangles [8, 20, 23]. However, Chew [9] chooses Steiner points randomly inside a *picking sphere* to avoid slivers. His goal is to construct a mesh with constant density; therefore he proves the termination, but not the size-optimality of the mesh. Ruppert [20] and Shewchuk [23] proved that if $\bar{\rho} \geq \sqrt{2}$, then Delaunay refinement with circumcenters terminates and, furthermore, produces size-optimal meshes. In this context, size-optimality means that the number of triangles in the resulting mesh will be within a constant multiple of the smallest possible number of triangles satisfying the input constraints.

Recently, Üngör [24] introduced a new type of Steiner points called *off-centers*. The idea is based on the observation that sometimes the triangles created as a result of inserting circumcenters of skinny triangles are also skinny and require further refinement. It combines the advantages of advancing front methods, which can produce very well-shaped elements in practice, and Delaunay methods, which offer theoretical guarantees. Üngör showed that the use of off-centers allows to significantly decrease the size of the final mesh in practice. Consider Figure 1(left). Suppose $\triangle(p_k p_l p_m)$ is skinny: $\rho(\triangle(p_k p_l p_m)) > \bar{\rho}$. If

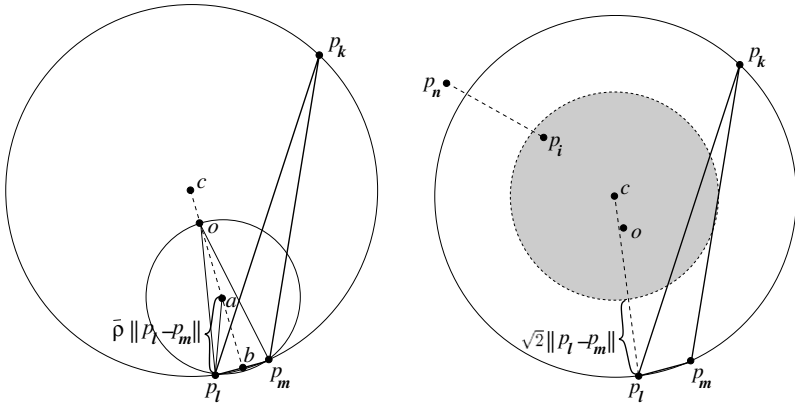


Fig. 1. (Left) Delaunay refinement with off-centers [24]. **(Right)** The selection disk (shaded) for the insertion of a Steiner point.

we insert its circumcenter c , the new triangle $\triangle(c p_l p_m)$ may also be skinny. In this case, instead of inserting c , Üngör suggests to insert the *off-center* o chosen on the perpendicular bisector of the shortest edge $e(p_l p_m)$ in such a way that the new triangle $\triangle(o p_l p_m)$ will have circumradius-to-shortest edge ratio equal to exactly $\bar{\rho}$. While eliminating additional point insertions, this strategy creates triangles with the longest possible edges, such that one can prove termination of the algorithm and size-optimality of the result.

However, circumcenters and off-centers are not the only possible positions for inserting the Steiner points, either sequentially or in parallel, such that one can prove termination and size-optimality.

Definition 1. *If t is a poor quality triangle with circumcenter c , shortest edge length l , circumradius r , and circumradius-to-shortest edge ratio $\rho = r/l > \bar{\rho} \geq \sqrt{2}$, then the selection disk for the insertion of a Steiner point that would eliminate t is the open disk with center c and radius $r - \sqrt{2}l$.*

For example, in Figure 1(right) $e(p_l p_m)$ is the shortest edge of a skinny triangle $\triangle(p_k p_l p_m)$ and c is its circumcenter. The selection disk is the shaded disk with center c and radius $r(\triangle(p_k p_l p_m)) - \sqrt{2}\|p_l - p_m\|$.

Below we prove that any point inside the selection disk of a triangle can be chosen for the elimination of the triangle, and that the generalized Delaunay refinement algorithm which chooses Steiner points inside the selection disks terminates and produces size-optimal meshes.

Remark 1. The radius of Chew’s picking sphere is fixed and is equal to one half of the length of the shortest possible edge in the final mesh [9]. We generalize the idea of his picking sphere to the selection disk, such that the radius of the selection disk varies among the triangles and depends on the length of the shortest edge of each particular triangle.

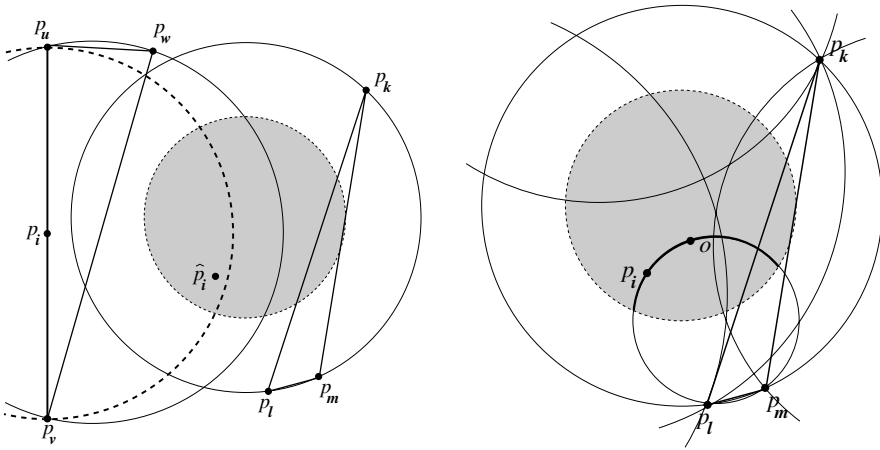


Fig. 2. (Left) \hat{p}_i is a Steiner point within a selection disk of a poor quality triangle which encroaches upon a constrained segment $e(p_u p_v)$. (Right) An optimization-based method for the selection of a Steiner point within a selection disk of a poor quality triangle.

