Proceedings

12th INTERNATIONAL MESHING ROUNDTABLE

September 14 – 17, 2003
Santa Fe, New Mexico

Prepared by:
Sandia National Laboratories
On behalf of the Meshing Roundtable Steering Committee we gratefully acknowledge and thank the following colleagues who provided technical paper reviews for this year's meeting:

J.P. Abelanet  
Basis Software, USA  
Vladislav Kraevoi  
Technion, Israel

Michael Aftosmis  
NASA Ames Res. Cent., USA  
Wra Kwok  
Ansys, USA

Pierre Alliez  
INRIA Sophia Antipolis, France  
Francois Labelle  
U. C. at Berkeley, USA

Steve Alter  
NASA Langley Res. Cent., USA  
Vladimir Liseikin  
ICT at Novosibirsk, Russia

Tim Baker  
Princeton University, USA  
Jan Meseth  
University of Bonn, Germany

Yih-En Andrew Ban  
Duke University, USA  
Todd Michal  
The Boeing Company, USA

Marshall Bern  
Palo Alto Res. Cent., USA  
Gary Miller  
Carnegie Mellon University, USA

Markus Berndt  
Los Alamos National Labs, USA  
Scott Mitchell  
Sandia National Labs, USA

Michael Borden  
Sandia National Labs, USA  
Neil Molino  
Stanford University, USA

Houman Borouchaki  
INRIA Roquencourt, France  
Vijay Natarajan  
Duke University, USA

Larisa Branets  
Univ. of Texas at Austin, USA  
Steven Owen  
Sandia National Labs, USA

David Cardozo  
Carnegie Mellon University, USA  
Steven Pav  
Carnegie Mellon University, USA

Graham Carey  
Univ. of Texas at Austin, USA  
Jay Palaniappan  
Univ. of Illinois at U.-C., USA

Siu-Wing Cheng  
HKUST, Hong Kong  
Kerem Pekkan  
Georgia Inst. of Technology, USA

Andrey Chernikov  
Col. of William & Mary, USA  
Philippe Pebay  
Sandia National Labs, USA

Nikos Chryssolouris  
Col. of William & Mary, USA  
Per-Olof Persson  
M.I.T., USA

John Dannenhoffer  
Syracuse University, USA  
Angel Plaza  
Universidad de Las Palmas, Spain

Tamal Dey  
Ohio State University, USA  
Mike Remotique  
Mississippi State University, USA

Neil Eccles  
Pointwise, USA  
Maria Rivara  
University of Chile, Chile

Peter Eiseman  
Program Development Co., USA  
Jamshid Samareh  
NASA Langley Res. Cent., USA

Kwang Ekki  
Duke University, USA  
Raimund Seidel  
Universitaet des Saarlandes, Germany

Jeff Erickson  
Univ. of Illinois at U.-C., USA  
Ariel Shamir  
IDC, Israel

John Fowler  
Comp. Geometry Consulting, USA  
Aila Sheffer  
Technion, Israel

Lori Freitag-Diachin  
Sandia National Labs, USA  
Mark Shephard  
Rensselaer Polytechnic Inst., USA

Pascal Frey  
INRIA Roquencourt, France  
Jason Shepherd  
Sandia National Labs, USA

Rao Garimella  
Los Alamos National Labs, USA  
Jonathan Shewchuk  
U. C. at Berkeley, USA

Paul-Louis George  
INRIA Roquencourt, France  
Krishnan Suresh  
University of Wisconsin, USA

Dammrong Guoy  
Univ. of Illinois at U.-C., USA  
Ertugrul Tacioglu  
U. C. L.A., USA

Igor Gusakov  
Univ. of Michigan, USA  
Tim Tautges  
Sandia National Labs, USA

Bob Haines  
M.I.T., USA  
Ko-Foa Tchon  
Universite de Montreal, Canada

David Hale  
Landmark Graphics, USA  
Shripad Thite  
Univ. of Illinois at U.-C., USA

Oubay Hassan  
University of Wales, UK  
David Thompson  
Mississippi State University, USA

Gerd Heber  
Cornell University, USA  
Harold Trease  
Pacific Northwest National Labs, USA

Weizhang Huang  
University of Kansas, USA  
Joe Tristano  
Ansys, USA

Benoit Hudson  
Carnegie Mellon University, USA  
Alper Ungor  
Duke University, USA

Keisuke Inoue  
IBM, Japan  
Paul Wawrzynek  
Cornell University, USA

Bill Jones  
NASA Langley Res. Cent., USA  
Rephael Wenger  
Ohio State University, USA

Bhautik Joshi  
Univ. of New South Wales, Australia  
David White  
Sandia National Labs, USA

Thomas Kalbacher  
Univ. of Tuebingen, Germany  
Frank Witzeman  
Air Force Res. Lab, USA

Steve Karman  
Univ. of Tennessee at C., USA  
Nick Wyman  
Pointwise, USA

Patrick Knupp  
Sandia National Labs, USA  
James Zhang  
Worcester Polytechnic Inst., USA

Paul Chew, John Steinbrenner, and Alper Ungor  
Technical papers co-chairs
Introduction

In 1992, Sandia National Laboratories started the Meshing Roundtable as a small meeting of like-minded companies and organizations striving to establish a common focus for research and development in the field of mesh and grid generation. Now after 12 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 relaxed atmosphere at each Roundtable provides an open and technically rich environment in which to share new ideas and technical advances.

The International Meshing Roundtable continues to focus on bringing together researchers and developers from academia, national labs and industry in a stimulating, open environment to share technical information related to mesh generation and general pre-processing techniques.

The Roundtable will consist of short course presentations, technical presentations from contributed papers, invited speakers, an invited panel of experts discussing topics related to the development and use of automatic mesh generation tools, Birds-of-the-feather sessions, and a poster session and competition.

The Program Committee would like to express gratitude to the meshing community for the response to the call for papers. This year, there were 56 submissions to the Roundtable, which is an increase of nearly 20% over the number of submissions received in previous years. During the Program Committee meeting in Santa Fe on June 23, 2003, the committee had the difficult responsibility of narrowing the number of papers accepted down to the 35 available time slots in the conference schedule. The 35 papers selected are scheduled for presentation at the conference and publication in the Roundtable Proceedings, and were selected based on their perceived quality and originality, and on the appropriateness to the theme of the International Meshing Roundtable.

Papers were sought that present original results on meshing and pre-processing techniques. In addition to our core topics in meshing related algorithms, we were also interested in obtaining technical papers that related analysis and application solution to the mesh generation process. Topics include but are not limited to:

- Adaptive meshing
- Anisotropic mesh generation
- CAD interface for meshing
- Geometry simplification, decomposition, and cleanup
- Tetrahedral/Hexahedral Hybrid meshing
- Industrial applications for complex geometries
- Mesh data formats and databases
- Mesh quality, smoothing, and optimization
- Mesh simplification and compression
- Meshing of parametric models
- Parallel meshing algorithms and software
- Scientific visualization
- Structured and unstructured grid generation
- Surface reconstruction
- Theoretical basis of mesh generation
- Volume and surface mesh generation

The 2003 Roundtable is steered by a committee taken from private industry, universities, and government laboratories.

Roundtable information can be found on the World Wide Web at URL http://www.imr.sandia.gov
Jason Shepherd, Committee Chairman, Short Courses and Panel Discussions
Sandia National Laboratories
P.O. Box 5800, MS 0847
Albuquerque, NM 87185
Phone: 505-284-6600
Fax: 505-844-9297
E-mail: jfsheph@sandia.gov
http://www.sandia.gov

Paul Chew, Technical Papers Co-Chair
Cornell University
Computer Science Department
Ithaca, NY 14853
Phone: 607-255-9217
Fax: 607-255-4428
E-mail: chew@cs.cornell.edu
http://www.cornell.edu

Eric Hjelmfelt, Co-Chair for Industry, BOF and Posters
Altair Engineering
1820 East Big Beaver Road
Troy, MI 48083
Phone: 248-614-2400
Fax: 248-614-2411
E-mail: eeh@altair.com
http://www.altair.com

Alper Üngör, Technical Papers Co-Chair
Duke University
Department of Computer Science, LSRC
Durham, NC 27708
Phone: 919-660-6503
Fax: 919-660-6519
E-mail: ungor@cs.duke.edu
http://www.cs.duke.edu/~ungor

John Steinbrenner, Technical Papers Co-Chair and Panel Discussions
Pointwise, Inc.
213 S. Jennings Ave.
Fort Worth, TX 76104
Phone: 817-377-2807 ext. 106
Fax: 817-377-2799
E-mail: jps@pointwise.com
http://www.pointwise.com

Jamshid Samareh, Co-Chair for Government, BOF and Short Courses
Bldg. 1192D, Room 179 MS 159
Hampton VA 23681
Phone: 757-864-5776
Fax: 757-864-9713
E-mail: j.a.samareh@larc.nasa.gov
http://www.larc.nasa.gov