Remark 2. Üngör’s off-center always lies inside the selection disk.

Remark 3. As $\rho(\Delta(p_k p_l p_m))$ approaches $\sqrt{2}$, the selection disk shrinks to the circumcenter c of the triangle. If, furthermore, $\rho(\Delta(p_k p_l p_m)) \leq \sqrt{2}$, the selection disk vanishes, which coincides with the fact that the triangle $\Delta(p_k p_l p_m)$ cannot be considered skinny.

The proofs of termination and size-optimality of Delaunay refinement with circumcenters in [8, 20, 23] rely on the assumption that the insertion radius of the Steiner point is equal to the circumradius of the poor quality triangle. This assumption holds when the Steiner point is chosen in the circumcenter of a triangle, since by Delaunay property the circumdisk of the triangle is empty, and, hence, there is no vertex closer to the circumcenter than the vertices of this triangle. However, the above assumption does **not** hold if we pick an arbitrary point p_i within the selection disk, see Figure 1(right). Therefore, we need new proofs in the new context when Steiner points can be inserted anywhere within the selection disks of poor quality triangles.

Proof of Termination

Lemma 2. *If p_i is a vertex of the mesh produced by a Delaunay refinement algorithm which chooses points within selection disks of triangles with circumradius-to-shortest-edge ratios greater than $\bar{\rho} \geq \sqrt{2}$, then the following inequality holds:*

$$R(p_i) \geq C \cdot R(\hat{p}_i), \tag{3}$$

where we distinguish among the following cases:

(i) $C = \sqrt{2}$ if p_i is a Steiner point chosen within the selection disk of a skinny triangle;

Otherwise, let p_i be the midpoint of subsegment s . Then

(ii) $C = \frac{1}{\sqrt{2}}$ if \hat{p}_i is a Steiner point which encroaches upon s , chosen within the selection disk of a skinny triangle;

(iii) $C = \frac{1}{2 \cos \alpha}$ if p_i and \hat{p}_i lie on incident subsegments separated by an angle of α (with \hat{p}_i encroaching upon s), where $45^\circ \leq \alpha \leq 90^\circ$;

(iv) $C = \sin \alpha$ if p_i and \hat{p}_i lie on incident segments separated by an angle of $\alpha \leq 45^\circ$.

If p_i is an input vertex, then

$$R(p_i) \geq \text{lfs}(p_i). \tag{4}$$

Proof. We need to present new proofs only for cases (i) and (ii), since the proofs for all other cases are independent of the choice of the point within the selection disk and are given in [23].

Case (i) By the definition of a parent vertex, \hat{p}_i is the most recently inserted endpoint of the shortest edge of the triangle; without loss of generality let $\hat{p}_i = p_l$ and $e(p_l p_m)$ be the shortest edge of the skinny triangle $\Delta(p_k p_l p_m)$, see Figure 1(right). If $e(p_l p_m)$ was the shortest edge among the edges incident upon p_l at the time p_l was inserted into the mesh, then $\|p_l - p_m\| = R(p_l)$ by the definition of the insertion radius; otherwise, $\|p_l - p_m\| \geq R(p_l)$. In either case,

$$\|p_l - p_m\| \geq R(p_l). \tag{5}$$

Now we can derive the relation between the insertion radius of point p_i and the insertion radius of its parent $\hat{p}_i = p_l$:

$$\begin{aligned} R(p_i) &> \sqrt{2}\|p_l - p_m\| && \text{(by the construction of the selection disk)} \\ &\geq \sqrt{2}R(p_m). && \text{(from (5))} \end{aligned}$$

Hence, $R(p_i) > \sqrt{2}R(\hat{p}_i)$; choose $C = \sqrt{2}$.

Case (ii) Let \hat{p}_i be inside the selection disk of a skinny triangle $\Delta(p_k p_l p_m)$, such that \hat{p}_i encroaches upon $e(p_u p_v)$, see Figure 2(left). Since the edge $e(p_u p_v)$ is part of the mesh, there must exist some vertex p_w such that p_u , p_v , and p_w form a triangle. Because p_w is outside of the diametral circle of $e(p_u p_v)$, the circumdisk $\circ(\Delta(p_u p_v p_w))$ has to include point \hat{p}_i . Therefore, if \hat{p}_i were inserted into the mesh, $\Delta(p_u p_v p_w)$ would be part of the cavity $\mathcal{C}(\hat{p}_i)$ and the edges connecting \hat{p}_i with p_u and p_v would be created. Therefore,

$$R(\hat{p}_i) \leq \min(\|\hat{p}_i - p_u\|, \|\hat{p}_i - p_v\|) < \sqrt{2} \frac{\|p_u - p_v\|}{2} = \sqrt{2}R(p_i);$$

choose $C = \frac{1}{\sqrt{2}}$. \square

Theorem 1 (Theorem 4 in Shewchuk [23]). *Let lfs_{\min} be the shortest distance between two non-incident entities (vertices or segments) of the input PSLG. Suppose that any two incident segments are separated by an angle of at least 60° , and a triangle is considered to be skinny if its circumradius-to-shortest edge ratio is larger than $\bar{\rho}$, where $\bar{\rho} \geq \sqrt{2}$. Ruppert’s algorithm will terminate with no triangulation edge shorter than lfs_{\min} .*

The proof of this theorem in [23] is based on the result of Lemma 3 in [23], which establishes the relations (3) and (4) in the context of circumcenter point insertion. Otherwise, the proof is independent of the specific position of inserted points. Since we proved (3) and (4) in the context of inserting arbitrary points within selection disks, this theorem also holds in this context.