Bernadette Watts, Financial Coordinator and Webmaster
Sandia National Laboratories
P.O. Box 5800, MS 0321
Albuquerque, NM 87185
Phone: 505-844-3936
Fax: 505-844-2415
E-mail: bmwatts@sandia.gov
http://www.sandia.gov

Anita Vasey, Conference Coordinator
Sandia National Laboratories
P.O. Box 5800, MS 0321
Albuquerque, NM 87185
Phone: 505-844-1338
Fax: 505-844-2415
E-mail: arvasey@sandia.gov
http://www.sandia.gov

VISIT OUR WEB SITE:
http://www.imr.sandia.gov
Agenda

**Sunday, September 14**

8:00-8:30 am    Short Course Registration  
8:30-10:00 am   Basic Mesh Generation Algorithms (Steve Owen)  
10:00-10:30 am  Break  
10:30-12:00 pm  Introduction to Geometry and CAD (Tim Tautges)  
12:00-1:30 pm   No-Host Lunch  
1:30-3:00 pm    Hexahedral Mesh Generation (Ted Blacker)  
3:00-3:30 pm    Break  
3:30-5:00 pm    Unstructured Meshing (Paul-Louis George)  
5:00 pm         Adjourn  
5:45-9:00 pm    Conference Registration and Reception

**Monday, September 15**

8:00-8:15 am    Registration  
8:15-8:30 am    Rob Leland, Welcome Address  
8:30-9:30 am    Tom Bickel, Keynote Address  
9:30-9:50 am    Break  
9:50-11:30 am   Session 1 (Parallel Sessions)  
                 1A – Geometric Models  
                 1B – Mesh Refinement  
11:30-2:00 pm  Luncheon and Poster Session – Catered at Hotel  
1:00-2:00 pm   Poster Judging and Voting  
2:00-3:00 pm   Shang-Hua Teng, Invited Speaker  
3:00-3:15 pm   Break  
3:15-4:55 pm   Plenary Session  
4:45-6:00 pm   Birds-of-a-feather Session  
6:00 pm        Adjourn
**Tuesday, September 16**

8:15-8:30 am  Opening Remarks
8:30-9:30 am  Michael Garland, Invited Speaker
9:30-9:50 am  Break
9:50-11:30 am  Session 2 (Parallel Sessions)
   2A – Surface Meshing
   2B – Quality
11:30-1:00 pm  Lunch – No host
1:00-2:00 pm  Jami Shah, Invited Speaker
2:00-2:10 pm  Break
2:10-3:50 pm  Session 3 (Parallel Sessions)
   3A Structured Meshing
   3B Adaptivity
3:50-4:00 pm  Break
4:00-5:30 pm  Panel Discussion
5:45-6:45 pm  Bus Run from Inn at Loretto to Bishop’s Lodge
7:00 pm  Dinner Banquet – Bishop’s Lodge
   Mark Shephard, Invited speaker
9:00-10:00  Bus Run from Bishop’s Lodge to Inn at Loretto

**Wednesday, September 17**

8:15-8:30 am  Opening Remarks
8:30-10:10 am  Session 4 (Single Session)
   Smoothing
10:10-10:30 am  Break
10:30-12:10 pm  Session 5 (Single Session)
   Mesh Topology
12:10 pm  Conference Closing Remarks
# Table of Contents

Acknowledgements .........................................................................................................2  
Introduction...................................................................................................................3  
Committee .......................................................................................................................4  
Agenda ............................................................................................................................5  
Table of Contents ............................................................................................................7  
Index of Authors & Co-Authors....................................................................................441  
Index by Affiliation ......................................................................................................443  
Distribution ..................................................................................................................445

---

## Welcome Address

Welcome Address..........................................................................................................15  
Dr. Robert Leland - Sandia National Laboratories

## Keynote Address

Keynote Address ..........................................................................................................17  
Dr. Tom Bickel - Sandia National Laboratories

## Session 1A – Geometric Models

1a.1  *Unified Geometry Access for Analysis and Design* ...........................................21  
R. Haimes, C. Crawford

1a.2  *Accessing CAD Geometry for Mesh Generation* .............................................33  
M. Beall, J. Walsh, M. Shephard

1a.3  *Parallel Generation of Unstructured Surface Grids* .........................................43  
U. Tremel, F. Deister, O. Hassan, N. Weatherill

1a.4  *Interpolation from a Cloud of Points* .................................................................55  
T. Baker
Session 1B – Mesh Refinement

1b.1  Mesh Refinement Based on the 8-Tetrahedra Longest-Edge Partition .......................... 67
       A. Plaza, M. Rivara

1b.2  Propagation Path Properties in Iterative Longest-Edge Refinement ......................... 79
       J. Suarez, A. Plaza, G. Carey

1b.3  When and Why Ruppert’s Algorithm Works ....................................................... 91
       G. Miller, S. Pave, N. Walkington

1b.4  A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra .......................................................... 103
       N. Molino, R. Bridson, J. Teran, R. Fedkiw

Invited Speaker

Generating Sliver-Free Well-Shaped Three Dimensional Delaunay Meshes (Invited) ................... 117
Shang-Hua Teng – Boston University and Akama Technologies Inc.

Plenary Session

1a  Meshing Complexity of Single Part Cad Models ..................................................... 121
       D. White, S. Saigal, S. Owen

1b  Compact Representations of Simplicial Meshes in Two and Three Dimensions ................. 135
       D. Blandford, G. Blelloch, D. Cardoze, C. Kadow

1c  A Mesh Warping Algorithm Based on Weighted Laplacian Smoothing ......................... 147
       S. Shontz, S. Vavasis

1d  A New Algorithm for Generating Quadrilateral Meshes and its Application to FE-Based Image Registration ........................................... 159
       S. Ramaswami, M. Siqueira, T. Sundaram, J. Gallier, J. Gee

Invited Speaker

Surface Approximation and Remeshing in Computer Graphics (Invited) .................. 173
Michael Garland – University of Illinois at Urbana-Champaign
Session 2A – Surface Meshing

2a.1 Meshing of Diffusion Surfaces for Point-Based Tensor Field Visualization .......................................................... 177
R. Sondershaus, S. Gumhold

2a.2 Geodesic-Based Surface Remeshing ............................................................... 189
O. Sifri, A. Sheffer, C. Gotsman

2a.3 Toward Quality Surface Meshing ................................................................. 201
J. Cabello

2a.4 Isotropic Remeshing of Surfaces: A Local Parameterization Approach .......................................................... 215
V. Surazhsky, P. Alliez, C. Gotsman

Session 2B – Quality

2b.1 An Adaptable Surface Parameterization Method .......................................................... 227
P. Degener, J. Meseth, R. Klein

2b.2 The Mesquite Mesh Quality Improvement Toolkit .......................................................... 239
M. Brewer, L. Diachin, P. Knupp, T. Leurent, D. Melander

2b.3 BSP-Assisted Constrained Tetrahedralization .......................................................... 251
B. Joshi, S. Ourselin

2b.4 Generalization of the Kantorovich Method of Dimensional Reduction .......... 261
K. Suresh

Invited Speaker

Evolution of Geometric Feature Recognition Techniques Through Four Generations (Invited) .......................................................... 273
Jami Shah – Arizona State University
Session 3A – Structured Meshing

3a.1 Automatic Nested Refinement – ATtechnique for the Generation of High Quality Multi-Block Structured Grids for Multi-Scale Problems Using Gridpro ................................................................. 277
K. Rajagopalan, P. Eiseman

3a.2 Multiple Stationary and Moving Boundary Handling in Cartesian Grids ....... 285
K. Pekkan

3a.3 Constructing Anisotropic Geometric Metrics Using Octrees and Skeletons ...... 293
K-F. Tchon, M. Khachan, F. Guibault, R. Camarero

Session 3B – Adaptivity

3b.1 Fully Automatic Adaptive Mesh Refinement Integrated Into the Solution Process ................................................................. 307
J. Tristano, Z. Chen, D. Hancq, W. Kwok

3b.2 A New Type of Size Function Respecting Premeshed Entities .................. 315
J. Zhu

3b.3 Automated Adaptive Forming Simulations ........................................... 323
J. Wan, S. Kocak, M. Shephard

3b.4 Anisotropic Mesh Adaptation for Transient Flows Simulations ................. 335
P. Frey, F. Alauzet

Session 4 – Smoothing

4a Multilevel Accelerated Optimization for Problems in Grid Generation .......... 351
M. Berndt, M. Shashkov

4b Combined Laplacian and Optimization-Based Smoothing For Quadratic Mixed Surface Meshes ................................................. 361
Z. Chen, J. Tristano, W. Kwok