Proof of Good Grading and Size Optimality

The quantity $D(p)$ is defined as the ratio of $\text{lfs}(p)$ over $R(p)$ [23]:

$$D(p) = \frac{\text{lfs}(p)}{R(p)}. \tag{6}$$

It reflects the density of vertices near p at the time p is inserted, weighted by the local feature size.

Lemma 3. *If p_i is a vertex of the mesh produced by a Delaunay refinement algorithm which chooses points within selection disks of skinny triangles and C is the constant specified by Lemma 2, then the following inequality holds:*

$$D(p_i) \leq 1 + \frac{D(\hat{p}_i)}{C}. \tag{7}$$

Proof. If p_i is chosen within the selection disk of a skinny triangle, then $R(p_i) > \sqrt{2}\|p_i - \hat{p}_i\|$ by construction. If p_i is a segment midpoint and \hat{p}_i is a rejected encroaching Steiner point within a selection disk, $R(p_i) > \|p_i - \hat{p}_i\|$ because \hat{p}_i is inside the diametral circle of the segment. If p_i is a segment midpoint and \hat{p}_i is an encroaching vertex which lies on another segment, then by the definition of the insertion radius $R(p_i) = \|p_i - \hat{p}_i\|$ by the definition of the insertion radius. In all cases,

$$R(p_i) \geq \|p_i - \hat{p}_i\|. \tag{8}$$

Then

$$\begin{aligned} \text{lfs}(p_i) &\leq \text{lfs}(\hat{p}_i) + \|p_i - \hat{p}_i\| && \text{(from Lemma 1)} \\ &\leq \text{lfs}(\hat{p}_i) + R(p_i) && \text{(from (8))} \\ &= D(\hat{p}_i) R(\hat{p}_i) + R(p_i) && \text{(from (6))} \\ &\leq D(\hat{p}_i) \frac{R(p_i)}{C} + R(p_i) && \text{(from Lemma 2)} \end{aligned}$$

The result follows from dividing both sides by $R(p_i)$. \square

Lemma 4 (Extension of Lemma 7 in Shewchuk [23] and Lemma 2 in Ruppert [20]). *Suppose that $\bar{\rho} > \sqrt{2}$ and the smallest angle in the input PSLG is strictly greater than 60° . There exist fixed constants C_T and C_S such that, for any vertex p_i inserted (or considered for insertion and rejected) within the selection disk of a skinny triangle, $D(p_i) \leq C_T$, and for any vertex p_i inserted at the midpoint of an encroached subsegment, $D(p_i) \leq C_S$. Hence, the insertion radius of a vertex has a lower bound proportional to its local feature size.*

The proof of this Lemma in [23] relies only on Lemmata 2 and 3 here which have been proven to hold for the Steiner points chosen within selection disks of skinny triangles. Hence, the Lemma holds in this context, too.

Theorem 2 (Theorem 8 in Shewchuk [23], Theorem 1 in Ruppert [20]). *For any vertex p_i of the output mesh, the distance to its nearest neighbor is at least $\frac{\text{lfs}(p_i)}{C_S+1}$.*

The proof in [23] relies only on Lemmata 1 and 4 here and, therefore, holds for the arbitrary points chosen within selection disks of skinny triangles.

Theorem 2 is the precondition of the following theorem:

Theorem 3 (Theorem 10 in Shewchuk [23], Theorem 14 in Mitchell [18], Theorem 3 in Ruppert [20]). *Let $\text{lfs}_{\mathcal{M}}(p_i)$ be the local feature size at p_i with respect to a mesh \mathcal{M} (treating \mathcal{M} as a PSLG), whereas $\text{lfs}(p_i)$ remains the local feature size at p_i with respect to the input PSLG. Suppose a mesh \mathcal{M} with smallest angle θ has the property that there is some constant $k_1 \geq 1$, such that for every point p_i , $k_1 \text{lfs}_{\mathcal{M}}(p_i) \geq \text{lfs}(p_i)$. Then the cardinality of \mathcal{M} is less than k_2 times the cardinality of any other mesh of the input PSLG with smallest angle θ , where $k_2 = \mathcal{O}(k_1^2/\theta)$.*

An Example of a Point Selection Strategy

Let us consider Figure 2(right). We can see that the off-center o of the skinny triangle $\Delta(p_k p_l p_m)$ is not the only location for a Steiner point p_i that will lead to the creation of the new triangle incident to the edge $e(p_l p_m)$ with circumradius-to-shortest edge ratio equal to exactly $\bar{\rho}$. The arc shown in bold in the Figure is the intersection of the circle passing through p_l, p_m , and o with the selection disk of $\Delta(p_k p_l p_m)$. Let us denote this arc as Γ . The thin arcs show parts of the circumcircles of other triangles in the mesh. We can observe that the cavity $\mathcal{C}(p_i)$ will vary depending on the location of p_i , according to the set of triangle circumdisks that include p_i . Let us also represent the penalty for deleting triangle t as $P(t)$:

$$P(t) = \begin{cases} -1, & \text{if } (\rho(t) > \bar{\rho}) \vee (\Delta(t) > \bar{\Delta}), \\ 1, & \text{otherwise.} \end{cases}$$



Fig. 3. (Left) Un upper part of a model of cylinder flow. (Right) Pipe cross-section model.