4c A Local Cell Quality Metric and Variational Grid Smoothing Algorithm ........ 371
L. Branets, G. Carey

4d Increasing TAU3P Abort-Time via Mesh Quality Improvement .................. 379
N. Folwell, P. Knupp, M. Brewer
Session 5 – Mesh Topology

5a  Back to Edge Flips in 3 Dimensions ................................................................. 393
    P-L. George, H. Borouchaki

5b  Increasing the Number and Volume of Hexahedral and Prism
    Elements in a Hex-Dominant Mesh by Topological Transformations .......... 403
    S. Yamakawa, K. Shimada

5c  Topology Modification of Hexahedral Meshes using
    Atomic Dual-Based Operations ................................................................. 415
    T. Taugtes, S. Knoop

5c  Tetrahedral Mesh Generation From Segmented Voxel Data ...................... 425
    T. Udeshi

Panel Discussion and Birds-of-a-Feather Session

Notes: ........................................................................................................439-440
Welcome/Keynote Address
Robert Leland studied undergraduate electrical engineering with a minor in mechanical engineering at Michigan State University. He attended Oxford University as a Rhodes Scholar and studied applied mathematics and computer science. There he completed a Ph.D. in Parallel Computing in 1989 and continued his studies as a National Science Foundation Graduate Fellow. He joined the Parallel Computing Sciences Department at Sandia National Laboratories in 1990 and pursued work principally in parallel algorithm development, sparse iterative methods and applied graph theory. There he co-authored Chaco, a graph partitioning and sequencing toolkit that is widely used to optimize parallel computations and which was a finalist in the Wilkinson competition for the best numerical software released in a four year period. In 1995 he worked for the White House as one of fourteen White House Fellows appointed that year by the President. His primary responsibility was to advise the Deputy Secretary and Secretary of the Treasury on technology modernization at the IRS. Upon returning to Sandia in 1996, he lead the Parallel Computing Sciences Department, an R&D group of approximately 30 staff developing algorithmic technology and software tools in support of the Lab’s supercomputing efforts. He also served part time for several years as a member of Sandia’s Advanced Concepts Group studying long term national security issues. In 2001 he became responsible for Computer and Software Systems, a group of 80 staff members organized into four departments working on R&D in supercomputing hardware, operating systems, meshing and visualization.
Thomas C. Bickel
Sandia National Laboratories
1515 Eubank SE
Albuquerque, New Mexico 87123-0841
tbickel@sandia.gov
505-845-9301

Thomas Bickel attended Trinity Univ. in San Antonio, Texas. He received a Bachelors of Science degree with honors in Engineering Science in 1973. He attended the Univ. of Texas, receiving a Ph.D. degree in Chemical Engineering in 1978. His graduate work was in the field of nonlinear, integer programming.

Dr. Bickel has been employed at Sandia National Laboratories since 1978. From 1978 through 1990, as a Member of the Technical Staff, he performed research in areas of chemical kinetics, heat transfer and multiphase fluid dynamics of fossil fuels, and applied geophysics of petroleum reservoirs. Dr. Bickel left Sandia for a brief period in 1981 to become manager of the Engineering Dept. at Vedette Energy Company. He was responsible for the thermal and chemical performance of the Down-Hole Steam Generator for secondary and tertiary oil recovery. He subsequently returned to Sandia and was principal investigator on the joint Occidental Oil Shale/DOE Modified In Situ Oil Shale retorts 7 and 8, geophysical simulation and modeling of the surface electric potential of oil reservoirs, and high-temperature superconductivity material science research and application development.

In 1990 Dr. Bickel was appointed a Distinguished Member of the Technical Staff in recognition of his work in energy research. In 1991 Dr. Bickel became a Division Supervisor with responsibility for High-Temperature Superconductivity and Optical Properties Research. Subsequently, he was given responsibility of managing photovoltaic research on single-crystal and multicrystalline silicon materials and concentrating photovoltaic systems engineering. In 1995 Dr. Bickel was appointed an American Society of Mechanical Engineers Federal Government Fellow attached to the Secretary of Energy Advisory Board in Washington, DC. He was the senior technical advisor to the Task Force on Strategic Energy Research and Development chaired by Dr. Daniel Yergin. At the completion of the Yergin Task Force, he became technical advisor to Deputy Secretary Charles Curtis and Pete Didisheim on the DOE Laboratory Operations Board, developing the Strategic Laboratory Missions Plan for the 28 DOE laboratory complex.

In 1996 Dr. Bickel returned to Sandia as the Manager of the Thermal Sciences Dept., guiding work in massively parallel, computational simulation of the thermal response of nuclear weapons with programmatic responsibility for fundamental research in thermal sciences. In 1997 Dr. Bickel became the Deputy Director of the Engineering Sciences Center with responsibility for the stewardship of Engineering Sciences research and development at Sandia. He was subsequently appointed a Senior Manager. In 2000, Dr. Bickel was promoted to Director of Engineering Sciences at Sandia. In this role he has responsibility for the stewardship of R->D->A of the engineering disciplines at Sandia. With over 180 professional staff, he is responsible for the development of massively parallel computational mechanics software at Sandia as well as the experimental validation of the engineering models of the codes.
UNIFIED GEOMETRY ACCESS FOR ANALYSIS AND DESIGN

Robert Haimes¹ Curran Crawford ²

¹ Aerospace Computational Design Laboratory, Massachusetts Institute of Technology, U.S.A
haimes@mit.edu

² Aerospace Computational Design Laboratory, Massachusetts Institute of Technology, U.S.A
ccrawfor@mit.edu

ABSTRACT

This paper presents a comprehensive approach for CAD based geometry handling in support of single and multidisciplinary analysis and design. Unlike previous schemes, the model presented here allows for hands-off automated meshing, a requirement for design studies. Multidisciplinary analysis is handled through solid modeling constructs, using the geometry as a transfer media. For design applications, the engineer specifies the design space directly by the parameters of the CAD model. Key defining values in the model specified during part synthesis are later exposed to carry out design studies and optimization. Proper layout and definition of the CAD model facilitates design changes and use across all stages of design. A turbine blade model suitable for detailed aero/structural/thermal analysis is used to illustrate these concepts.

Keywords: CAD interfaces, Parametric build, Feature tree, Master-Model, Analysis, Design, Optimization

1. INTRODUCTION

There are a number of projects [1, 2, 3] and some commercial software [4, 5, 6] that facilitate building complex, multidisciplinary analysis, optimization and design systems. These software products either provide closed systems or can take existing applications and through encapsulation allow the building of complex custom applications. This is primarily a top-down view where each module requires a number of inputs and produces some output. The infrastructure’s function is to control the flow of data between modules. None of these systems can alleviate a bottleneck in a discipline, and at the simplest level these software systems just minimize the need for input and output files. These efforts primarily deal with discrete design variables, objectives and constraint functions, rather than complete geometric descriptions of the system.

When analyzing (or designing/optimizing) some physical object that will ultimately be manufactured, it is common practice to create a geometric definition in a CAD system. Erroneously, the CAD model is often only a final repository of the design details. For most disciplines (all except for Structural Analysis), the bulk of time (and human intervention) is expended in transforming this data into a mesh that is suitable for the physics to be analyzed. In some cases, such as for Computational Fluid Dynamics (CFD), this time period can be weeks to months when first setting up a new complex part. Clearly, one cannot hope to do a parametric study, no less design optimization, under these conditions. The problems associated with grid generation stem from a number of sources, with the most obvious being the use of inadequate file ‘standards’ for transmitting the geometry.

The commonly used IGES file format contains data that is defined as disjoint and unconnected surfaces and curves; that is, it only contains geometry with no notion of topology. 3D meshing software ultimately requires a closed “water tight” model. Much effort is therefore needed to take the geometric data, trim the curves and surfaces, and then deduce the topology. A common side effect of this process is the creation of many “sliver” surfaces to close the model. This process is particularly onerous for all modern CAD systems where solid modeling is fully supported. Before translation, the part was a proper closed solid with defined topology; the CAD system having been
responsible for verifying the necessary conditions; in the translation to IGES this important characteristic is needlessly lost.

The STEP file format supports topology as well as geometry. This is therefore the preferable file type to use for the transmittal of CAD neutral data. Surprisingly, this format is seldom used in practice. This may be due to the fact that constructing a STEP reader is complex and requires a complete solid modeling geometry kernel to deal with the data. Also, transferring data via STEP is not without its own set of problems. Each CAD system uses a different mathematical formulation to represent the same types of surfaces and also have different tolerances for closure. After reading a solid part, one may find the model is now open, again requiring some form of patching. An additional problem with data transmitted via IGES and STEP formats are that the writers rarely are in strict adherence to the “standard”. Standards can only be as good as the extent to which they are followed.