Table 1. The comparison of the number of triangles generated with the use of three different point insertion strategies, for different models and minimal angle bounds. No area bound is used.

Point position	$\theta = 10^\circ$		$\theta = 20^\circ$		$\theta = 30^\circ$	
	flow	pipe	flow	pipe	flow	pipe
Circumcenter	2173	3033	3153	4651	8758	10655
Off-center	1906	2941	2942	4411	6175	8585
Our optimization-based method	1805	2841	2932	4359	6319	8581

Then our objective is to minimize the profit function associated with the insertion of point p_i as the sum of the penalties for deleting all triangles in the cavity $\mathcal{C}(p_i)$:

$$\min_{p_i \in \Gamma} F(p_i), \quad F(p_i) = \sum_{t \in \mathcal{C}(p_i)} P(t)$$

In other words, we try to minimize the number of deleted good quality triangles and at the same time to maximize the number of deleted poor quality triangles. The results of our experiments with the cylinder flow (Fig. 3(left)) and the pipe cross-section (Fig. 3(right)) models using Triangle version 1.6 [22] are summarized in Tables 1 and 2. We do not list the running times since our experimental implementation is built on top of the Triangle and does not take advantage of its intermediate calculations as do the circumcenter and off-center insertion methods. From Table 1 we can see that our optimization-based method tends to produce up to 20% fewer triangles than the circumcenter method and up to 5% fewer triangles than the off-center method for small values of the minimal angle bound and no area bound, and the improvement diminishes as the angle bounds increase. Table 2 lists the results of the similar experiments, with an additional area bound constraint. We observe that the introduction of an area bound effectively voids the difference among the presented point insertion strategies.

3 Generalized Parallel Delaunay Refinement

In [6] we described the construction of a quadtree which overlays the mesh. If a part of the mesh associated with a leaf *Leaf* of the quadtree is scheduled

Table 2. The comparison of the number of triangles generated with the use of three different point insertion strategies. For the cylinder flow model, the area bound is set to $\bar{\Delta} = 0.01$, and for the pipe cross-section model $\bar{\Delta} = 1.0$.

Point position	$\theta = 10^\circ$		$\theta = 20^\circ$		$\theta = 30^\circ$	
	flow	pipe	flow	pipe	flow	pipe
Circumcenter	219914	290063	220509	289511	228957	294272
Off-center	219517	290057	220479	289331	226894	295644
Our optimization-based method	219470	289505	220281	289396	226585	294694

DELAUNAYREFINEMENT($\mathcal{X}, g, \bar{\Delta}(\cdot), \bar{\rho}, f(\cdot), \mathcal{M}, Leaf$)

Input: \mathcal{X} is a PSLG which defines Ω

g is the granularity

$\bar{\Delta}(\cdot)$ is the triangle area grading function

$\bar{\rho}$ is the upper bound on triangle circumradius-to-shortest edge ratio

$f(\cdot)$ is a deterministic function which returns a specific position within triangle's selection disk

\mathcal{M} is the current Delaunay mesh

$Leaf$ is the leaf scheduled for refinement

Output: Locally refined Delaunay mesh \mathcal{M}

Locally refined quadtree node $Leaf$

```

1   $i_{min} \leftarrow \min_{PoorTriangles_i(Leaf) \neq \emptyset} i$ 
2   $i_{max} \leftarrow i_{min} + g$ 
3  for  $j = i_{min}, \dots, i_{max}$ 
4    while  $PoorTriangles_j(Leaf) \neq \emptyset$ 
5      Let  $t \in PoorTriangles_j(Leaf)$ 
6       $p \leftarrow f(t)$ 
7      Insert  $p$  into  $\mathcal{M}$ 
8      for  $L \in \{Leaf\} \cup BUF(Leaf)$ 
9        Update  $PoorTriangles(L)$  and  $Counter(L)$ 
10     endfor
11   endwhile
12 endfor
13 Split  $Leaf$  recursively while (9) holds
14 return  $\mathcal{M}, Leaf$ 

```

Fig. 4. The algorithm executed by each of the refinement threads.

for refinement by a thread, no other thread can refine the parts of the mesh associated with the buffer zone $BUF(Leaf)$ of this leaf. For each leaf of the quadtree the following relation is maintained:

$$\forall t \in \mathcal{M} : \bigcirc(t) \cap Leaf \neq \emptyset \implies r(t) < \frac{1}{4} \ell(Leaf), \tag{9}$$

where $\ell(Leaf)$ is the length of the side of $Leaf$. See [6] for the details.

The algorithm is designed for the execution by one master thread which manages the work pool and multiple refinement threads which refine the mesh

and the quadtree. Figure 4 presents the part of the algorithm executed by each of the refinement threads.

When a quadtree leaf $Leaf$ is scheduled for refinement, we remove not only the nodes from the buffer zone $BUF(Leaf)$ of $Leaf$ from the refinement queue, but also the nodes from $BUF(L)$ for each $L \in BUF(Leaf)$. Although this is not required by our theory, there are two implementation considerations for doing so, and both are related to the goal of reducing fine-grain synchronization.¹ First, each leaf has an associated data structure which stores the poor quality triangles whose circumdisks intersect this leaf, so that we can maintain the relation (9). Even though in theory the refinement of the mesh by concurrent threads is not going to cause problems when the threads work within the same quadtree leaf, in practice we would have to introduce synchronization in line 9 of the algorithm in Figure 4 to maintain this data structure. Second, for efficiency considerations, we followed the design of the triangle data element that is used in the Triangle [22]. In particular, each triangle contains pointers to neighboring triangles for easy mesh traversal. However, if two cavities share an edge and are updated by the concurrent threads, which can be done legitimately in certain cases [6], these triangle-neighbor pointers will be invalidated. For these reasons, we chose to completely separate the sets of leaves affected by the mesh refinement performed by multiple threads.

Each of the worker threads performs the refinement of the mesh and the refinement of the quadtree. The poor quality triangles whose split-points selected by a deterministic function $f(\cdot)$ are inside the square of $Leaf$ are stored in the data structure denoted here as $PoorTriangles(Leaf)$. $Leaf$ needs to be scheduled for refinement if the size of this data structure is not empty. In addition, each $Leaf$ has a counter for the triangles with various ratios of the side length of $Leaf$ to their circumradius. If we denote $\sigma(t, Leaf) = \left\lfloor \log_2 \frac{\ell(Leaf)}{r(t)} \right\rfloor$, then $Counter_i(Leaf) = |\{t \in \mathcal{M} \mid (\bigcirc(t) \cap Leaf \neq \emptyset) \wedge (\sigma(t, Leaf) = i)\}|$. When $Counter_i(Leaf) = 0, \forall i = 1, 2, 3$, it implies that (9) would hold for each of the children of $Leaf$, and $Leaf$ can be split. In [5] we proved that when a point is inserted into a Delaunay mesh using the B-W algorithm, the circumradii of the new triangles are not going to be larger than the circumradii of the triangles in the cavity of the point or those that are adjacent to the cavity. Therefore, new triangles that would violate (9) are not going to be created. Each leaf of the quadtree has associated with it a bucketing structure which holds poor quality triangles:

$$\begin{aligned}
 PoorTriangles_i(Leaf) = \{t \in \mathcal{M} \mid (f(t) \in Leaf) \wedge (\sigma(t, Leaf) = i) \wedge \\
 ((\Delta(t) > \bar{\Delta}(t)) \vee (\rho(t) > \bar{\rho}))\}.
 \end{aligned}$$

At each mesh refinement step, all triangles in $PoorTriangles_j(Leaf)$ are refined, for all $j = i_{min}, \dots, i_{min} + granularity$, where

¹As we have shown in [1], modern SMTs are not suitable for executing fine-grain parallelism in Delaunay mesh refinement.

$i_{min} = \min_{PoorTriangles; (Leaf) \neq \emptyset} i$, and *granularity* ≥ 1 is a parameter that controls how much computation is done during a single mesh refinement call. After a mesh refinement call returns, the feasibility of splitting *Leaf* is evaluated, and it is recursively subdivided if necessary.

3.1 Implementation and Experimental Evaluation

We implemented the algorithm in C++ using Pthreads for thread management. The experiments were conducted on an IBM Power5 node with two dual-core processors running at 1.6 GHz and 8 GBytes of total physical memory. We compared our implementation with the fastest to our knowledge sequential Delaunay mesh generator the Triangle version 1.6 [22]. This is the latest release of the Triangle, which uses the off-center point insertion algorithm [24]. In order to make the results comparable, our GPDR implementation also uses the off-center point insertion [24]. Triangle provides a convenient facility for the generation of meshes respecting user-defined area bounds. The user can write his own `triunsuitable()` function and link it against the Triangle. This function is called to examine each new triangle and to decide whether or not it should be considered big and enqueued for refinement. We encoded our grading function into the `triunsuitable()` function, compiled it into an object file, and linked against both the Triangle and our own GPDR implementation. We ran each of the tests 10 times and used average or median timing measurements as indicated.

Figure 5(left) presents the total running times for several granularity values, as the number of compute threads increases from 1 to 4. One additional thread was used to manage the refinement queue. The mesh was constructed for the pipe cross-section model shown in Figure 3(left), using the grading function

$$\bar{\Delta}(x, y) = c \cdot (\sqrt{(x - 200)^2 + (y - 200)^2} + 1), \quad (10)$$

where $c = 10^{-4}$ and (x, y) is the centroid of a triangle. Thus a triangle is considered big if its area is greater than $\bar{\Delta}(x, y)$. In all tests we used the same 20° degrees angle bound. The total number of triangles produced both by the Triangle and GPDR was approximately 17 million.

We can see that the best running time was achieved using 4 compute threads with the granularity value equal to 2, and it constituted 56% of the Triangle's sequential running time. It is also interesting to see the intersection of lines corresponding to granularities 2 and 3, when the number of compute threads was increased from 3 to 4. This intersection reflects one of the basic tradeoffs in parallel computing, between granularity and concurrency: in order to increase the concurrency we have to decrease the granularity, which introduces more overheads.