These problems do not exist for some native CAD/Analysis interfaces. Ansys, MARC, and Patran (as well as other widely used commercial codes) couple directly to CAD systems. These analysis codes are designed to be activated from within the graphical user interface of the CAD package. The part’s geometry and topology are queried directly from the CAD’s geometry kernel using an Application Programming Interface (API). Curve and surface queries through the originating CAD system insure that points placed on these entities match the part being meshed. These direct interfaces are inherently limiting, in that a module must be pre-existing for the analysis code of interest (i.e. what CFD codes are run from within the CAD environment). Furthermore, the full functionality of the analysis package may not be available from within the CAD system, limiting the user’s options for exploiting the software’s full capabilities.

Even if all of the analysis codes in an integrated suite have no geometry-handling issues the following question needs to be asked: is each application seeing the same representation of the design? Some codes with direct CAD interfaces still translate the geometry into a simplified or uniform definition before using the data to generate a mesh. Many organizations also maintain multiple models of the same part within different divisions, introducing unnecessary overhead.

2. A UNIFORM DIRECT INTERFACE

The grid generation techniques used in a discipline such as CFD are more varied and complex than the (relatively coarse) tetrahedral meshes used for Structural Analysis. Due to the economics (the enormous amount of work) of coupling to each CAD system, this approach is prohibitive for other “smaller” disciplines. A direct vendor neutral API would allow an analysis builder access to the CAD data without programming directly for each system. Examples of this approach include OMG CAD Services [7], CADScript [8], and CGM [9]. CAPRI [10] (Computational Analysis Programming Interface) also provides a solution to the CAD dependency issue. Coupling to any supported CAD package is both unified and simplified by using the CAPRI definition of geometry (with topology) and its API to access the geometry and topological data.

CAPRI’s CAD-vendor neutral API is more than just an interface to CAD data; it is specifically designed for the construction of complete analysis suites. A ‘Geometry Centric’ approach allows access to the CAD part from within all sub-modules (grid generators, solvers and post-processors), facilitating such tasks as node enrichment by solvers and designation of mesh faces as boundaries (for the solver and the visualization system). CAPRI supports only manifold solids at its base level, eliminating problems associated with manually closing surfaces outside of the underlying CAD kernel. Multidisciplinary coupling algorithms can use the actual geometry as the medium to interpolate data from differing grids.

One clear advantage to this approach is that the geometry never needs to be translated and hence remains simpler and closed. The other major advantage is that writing and maintaining the grid generator (coupled to the CAD system) can be done once through the API; all of the major CAD vendors are then automatically supported.

2.1 CAD Representation of Geometry

CAD systems have a tolerance that determines the meaning of “closure” for solids. This means that the Nodes that bound an Edge are probably not on the underlying curve; Edges that bound a Face (through the Loops) do not necessarily sit on the supporting surface. All that is required is that the bounding objects be within a specified tolerance of the higher dimensioned entity. Therefore, for any precision higher than the tolerance, gaps and overlaps may exist in the geometry definition. This tolerance is generally much larger than those associated with double precision floating-point arithmetic (e.g. the default relative tolerance for Pro/ENGINEER is only $10^{-2}$).

In order to deal with the gap and overlaps, most CAD-based applications must “fix” the geometry. This usually entails translating the geometric definition to another (simpler) representation where the bounding entities fall closer to, or on the object. This type of translation has a variety of side effects, including:
• Inconsistency: Not querying the same geometry. Since the geometry has changed, the representation is different than in the CAD system.

• Complexity: At times additional Faces are required to close the model. There is no way to predict how many of these “sliver faces” may need to be introduced; moreover, slivers can cause significant problems for grid generators.

• Automatic: There are always situations that cannot be healed in a hands-off manner. The requirement of user intervention is problematic for any fully automated process such as design optimization.

CAPRI’s perspective is that the geometry in the CAD system is truth and should not be modified (though CAPRI may modify the topology). Therefore fixing the CAD’s model is no longer part of the analysis procedure.

2.2 An Associative Triangulation

Early in the design and implementation of CAPRI, it became obvious that providing an API only giving the programmer access to the geometry and topology of a solid part was insufficient. The burden of deciphering the CAD data and attempting to generate a discrete representation of the surfaces required for mesh generation was too great. Fortunately, many grid generation systems (used in CFD and other disciplines) can use STL (Stereo Lithography) files as input. Combining a discretized view of the solid part as well as it’s geometry and topology can provide a complete, and easier to use, access point into the CAD data. A tessellation of the object that contains not only the mesh coordinates and supporting triangle indices but other data, such as the underlying CAD surface parameters (for each point), as well as the connectivity of the triangles, assists in traversing through and dissecting the CAD representation of a part. This is a fundamental difference between CAPRI and the other Direct Interface implementations.

An important aspect of CAPRI is that it provides CAD vendor neutral access to all of the data obtained from the models that is to be passed back to the application. The triangulation generated by CAPRI is guaranteed to be “watertight”, regardless of the CAD kernel in use. Some CAD system geometry kernels can provide data of this quality (i.e., UniGraphics, Parasolid, CATIA and ComputerVision). Other CAD systems can provide the data, but it is not of sufficiently high quality to use. (For example, Pro/Engineer requires one to buy Pro/MESH to get a closed triangulation.) Finally, SDRC’s Open I-DEAS API does not provide access to a triangulation at all. The fact that not all CAD systems provide such a tessellation has forced the development of a surface triangulator within CAPRI for CAD solid parts that does meet all of the quality requirements.

It should be noted that CAPRI’s tessellations are not intended as the starting point for computational analysis (though they could be used in some cases). CAPRI sees only geometry, and it cannot anticipate the smoothness, resolution or other requirements of the downstream application(s). The triangulations approximate the geometry only; some processing of the tessellation is expected in order to refine the triangulation to a state suitable for the physical problem being investigated. The triangulation can be enhanced through either physical or parameter space manipulation, using point “snap” and \((u,v)\) surface evaluations provided by the CAPRI API [11]. The triangulation technique used within CAPRI displays the following characteristics [12]:

• Robust. It is imperative that the scheme work for all possible topologies and provide a tessellation that can be used.

• Correct. The triangulation is of no use if it is not true to the CAD model. The tessellation must be logically correct; i.e. provide a valid triangulation in the parameter space \((u,v)\) of the individual surface. It must also be geometrically correct; i.e. depict a surface triangulation that truly approximates the geometry. This involves ensuring all facets have a consistent orientation with no creases or abrupt changes in triangle normals. Correctness in both physical and parameter space allows CAPRI based application enhancement schemes to operate in either or both.

• Adjustable. To minimize the post-processing of CAPRI’s tessellation for a specific discipline or analysis, some a priori adjustment of the resultant quality is available. It must be noted however, that any criteria may not be met (especially near the bounds of a CAD object) due to issues of closure and solid model accuracy. This goal may conflict with the more important characteristic of being watertight and having a smooth surface representation.

• No geometric translation. To truly facilitate hands-off grid generation, anything that requires user intervention must be avoided. All data maintained within CAPRI is consistent with the CAD’s solid model representation. An alternate or translated representation is not used, because then the result will be something different than resides within CAD.
3. MULTIDISCIPLINARY ANALYSIS

The classic example of multidisciplinary analysis is the modeling of fluids/structures interactions. In this case, the domain of interest is clearly demarcated between regions containing the fluid and the rest of the volume representing the structural components. The interaction effects are usually handled at the bounds of the shared domain (the interface), represented by some surface (or set of surfaces) found in the solid geometry (usually of the structural component). The interaction can be simulated by adjusting the boundary conditions at the interface surface(s) from the computed results of the other discipline. Most commonly, pressure values on the structure are obtained from the CFD calculation, causing regions of the structure to deflect. This movement then effects the fluid simulation, which generates a new pressure field, et cetera. In most cases, the CFD simulation can take the structural deflections and morph the existing mesh so that the fluid domain need not be regenerated. This still requires careful handling of the following issues:

3.1 Geometry

It is important that each discipline see the same interface. There are two reasons for this: (1) so that each analysis is examining results from the same part; (2) so that the interpolation task can be done with some degree of accuracy.

CAPRI supports multidisciplinary analysis tasks by supplying API calls that implement solid Boolean operations. Since the intersection, subtraction and union operations are performed by the CAD system’s geometry kernel, the end results contain the same geometry fragments. For example, if the solid is of an airplane, subtracting it from a larger box will produce the fluids domain. The surfaces that make up the bounds of the aircraft will be identical in both the original and the new solid (though the normals will be opposite).

3.2 Interpolation

CAPRI facilitates multidisciplinary coupling by providing interpolation routines. These routines work on sets of Faces (the topological equivalent of surfaces) called Boundaries. Information associated with the points in the Boundary can be scalar, vector or general state-vector. This data can be interpolated to the points of the mating Boundary found in the other Volume. CAPRI provides a number of API functions that help manage the Boundary information so that the interpolation can be a single call.

4. WRITING GEOMETRY

At beginning of the CAPRI project there was always the notion that design functionality would be supported. At the time, it was thought that CAPRI would support the direct construction of 3D solid geometry in order to allow for the modification of said geometry. As the readers were being implemented, it became obvious that this would not be possible. Each CAD system deals with the low-level geometry construction in very different manner. There was not a common vendor neutral perspective on direct construction. In fact, only those systems based on geometry kernels (and allowing the use of the kernel) could perform construction. Therefore, only if one programmed in Parasolid, ACIS or OpenCASCADE could this kind of construction be performed.

As it turns out, this limitation was fortunate; another type of construction was required that could be driven by an API. Most modern CAD systems support the Master-Model concept of representing an object. A Master-Model describes the sequence of topological operations to build the geometry of a solid model. At a basic level, it is an ordered list of extrude, revolve, merge, subtract and intersection operations. CAD systems support more meaningful abstractions, such as blends, fillets, drilled holes and bosses. When the CAD model is regenerated, the operation list is interpreted by the CAD system to sequentially build the geometry of the part. This gives the operator the ability to construct a family of parts (or assemblies) by building a single instance. Many of the operations used in the construction can be controlled by parameters that may be adjustable. By changing these values, a new member of the family can be built by simply following the prescription outlined in the Master-Model definition.

The recipe may be simple, like a serial collection of primitive operations, but can also be complex, where operations are performed on previously or temporarily constructed geometry. The representation of this construction in most CAD systems is the form of a tree, usually referred to as the “Feature Tree”. By sup-
porting this method of construction, a direct API can provide both simple and powerful access to the CAD system. This approach is clearly outside the static view traditionally held of geometry. That is, this kind of access and control is not possible from any type of file transfer.

Within CAPRI, this tree is presented to the programmer in the form of “branches”. Each of these entities has an index to identify where in the tree the reference is made. All indices are relative (that is they can occur anywhere in the tree – the assignment is usually given during initial parsing of the CAD internal structures). There is a special branch always given the index zero, the root of the tree. Therefore, the entire tree may be traversed starting at the root and moving toward the end of each branch. The branches terminate at leaves (branches that do not contain any children). To aid in traversing the tree toward the root the parent branch is always available. Unlike simple binary trees, a branch in CAPRI’s Feature Tree may contain zero or more children.

Currently, the structure of tree itself cannot be edited from within CAPRI (though this may change at some future release). However, some branches may be marked “suppressible” – these features may be turned off, in a sense removing that branch (and any children of the branch) from the regeneration. This is powerful in that it allows for defeating the model, so that it may be made appropriate for the type of analysis at hand. For example: if fasteners are too small for a fluid flow calculation, they may be easily suppressed (if the Master-Model was constructed with this in mind). After part regeneration the resultant geometry would be simplified and the details associated with the fasteners would not be expressed.

Parameters are those components of the Master-Model that contain values (and should not be confused with the geometric parameterization). CAPRI exposes all of the adjustable (non-driven) parameters found in the model. This is a separate list from the Feature Tree, but references back to the associated branch features where the values are used or defined. Parameters may be single or multi-valued and can be Booleans, integers, floating-points or strings.

This CAD perspective on parametric building of parts and assemblies is fine for driving the part using simple parameters but is problematic for shape design. For example, simple parameters may be used to define the plan-form of an aircraft, but are difficult to use to define the airfoil shape of the wing and tail components. The designer would need to expose the curve/surface definition at a very fine and detailed level (i.e. knot points as the parameters) to allow for the exact specification of shapes. CAPRI avoids placing this burden on the CAD designer by exposing certain curves as multi-valued “parameters”. These curves are obtained from independent sketched features in the model that later are used in solid generation as the basis for rotation, extrusion, blending and/or lofting. The curves can be modified, and when regenerated, the new part expresses the changed shape(s). This functionality is critical for shape design in general and specifically aerodynamic shape design.

5. AN UPSTREAM VIEW

The traditional design process (in many fields) starts from a conception stage where no actual geometry may be specified, to a final design where the part is fully realized down to the finest details. In a multidisciplinary design setting, one discipline may set some “parameters” before passing its information along to the next. Only when there is the requirement for more detailed analysis requiring geometric properties will the design be fleshed out and placed into a CAD system in a solid representation. It should now be clear that if the design process changes from this traditional situation to one where the designer predefines the part’s intent and possible expression (through a Master-Model definition) the following becomes feasible:

- Consistency. Each phase in the design process uses the same suite (or a subset of the suite) of parameters. Any parameter value change that produces differing geometry can be viewed by another stage in the process without writing and reading the geometry in files. The CAD part, regenerated with a particular set of parameter values and Feature Tree suppression statuses, uniquely describes the geometry.

- Data Repository. The CAD system and Product Data Management (PDM) software can be used to track and maintain the design. Also, because the design is in the CAD system from the beginning, issues of manufacturability can be easily addressed early on and unrealistic expressions kept out of the design space.

- Use of defeating to go from preliminary to final design. If the Master-Model is built in a manner that reflects the design process, then traversing the stages in the process is just a matter of adjusting the Feature Tree. During preliminary design where the resultant geometry may be simple (or nonexistent) most of the branches of the tree are suppressed. As the design approaches the final intent, more and more of the details of the part are expressed by unsuppressing the branches. This will also require setting various parameters as their effects become active.
Figure 1.a The source model was constructed using Pro/ENGINEER. A turbomachinery blade model is manipulated to illustrate design dynamically constrained applications. This approach be modeled inside an assembly, especially for aero-fully capture the design intent to achieve an even higher level of modularity and to more through to the finest manufacturing details. To address the design, starting with the most basic definitionModel effectively captures the decomposed intent of the complete representation of the blade can be seen in (Figure 2.a).

The fillets that merge the aerodynamic portion of the full part as seen Figure 1.a from this instance can clearly be seen. Of these parts should the fillets later be deleted from the design. This may seem excessive until one realizes that the turbine blade is internally cooled. Figure 1.b displays the same part as seen in Figure 1.a except that the exterior and interior Faces that bound the pressure surface of the blade have been removed, exposing the interior of the blade. The triangulation of the interior of the suction surface wall has been highlighted to better display the structure. The internal flow enters through a hole that can be seen on the left hand side of the root (under the hub surface). The fluid travels up the left (leading edge), being tripped by the ridges that can be seen (and do not completely block the passage). The cooling air continues to follow the serpentine until it is back down at the root in the center of the blade. The flow leaves the slot in the trailing edge as it continues to be mixed by the pins blocking the exit.

The CAD model should be constructed so that Master-Model effectively captures the decomposed intent of the design, starting with the most basic definition through to the finest manufacturing details. To achieve an even higher level of modularity and to more fully capture the design intent, complex parts should be modeled inside an assembly, especially for aerodynamically constrained applications. This approach will be illustrated in the next Section.

6. A TURBINE BLADE EXAMPLE

A turbomachinery blade model is manipulated to illustrate ideas expressed in the previous Section. The complete representation of the blade can be seen in Figure 1.a. The source model was constructed using Pro/ENGINEER.

It should be noted that all of the figures in this Section were generated from CAPRI’s triangulation directly and without any massaging. Also, all modifications to the Master-Model were done using CAPRI functionality and not Pro/ENGINEER’s Graphical User Interface.

This solid part model contains 134 parameters. 12 of these parameters are the curves gleaned from four sketches in the model. The sketches are located at the root, tip and two mid-span locations on the blade, and each contains curves used to define the suction and pressure surfaces, in addition to a third curve defining the camber-line (metal turning angles at the leading and trailing edges connected by a tangent spline). 13 are floating point and integer named parameters, which include things like trailing-edge thickness, wall thickness, and number of blades in the row, etc. The remaining parameters are what Pro/ENGINEER calls dimensions. These are unnamed values that are required by the sketcher to fully determine sections (CAPRI also exposes these values when not driven and treats them the same as Pro/ENGINEER parameters).

The Feature Tree for the model contains 233 nodes. This may seem excessive until one realizes that the turbine blade is internally cooled. Figure 1.b displays the same part as seen in Figure 1.a except that the exterior and interior Faces that bound the pressure surface of the blade have been removed, exposing the interior of the blade. The triangulation of the interior of the suction surface wall has been highlighted to better display the structure. The internal flow enters through a hole that can be seen on the left hand side of the root (under the hub surface). The fluid travels up the left (leading edge), being tripped by the ridges that can be seen (and do not completely block the passage). The cooling air continues to follow the serpentine until it is back down at the root in the center of the blade. The flow leaves the slot in the trailing edge as it continues to be mixed by the pins blocking the exit.

Figure 1.c depicts just the internal void of the blade displayed as a solid. Here the entire internal flow path can clearly be seen.