Figure 5(right) shows the breakdown of the total execution time for each of the threads. The fact that the management thread is idle for 93% of the total time suggests the possibility of high scalability of the code on larger

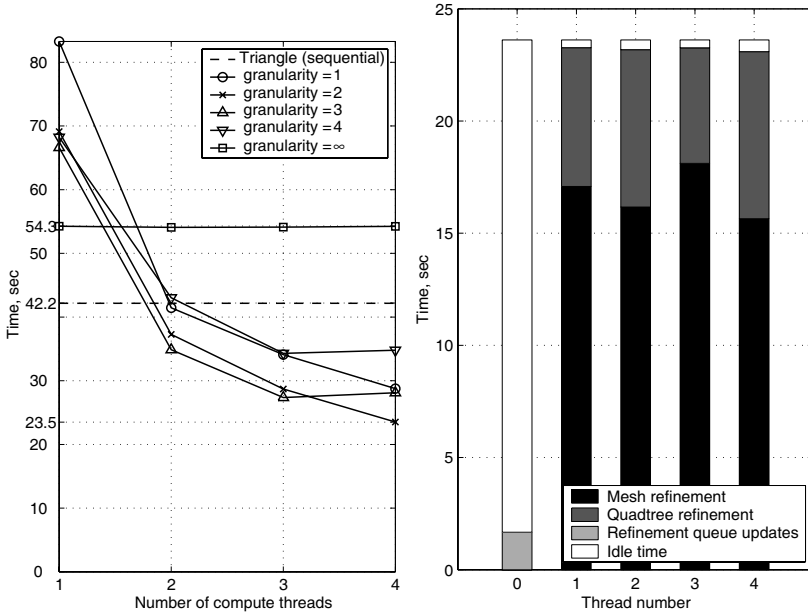


Fig. 5. (Left) The total running time of the GPDR code, for different granularity values, as the number of compute threads is increased from 1 to 4, compared to the Triangle [22]. Each point on the graph is the average of 10 measurements. **(Right)** The breakdown of the total GPDR execution time for each of the threads, when the number of compute threads is 4 and granularity is 2. Thread number 0 performs only the management of the refinement queue, and threads 1–4 perform mesh and quadtree refinement. The data correspond to the test with the median total time.

machines, since it can handle many more refinement threads (cores) than the current widely available machines have.

Standard system memory allocators exhibited high latency and poor scalability in our experiments, which lead us to develop a custom memory management class. At initialization, our memory pool class takes the size of the underlying object (triangle, vertex, quadtree node, or quadtree junction point) as a parameter and at runtime it allocates blocks of memory which can fit a large number of objects. When the objects are deleted, they are not deallocated but are kept for later reuse instead. Our qualitative study of the performance of the standard, the custom, and a novel generic multiprocessor allocator appears in [21].

4 Conclusions

We analyzed the existing point insertion methods for guaranteed quality Delaunay refinement and unified them into a framework which allows to develop

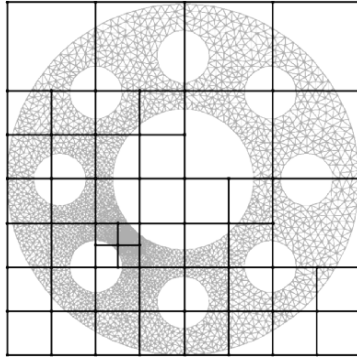


Fig. 6. An example mesh of the pipe model, with the corresponding quadtree. The grading function is given by (10) with $c = 0.3$.

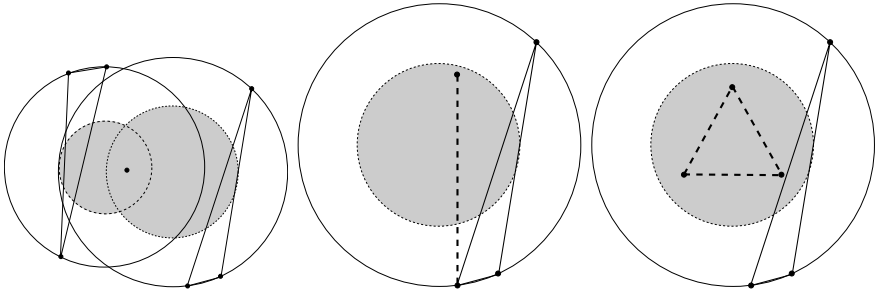


Fig. 7. Examples of the approaches for choosing Steiner points within selection disks of skinny triangles.

customized mesh optimization techniques. The goals of these techniques may include the following:

- minimizing the number of inserted points, see for example [24] and Subsection 2.2 here;
- eliminating slivers, see [9];
- splitting multiple poor quality triangles simultaneously, see Fig. 7(left).
- creating elongated edges in required directions, see Fig. 7(center);
- inserting more than one point, e.g., to create elements with specific shapes, see Fig. 7(right);
- satisfying other application-specific requirements.

In this paper, we extended our theoretical framework for parallel Delaunay refinement presented in [6] to work with custom point placement techniques. We used three different point placement methods: circumcenter, off-center and a new optimization-based method which allows to improve the size of the mesh by up to 20% and up to 5% over the first two methods, respectively.

Our current algorithm is limited to deterministic point selection; incorporating randomized point selection is left to the future research.

We presented the algorithm and the implementation of a parallel 2D graded guaranteed quality Delaunay mesh generator. The experimental results show that our code on a machine with two dual-core processors runs in 56% of the time taken by the fastest sequential code the Triangle [22]. By using a quadtree constructed in a specific way, we eliminated the need to solve the difficult domain decomposition problem. Our ongoing research includes the extension of the theory and of the implementation to three dimensions.