Figure 2 shows the turbine blade at various levels of feature suppression. Figure 2.a displays the turbine blade with all of the internals removed (i.e. this is a solid blade). In this view, the only visible difference is the absence of a hole at the root. It should be noted that the solid seen in Figure 1.c was generated in CAPRI by performing a solid Boolean subtraction of the full part as seen Figure 1.a from this instance (Figure 2.a).

The fillets that merge the aerodynamic portion of the blade to the hub and tip casements have been removed.
in Figure 2.b. Due to the view angle this difference can only be seen down at the root/blade juncture. Figure 2.c depicts the blade in its simplest solid form; all extraneous details of the hub and tip have been removed. This geometry could be used for preliminary and/or aerodynamic design.

When performing grid generation for CFD, one needs a solid representing the flow regime. A fluid volume suitable for analysis of the turbine blade using periodic boundary conditions is shown in Figure 3.a. This step illustrates the utility of using assemblies during part construction. The solid model of the blade is contained as the second part of an assembly. Analogous to parts, assemblies are also described by Master-Models. In this case, the first element in the assembly Master-Model is a specialized type of part, referred to as a skeleton in Pro/E parlance. The distinction is that it contains no solid geometry, only datums, which can include non-manifold surfaces.

The skeleton Master-Model contains two revolved surfaces, one for each of the inner and outer walls of the turbine stage annulus. The solid blade part then references these two surfaces to construct the root and tip shrouds on the blade, adding material to go from Figure 2.c to Figure 2.b. A third part was created in the assembly, again referencing the annulus surfaces to create a complete manifold solid over the entire circumference of the annulus, both upstream and downstream of the blade. The sketches of the metal turning angles in the blade part are then referenced to create the offset sides of the periodic wedge seen in Figure 3.a, by slicing the complete annulus. The dimensions of the wedge are determined by a parameter in the blade Master-Model specifying the number of blades in the row.

Modularizing the CAD model in the above manner effectively separates the design intent. The flow annulus design becomes an independent operation, while remaining a driving factor in the blade design. In this way, consistency is also maintained between the parts. The references used in each part are created from the published features of the other parts, affording further control over the model composition and avoiding non-robust topological constructs. For example, the
rotation axis of the rotor is made a common reference point between each part. As the design matures, the upstream and downstream rows of blades and stators will be able to reference the same annulus definition, again maintaining coherence between the CAD models.

In order to produce the complete CFD domain, as seen in Figure 3.b, the blade seen in Figure 2.c is subtracted from the blank passage of Figure 3.a. Figure 3.b is displayed with the periodic surface closest to the viewer stripped away to show the blade cut-out.

If it was desirable to compute on the complete flow regime (both internal and external), a solid Boolean union of the objects seen in Figure 3.b and Figure 1.c could be used. The result can be seen in Figure 4.a. This could also have been performed by subtracting the complete turbine blade (Figure 1.a) from the passage seen in Figure 3.a, but the inflow region (seen below the passage) would be absent, because the lower surface of the wedge did not extend below the hub surface definition.

Figure 4.b displays the effect of changing a parameter value. In this case the number of blades in the blade row was reduced by half. The result is that the casing treatments grew to accommodate the requirement of completing the circumference.

7. DISCUSSION

The approach articulated in this paper, with its precise control over the geometry, is quite powerful (as displayed in the previous Section). One can now have the CAD system central in an automated design optimization loop, without the encumbrance of user interaction. This approach does not replace the top-down approach but augments it well, producing a system that also has a consistent bottom-up geometry methodology.

It should be reiterated that when using a direct interface there is CAD vendor independence. The analysis modules are isolated from the details of the underlying back-end system. Also when the modifications to the model are made through the API, by the analysis application, there is no need to step back out to interactively use the CAD system (hence interrupting the
design process).

There remain some issues that need attention before this view of design can be considered complete and realized. They are discussed below:

### 7.1 Regeneration and Consistency

The Master-Model method of generating parts has little ability to control the part’s topological outcome. In fact, the resultant geometry may not be explicitly specified in any branch of the Feature Tree. This has the side effect that the topology may fundamentally change, even when two parts may be almost identical. Consider a wing/fuselage configuration where the fuselage is a cylindrical surface that due to the CAD modeller happens to be split along an axis generally aligned with the wing juncture. Assume the Master-Model has a parameter that controls the vertical mating position of the wing relative to the fuselage center-line, where zero aligns the wing with the seam of the cylinder. Consequently, the fuselage is maintained in CAD as 2 Faces each with half of the wing cutout. If the parameter is now set to a value greater than 1/2 of the wing thickness, the underlying topology is completely altered even though the parts differ only in a minor way. In this latter case, one of the two half cylinders is left untouched, while the other one ends up with a complete cutout (hole) where the wing fuses with the cylinder.

Topological inconsistencies are not a problem for most types of analysis, but can introduce difficulties in some cases. When using Adjoint methods (or other gradient approaches), one needs to compute geometric sensitivities for parameter changes. In order to perform this task, one determines the sensitivities either analytically or by computation. Because it is not possible to differentiate through the CAD system, the easiest way to get the sensitivities is to difference two instances. But, how does one track point movement from one part to a position in another instance when they differ at the topological level? Stated another way: how does one smoothly map from a start position to another position when components of the mapping can appear and disappear?

Assuming that there was some consistency at the topological level, tracking points is still problematic. The methods that provide a discrete view (i.e. triangulation) have no regard for history. All schemes place points to best satisfy some geometric criteria. Two parts that are almost identical will most likely display different tessellations, described by different densities of triangles. There has been some success with a technique that scribes a consistent quadrilateral patch topology over the CAD geometry [13].

It is possible to get sensitivities analytically if the construction is performed in a geometry kernel and control is exerted over the type of surfaces used. The result could be incorporated as an initial component of the Feature Tree. This has been done for turbomachinery aerodynamic design using Parasolid for construction[14]. UniGraphics could import the result of this work so that the fundamental blade shape could be defined outside of CAD as the rest of the model could be built upon the initial aerodynamic
blade shape. This would obviously make a more complex (and less consistent) system and detracts from the overall utility of the approach by limiting generality.

7.2 Tagging

In order to completely setup analysis codes, boundary conditions must be set. Some of the values are linked to the geometry (i.e., points of load application, surface(s) representing inflow, outflow or solid walls, etc.). Because the Feature Tree does not directly represent geometry, but creates the part, it is not clear from the tree perspective how to tag or map the resultant geometry.

This could be accomplished at the Face level by querying the CAD system for the branch of the Feature Tree that is responsible for generating the surface.

7.3 CAD Model Construction Issues

Top-down design is often referred to in CAD circles. All too often, proper techniques are not followed since proper practice requires slightly more effort up front in the design process. It is far too easy to create a bad one-off model that is virtually useless and must be thrown away if later changes are required. Engineers must be educated in the proper methodology for building robust CAD models that are extensible and can hold together through repeated design alterations and regeneration cycles.

7.4 Organizational changes

By far the most difficult challenge in having this approach adopted is not a technical one. Before the benefits articulated in this paper can be realized the process of design as it currently stands (in organizations that manufacture) needs to change. This is a difficult task in that the organization currently has a process that works no matter the efficiency and time-to-market for new designs.
ACKNOWLEDGEMENTS

The authors would like to thank the following CAPRI “usual suspects” for their prodding, assistance and/or support in this effort: Michael Aftosmis (NASA Ames Research Center), Juan Alonso (Stanford University), John Dannenhoffer III (Syracuse University), Peter Gage (NASA Ames Research Center), William T. Jones (NASA Langley Research Center), Al Magnuson (ICEM-CFD/ANSYS) and James Reuther (NASA Ames Research Center).

References


ACCESSING CAD GEOMETRY FOR MESH GENERATION

Mark W. Beall¹, Joe Walsh², Mark S. Shephard³

¹Simmetrix Inc., Clifton Park, NY., U.S.A. mbeall@simmetrix.com
²Simmetrix Inc., Clifton Park, NY., U.S.A. jwalshl@simmetrix.com
³Rensselaer Polytechnic Institute, Troy, NY., U.S.A. shephard@scorec.rpi.edu

ABSTRACT

One of the major issues of mesh generation today is access to CAD geometry in an accurate and efficient manner. This paper will provide an overview of the process of accessing CAD geometry for mesh generation and will review several of the issues associated with accessing CAD geometry for mesh generation. This paper will also evaluate alternative techniques for accessing CAD geometry and review how these techniques address or do not address the issues related to CAD geometry access for mesh generation. The techniques for CAD geometry access to be reviewed include: Translation & Healing, Discrete Representations, Direct Geometry Access, and Unified Topology Accessing Geometry Directly. The intent of this paper is to provide an overview to the alternative approaches and how they address the specific issues related to accessing CAD geometry for mesh generation. It is not the intent of this paper to provide detailed algorithms related to accessing or repairing CAD data.