Acknowledgments

This work was supported (in part) by the following grants: EIA-0203974, ACI-0312980, and CNS-0521381. We thank the anonymous reviewers for helpful comments.

References

1. C. D. Antonopoulos, X. Ding, A. N. Chernikov, F. Blagojevic, D. S. Nikolopoulos, and N. P. Chrisochoides. Multigrain parallel Delaunay mesh generation: Challenges and opportunities for multithreaded architectures. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 367–376. ACM Press, 2005.
2. A. Bowyer. Computing Dirichlet tessellations. *Computer Journal*, 24:162–166, 1981.
3. S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *J. ACM*, 47(5):883–904, 2000.
4. A. N. Chernikov and N. P. Chrisochoides. Parallel guaranteed quality planar Delaunay mesh generation by concurrent point insertion. In *14th Annual Fall Workshop on Computational Geometry*, pages 55–56. MIT, Nov. 2004.
5. A. N. Chernikov and N. P. Chrisochoides. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. In *Proceedings of the 18th Annual International Conference on Supercomputing*, pages 48–57. ACM Press, 2004.
6. A. N. Chernikov and N. P. Chrisochoides. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In *Proceedings of the 14th International Meshing Roundtable*, pages 505–517. Springer, Sept. 2005.
7. A. N. Chernikov and N. P. Chrisochoides. Parallel guaranteed quality Delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing*, in print, May 2006.
8. L. P. Chew. Guaranteed quality mesh generation for curved surfaces. In *Annual ACM Symposium on Computational Geometry*, pages 274–280, 1993.
9. L. P. Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 391–393, 1997.
10. N. Chrisochoides and D. Nave. Parallel Delaunay mesh generation kernel. *Int. J. Numer. Meth. Engng.*, 58:161–176, 2003.

11. N. P. Chrisochoides. A survey of parallel mesh generation methods. Technical Report BrownSC-2005-09, Brown University, 2005. Also appears as a chapter in *Numerical Solution of Partial Differential Equations on Parallel Computers* (eds. Are Magnus Bruaset, Petter Bjorstad, Aslak Tveito), Springer Verlag.
12. S. Dong, D. Lucor, and G. E. Karniadakis. Flow past a stationary and moving cylinder: DNS at $Re=10,000$. In *2004 Users Group Conference (DOD-UGC'04)*, pages 88–95, 2004.
13. P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing. Application to Finite Elements*. HERMES, 1998.
14. L. Linardakis and N. Chrisochoides. A static medial axis domain decomposition for 2d geometries. *ACM Transactions on Mathematical Software*, 2005. Submitted.
15. L. Linardakis and N. Chrisochoides. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing*, 27(4):1394–1423, 2006.
16. G. L. Miller. A time efficient Delaunay refinement algorithm. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 400–409, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
17. G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: Generation, formulation, and partition. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 683–692. ACM Press, May 1995.
18. S. A. Mitchell. Cardinality bounds for triangulations with bounded minimum angle. In *Proceedings of the 6th Canadian Conference on Computational Geometry*, pages 326–331, Aug. 1994.
19. D. Nave, N. Chrisochoides, and L. P. Chew. Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28:191–215, 2004.
20. J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.
21. S. Schneider, C. D. Antonopoulos, A. N. Chernikov, D. S. Nikolopoulos, and N. P. Chrisochoides. Designing effective memory allocators for multicore and multithreaded systems: A case study with irregular and adaptive applications. Submitted to the Supercomputing Conference, 2006.
22. J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.
23. J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1–3):21–74, May 2002.
24. A. Üngör. Off-centers: A new type of Steiner points for computing size-optimal guaranteed-quality Delaunay triangulations. In *Proceedings of LATIN*, pages 152–161, Apr. 2004.
25. R. A. Walters. Coastal ocean models: Two useful finite element methods. *Recent Developments in Physical Oceanographic Modelling: Part II*, 25:775–793, 2005.
26. D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, 24:167–172, 1981.

Index of Authors and Co-Authors

- Aftosmis, Michael J. (375)
Alauzet, Frederic (191)
Atalay, F. Betul (57)
Bajaj, Chandrajit (73)
Bazilevs, Yuri (73)
Blacker, Ted D. (469)
Borouchaki, Houman (127)
Boulanger, Pierre (411)
Branch, John (411)
Bunin, Guy (3)
Burgos, Manuel A. (93)
Camarero, Ricardo (21)
Canas, Guillermo (289)
Carbonera, Calos (435)
Cascón, José Manuel (325)
Chernikov, Andrey (563)
Cherouat, Abdelhakim (127)
Chrisochoides, Nikos (563)
Coll, Narcís (39)
Colombi, Andrew (173)
Corral, Roque (93)
Dervieux, Alain (191)
Dompierre, Julien (35)
Eklund, Erling (393)
Escobar, José María (325)
Fernandez-Castañeda, Jaime (93)
Frey, Pascal (191)
Ghazialam, Hamid (393)
Giraud-Moreau, Laurence (127)
Gortler, Steven (289)
Goswami, Samrat (73)
Guerrieri, Marité (39)
Guibault, François (35)
Haimes, Robert (375)
Hariya, Masayuki (453)
Hart, John (173)
Hassan, Oubay (529)
Hecht, Frederic (259)
Hiro, Yoshimitsu (453)
Hudson, Benoit (339)
Hughes, Thomas (73)
Ito, Yasushi (147)
Jiao, Xiangmin (359)
Kataoka, Ichiro (453)
Kerr, Robert A. (469)
Koomullil, Roy (147)
Lague, Jean-Francois (259)
Lee, Y. K. (393)
Lim, Chin K. (393)
Lipnikov, Konstantin (163)
Liu, Jianfei (241)
Loseille, Adrien (191)
López, Carlos (93)
Miller, Gary (339)
Montenegro, Rafael (325)
Montero, Gustavo (325)
Morgan, Ken (529)
Mount, David (57)
Mukherjee, Nilanjan (309)
Murgie, Samuel (109)
Nishigaki, Ichiro (453)
Ni, Xinlai (173)
Owen, Steve J. (469)
Persson, Per-Olof (375)
Phillips, Todd (339)
Prieto, Flavio (411)
Rivara, Maria-Cecilia (543)
Rodríguez, Eduardo (325)
Sarrate, Jose (487)
Sazonov, Igor (529)
Sellarès, J. Antoni (39)
Shepherd, Jason F. (435)
Shih, Alan (147)
Si, Hang (509)

Simpson, Bruce (215)
Sirois, Yannick (35)
Soni, Bharat (147)
Staten, Matthew L. (469)
Sun, Shuli (241)
Tchon, Ko-Foa (21)

Vallet, Marie-Gabrielle (35)
Vardhan, Harsh (393)
Vassilevski, Yuri (163)
Wang, Yuanli (109)
Weatherill, Nigel P. (529)
Zhang, Yongjie (73)

Index by Affiliation

ALGOR, Inc.

Yuanli Wang (109)

Samuel Murgie (109)

Carnegie Mellon University

Todd Phillips (339)

Gary Miller (339)

Benoit Hudson (339)

College of William and Mary

Nikos Chrisochoides (563)

Andrey Chernikov (563)

École Polytechnique de Montréal

Ko-Foa Tchou (21)

Ricardo Camarero (21)

Marie-Gabrielle Vallet (271)

Julien Dompierre (271)

Yannick Sirois (271)

François Guibault (271)

Fluent France S.A.

Erling Eklund (393)

Fluent India Pvt. Ltd.

Harsh Vardhan (393)

Fluent USA

Y. K. Lee (393)

Chin K. Lim (393)

Hamid Ghazialam (393)
Georgia Institute of Technology
Xiangmin Jiao (359)

Harvard University
Guillermo Canas (289)
Steven Gortler (289)

Hitachi, Ltd.
Masayuki Hariya (453)
Ichiro Nishigaki (453)
Ichiro Kataoka (453)
Yoshimitsu Hiro (453)

Industria de Turbopropulsores S.A.
Roque Corral (93)
Jaime Fernandez-Castañeda (93)
Cárlos López (93)

INRIA
Frederic Alauzet (191)
Adrien Loseille (191)
Alain Dervieux (191)

Institute of Numerical Mathematics, Russian Academy of Sciences
Yuri Vassilevski (163)

Los Alamos National Laboratory
Konstantin Lipnikov (163)

Massachusetts Institute of Technology
Per-Olof Persson (375)
Robert Haimes (375)

NASA Ames Research Center
Michael J. Aftosmis (375)

Peking University

Jianfei Liu (241)

Shuli Sun (241)

Saint Joseph's University

F. Betul Atalay (57)

Sandia National Laboratories

Jason F. Shepherd (435)

Matthew L. Staten (469)

Robert A. Kerr (469)

Steve J. Owen (469)

Ted D. Blacker (469)

Technion

Guy Bunin (3)

UGS

Nilanjan Mukherjee (309)

Universidad de Chile

Maria-Cecilia Rivara (543)

Universidad Nacional de Colombia

Flavio Prieto (411)

John Branch (411)

Universidad Politécnica de Cartagena

Mamuel. A. Burgos (93)

Universitat de Girona

Marité Guerrieri (39)

J. Antoni Sellarès (39)

Narcís Coll (39)

Universitat Politecnica de Catalunya

Jose Sarrate (487)

Université Pierre et Marie Curie

Jean-Francois Lague (259)

Frederic Hecht (259)

Pascal Frey (191)

University of Alberta

Pierre Boulanger (411)

University of Alabama at Birmingham

Bharat Soni (147)

Alan Shih (147)

Roy Koomullil (147)

Yasushi Ito (147)

University of Illinois

John Hart (173)

Andrew Colombi (173)

Xinlai Ni (173)

University of Las Palmas de Gran Canaria

Gustavo Montero (325)

Rafael Montenegro (325)

José María Escobar (325)

Eduardo Rodríguez (325)

University of Maryland, College Park

David Mount (57)

University of Puerto Rico

Calos Carbonera (435)

University of Salamanca

José Manuel Cascón (325)

University of Technology of Troyes

Laurence Giraud-Moreau (127)

Houman Borouchaki (127)

Abdelhakim Cherouat (127)

The University of Texas at Austin

Yongjie Zhang (73)

Thomas Hughes (73)

Yuri Bazilevs (73)

Samrat Goswami (73)

Chandrajit Bajaj (73)

University of Wales Swansea

Nigel P. Weatherill (529)

Ken Morgan (529)

Igor Sazonov (529)

Oubay Hassan (529)

University of Waterloo

Bruce Simpson (215)

WIAS Berlin

Hang Si (509)