Keywords: CAD, geometry, topology, tolerances, design integration, adaptivity, mesh generation, geometry-based, geometry access, Unified Topology Model

1. INTRODUCTION

Automatic and semi-automatic mesh generation has seen dramatic improvements over the last ten (10) years. One of the most important and often overlooked aspects to mesh generation is accessing CAD geometry. The emphasis on analysis in recent years has moved from failure analysis and validation to becoming an active part of the design process. There is a growing demand from manufacturing companies to include performance evaluation based on simulation results earlier in the design process, making simulation an integral part of their design process. To do this in a cost effective manner requires automation of all of the steps involved in performing such simulations from the product design data. Accessing CAD geometry for mesh generation is still one of the major technical issues related to moving simulation forward as an essential ingredient of the design process. This desired ability to move simulation forward in the design process requires a review of current techniques for accessing CAD geometry [1].

This paper will review several of the issues related to CAD geometry access and will evaluate four techniques for CAD geometry access as follows: 1) Translation & Healing, 2) Discrete Representations, 3) Direct Geometry Access, 4) Unified Topology Accessing CAD Geometry Directly.

2. CAD GEOMETRY

CAD systems and their geometric representations have been around for quite some time. Almost all CAD systems have evolved into similar representations for their models. This representation often includes feature based data and a resulting B-Rep instance or a B-Rep model. The B-Rep model consists of much more than just geometry, and indeed one of the major problems in accessing CAD geometry has been due to an oversimplification of what constitutes a valid B-Rep Model.

B-Rep models contain geometry (shape), topology (how things are connected), and tolerances (how closely do they actually fit together). This combination of model data is then accessed by the CAD systems methods to define a valid B-Rep model. Therefore, a valid B-Rep model should be considered to consist of geometry, topology, tolerances and methods used by the CAD system it was defined within [2].

CAD systems often use relatively large tolerances on an entity-by-entity basis to provide robustness to model operations. This approach is referred to as variable tolerances or tolerant modeling by different CAD systems. The use of these large variable tolerances produces gaps and overlaps in the geometry and topology of the CAD system B-Rep model as illustrated in the simple (and extreme) example in Figure 1.
The algorithms used in the CAD system modeling engines are written to deal with these tolerances in a consistent manner and they do not see the gaps or overlaps.

Figure 1. Large / variable tolerances result in gaps and overlaps

Geometric modeling kernels such as ACIS, Granite and Parasolid are often used to supply the methods and model representations used by CAD system modeling engines. CAD systems that use a common geometric modeling kernel also share common methods for evaluating tolerances and the validity of a B-Rep model. These methods can be accounted for directly in the mesh generation process in a consistent manner using information easily provided by the CAD system API [2], [3].

3. GEOMETRY RELATED ISSUES FOR MESH GENERATION

There are several issues associated with effective and efficient access of CAD geometry for mesh generation. This section will provide a quick overview of several of the major issues and the ramifications that this issues have on mesh generation. A detailed review of these issues is beyond the scope of this paper. Specifically excluded from this paper are model abstraction or idealization for analysis and domain decomposition.

3.1 Understanding the Analysis Requirements

The first major issue with CAD geometry access for mesh generation is the need to understand the analysis requirements. The appropriate mesh and geometry to be used for meshing is a function of the analysis to be performed and the desired accuracy [4]. There does not exist an optimal mesh independent of the analysis to be performed. A-priori element shape quality test have often been used as a misleading indicator of a good mesh independent of the analysis to be performed or the accuracy desired. The appropriate mesh is one that produces the desired accuracy for the problem to be solved. In practice this is only achievable through adaptivity.

Different types of analyses require different instances of the geometry to capture the physics. For example, we can perform a dynamic structural response analysis and a Computational Fluid Dynamics (CFD) analysis on the same part. The dynamic structural response analysis requires the solid geometry of the part while the CFD analysis requires the geometry of the cavities through which the fluid will flow. This simple illustration of different use of geometry representations is illustrated in figure 2.

Figure 2. Different analysis require different geometric representations

Physics simulations such as external flow, electromagnetics, and radiation are actually concerned with the volume not occupied by the part.

Different types of analysis also require different resolutions of mesh to achieve the desired accuracy on a particular design.

3.2 Defeaturing

Defeaturing is one of the most complex issues associated with CAD geometry access for mesh generation. Indeed one of the major issues that the CAD and CAE software industries have encountered is developing a consistent definition of a feature. For the purposes of this paper we will classify features into two main groups.
The first group of features will be called “intended features”. *Intended features* are features that were explicitly defined as features in the model that drive the resulting geometry. In this case a feature-based modeling system was used to create a model which contains *intended features*. *Intended features* can only be created by feature-based modeling systems and can be suppressed by the original modeling system.

The second group of features will be called “artifact features”. *Artifact features* are features that are created indirectly by the modeling process. One example of *artifact features* is the creation of engineering features such as holes by a modeling system that is not feature-based. The second example of *artifact features* is the creation of recognizable patterns of geometry / topology data that create a valid design model but also create difficulties associated with mesh generation. Artifact features can be created from any modeling system and cannot be suppressed in the original modeling system.

Part of the complexity associated with CAD geometry access for mesh generation is due to the fact that historically analyses are performed too late in the design process and the design model contains more details than are appropriate for analysis. Moving the analysis earlier in the design process will help to reduce, but will not remove, the need for defeaturing. Since multiple analysis types may be required for any design state there remains a need for defeating to various levels to support the range of analysis to be performed.

One of the most common unwanted *artifact features* encountered in CAD data are “slivers”. *Slivers* can be described as very small *artifact features* that are larger than the geometric tolerances of the CAD System modeling engine but extremely small with respect to the model size. These very small *artifact features* can provide problems to mesh generation algorithms and are meaningless to the analysis [5]. *Slivers* are introduced into models to maintain validity and integrity of the model. Native models contain far fewer *slivers* than translated models. A very common method of healing or repair algorithms used in translation is to introduce *slivers* to resolve gaps, overlaps and tangency conditions.

Modeling engines and healing algorithms may also introduce an large number of faces into the model to ensure that the model is valid. This often occurs in blend and chamfer regions and in areas of similar surface curvature or near tangent conditions. These additional faces may over-constrain mesh generation and one approach is to combine faces into a single larger logical face. This approach has had significant success but typically relies on user input to specify which faces to combine. Extra faces are another type of *artifact feature*.

Another common unwanted *artifact feature* type is “small” model features. Small model features can be described as *artifact features* that are very large with respect to the geometric tolerances but small with respect to the local target mesh size. This definition of *small features* indicates that the classification of a *small feature* is a function of the target element size and accuracy desired. The actual definition of the *small* size can vary with each analysis to be performed. Some typical values for *small features* are less than 25-30% of the target mesh size. This definition also allows for support of an adaptive representation of geometry used for meshing as part of the mesh adaptivity process that we will discuss further later in this section.

*Slivers* may be re-classified as a special case of *small features* that will remain *small* through all possible target mesh sizes. *Small features* are another type of *artifact feature*. The issue of dealing with *small* geometric features in the mesh generation process has been discussed in various references [6], [7]. An example of a *small feature* and its potential impact on mesh generation is illustrated in Figure 3.

![Figure 3. Small feature](image)

CAD models may include geometric features that are important for design but are irrelevant for the simulation to be performed. These unwanted features can be classified as “simple” features and “complex” features. These features can be suppressed by the CAD system if and only if they were *intended features*.

*Simple* features can be described as features which when suppressed or removed refer back to a single parent face on the B-Rep model. *Simple* features may be *intended features* or *artifact features* but are most likely *intended features* from a feature-based modeling system. *Simple* features are defined in terms of the topology of their base features rather than their size. Examples of simple features are illustrated in Figure 4.
3.3 Tolerances and Methods for Evaluating Tolerances

Understanding tolerances and methods for evaluating tolerances plays an important role in accessing CAD geometry for mesh generation. One of the key areas influenced by tolerances and their associated methods is that of tangencies and near tangencies. The methods used in CAD system modeling engines are written to deal with tolerances in a consistent manner. These methods are not available outside of the CAD system modeling engines, therefore, translated data introduces “dirty” geometry.

3.4 “Dirty” Geometry

Dirty geometry has been one of the most nagging issues related to geometry access. Dirty geometry consists of gaps, overlaps and other incompatibilities in the model preventing the model from being valid. These incompatibilities do not exist in the native CAD system and are introduced from translating the native CAD geometry to another format. Differences in representations, methods and tolerances between modeling engines create dirty geometry. Translators must then heal or repair the geometry to represent it as a valid model in the non-native system [5], [8], [9]. Note that without knowledge of the modeling system tolerances and methods, there is no a-priori means to ensure a healing process will successfully recover the correct model representation.

3.5 Support for Curved Meshing

The previous issues associated with geometry access have focused on ensuring the correct geometry representation and level of detail in the geometry be used for mesh generation. The next three issues deal with ensuring that the geometry access can support key mesh generation functionality. The first mesh generation functionality to be considered is curved meshing. Curved meshing involves the ability to create curved mesh edges and faces that have the level of geometric approximation needed to ensure that as the simulation results are improved by the introduction of higher-order equation approximations (e.g., high-order finite elements), the geometric approximation errors do not control the solution accuracy. The ability to properly curve the mesh entities occurs as soon as higher than linear basis functions are used and, as demonstrated by simple example in [10], the order of geometric approximation needed to be increased as the basis order increases.

In the simplest cases, the appropriate curved meshes can be created by moving mesh on the boundary of the model to the “closest” location on the model geometry. However, even in the simplest, and common, case of quadratic h-type finite elements (see example at the top of Figure 5), a more complex algorithm is required to ensure the elements can be properly curved [11]. The complexity of the curved mesh generation process increases further in the case of p-version methods where coarse meshes, such as the example at the bottom of Figure 5, must have higher order geometric approximations.

Figure 4. Simple features on top face

Complex features can be described as features that are not simple. Complex features may be intended features or artifact features but are most likely intended features from a feature-based modeling system. These features include a variety of features as follows:

- Features whose base feature spans across multiple faces.
- Features whose base features need to be extended for feature removal or suppression such as fillets and chamfers.
- Features that interfere with other features.

Complex features are the largest challenge to deal with in defeaturing. If these features are not small with respect to target mesh size, careful consideration should be given regarding why these are being defeatured and the impact on accuracy. If these features are small then they can be treated as small features independent of their complexity. For complex features that need to be removed or suppressed that are not small a thorough understanding of the feature data is required and it usually best to suppress these in the CAD system prior to geometry access.
3.6 Support for Curvature Based Mesh Refinement

The next meshing functionality to be considered as desirable to be supported is curvature based mesh refinement. This meshing functionality provides automatic refinement of the mesh based on the underlying geometry curvature. The benefits of this functionality are: 1) the ability to capture the geometry with a considerably smaller number of elements and/or grid points and 2) resulting improvement in mesh quality in areas of rapid geometric changes. Figure 6 illustrates the benefits of curvature based mesh refinement.

Figure 6. Curvature Based Mesh Refinement

3.7 Support for Geometry Based Mesh Adaptivity

The final mesh generation functionality to be considered, in this paper, as an issue for geometry access is the support for geometry based mesh adaptivity. This functionality involves the ability of the adapted mesh to adhere to the original geometry as illustrated in figure 7 and requires access to the original geometry to be present. Mesh adaptivity that does not adhere to the geometry is limited by the initial mesh geometric approximations and can provide results that are meaningless. For example, Figure 7 is a close-up of a geometric feature in an accelerator cavity geometry where the simulation procedures must provide highly accurate estimates of the electrical and magnetic losses. The sensitivity of the results to the local geometric shape is so high that if the mesh geometric approximation did not improve as the adaptive simulation process continued, the results obtained would have been not just a poor approximation, but meaningless.

In many problems of interest the mesh edges and faces are of the same size as the small geometric features that are often critical to the analysis, such as the accelerator cavity. In these cases, the simple movement of new nodes introduced during refinement to the curved model surfaces can yield invalid elements. The algorithms needed to effectively deal with these situations include must include general mesh modification operations and a control algorithm that ensures the procedure is progressing in a positive manner [12].

Figure 5. Curved Meshing

Figure 7. Geometry based mesh adaptivity

The advantages of geometry based mesh adaptivity include: 1) the ability to start with coarser initial meshes and, 2) the ability to ensure that the resulting model adheres at an appropriate level of accuracy to the design geometry. An additional benefit that may not be apparent is the
combination of geometry based mesh refinement with the small feature defeaturing as a function of target mesh size. This can result in adaptive geometry representation for mesh adaptivity where small features are ignored in the initial mesh and accounted for as a function of target mesh size in each stage of the mesh adaptivity process. This combined approach dramatically reduces the defeaturing requirements associated with geometry access for mesh generation and allows for initial coarse meshes of detailed geometric models. Figure 8 illustrates an example of this combined approach to adaptive geometry representation.

![Initial coarse meshes approximates small features](image1)

![Adaptive mesh accurately accounts for small features](image2)

**Figure 8. Adaptive geometry representation**

### 3.8 Evolving Geometry Problems

There are a number of situations where the model shape and topology can evolve during the simulation. When the simulation is performed using Lagrangian type analysis and there are large deformations and/or model fracturing, it is often necessary to update the domain and mesh several times during the simulation (e.g., in fragmentation simulations [13] or metal forming [14]). In these situations the model topology and shape must be updated based on the simulation results. Even in the case where the original geometric model was defined in a CAD system, it is most likely not desirable to continue to use the original CAD system to update the CAD model. This is because the new geometric information available from the simulation is limited to node point coordinates on the mesh facets and most CAD systems do not effectively support such geometry updates.

An important aspect of properly updating the geometric model for these cases is to update the model topology based on the simulation information and to associate the appropriate collections of mesh edges and mesh faces with the resulting model edges and faces to use in the subsequent definition of shape information. Algorithms to do this based on mesh based geometry parameters and/or simulation contact or fracture information have been developed [15], [16], [14]. Once the model topology has been defined, the geometric shape information can be defined directly in terms of the mesh facets, or can be made higher order using subdivision surfaces [17], [16] or higher order triangular patches [18] [19]. Reference [14] provides a description of an automated adaptive medal forming procedure where the updated geometric model is defined based on the simulation information and higher order updated shapes of the edges and surfaces are defined by subdivision patches applied on a model entity level.

### 3.9 Integration of Simulation in the Design Process

Integration of simulation in the design process is a driving factor for improved geometry access for mesh generation and support of this integration should be considered as a major issue when considering geometry access. This integration allows for simulation to be an integral part of the design process and requires use of the native CAD system geometry as the geometry source to allow for effective reuse through multiple design iterations. Mesh generation needs to access the current design state and evolve with the design [20]. Automatic meshing and geometry based mesh refinement are fundamental requirements to ensure efficiency and accuracy. Integration of simulation in the design process also requires sophisticated management of simulation attributes to support design change insensitivity for simulation.

### 3.10 Multiple CAD Geometry Sources

The geometry access issues discussed so far in this paper are limited to a single CAD system. These issues are further complicated by the need to support multiple CAD systems. Each CAD system modeling engines uses different representations for geometry and topology and different tolerances and methods for evaluating tolerances. Direct interface utilities to multiple CAD systems is both complex and expensive to develop and support. Modeling kernels such as ACIS, Granite and Parasolid help to reduce the scope of this problem.

Commercial software vendors need to provide support for multiple CAD systems to properly support their customer base. It should also be noted that large-scale design environments and processes typically consist of multiple CAD systems both internally within a company and throughout the supply chain.
4. TECHNIQUES FOR ACCESSING CAD GEOMETRY FOR MESH GENERATION

There are several techniques currently used and being developed to address the geometry access issues outlined in this paper. The techniques used for geometry access can be classified into four major approaches as follows:

• Translation & Healing
• Discrete Representations
• Direct Geometry Access
• Unified Topology Accessing Geometry Directly

4.1 Translation and Healing

Translation and Healing has historically been the most commonly used technique for geometry access. The translation may involve use of standard file formats or direct translators.

IGES does not address issues with representations, global tolerances, features, tolerancing or tolerance methods and typically results in dirty geometry [5]. Standards such as VDAFS and STEP do address issues with representations and global tolerances but do not address features, tolerancing or tolerance methods and often results in dirty geometry (typically cleaner than IGES).

Many companies have invested millions to resolve the translation related issues (ITI, Elyium, Spatial, TransMagic, CAD-CAMe, TTI, TTF, …) An entire interoperability industry has evolved to attempt to address the issues of Translation and Healing. Progress has been made but the Translation and Healing process is still not reliable or robust. The fundamental issue of differing native tolerance methods has not been addressed.

Evaluation of Translation and Healing as related to geometry access issues presented in this paper is as follows:

• Defeaturing is difficult since intended feature information is lost in translation and unwanted artifact features may be created.
  • Feature-based translators attempt to reproduce models from feature representations but do not address tolerance methods and may fail to rebuild models or introduce slivers and small features.
  • Healing typically introduces slivers and small features to resolve dirty geometry.
  • Non feature-based translators require explicit feature removal.
  • Feature suppression with non feature-based translators requires feature recognition algorithms.
  • Translation & Healing introduces dirty geometry due to differences in CAD systems modeling engines representations, tolerances and methods.

• The resulting geometry representation typically can support curved meshing, curvature based refinement and geometry based mesh adaptivity on modified representation.
  • It is possible to support adaptive geometry representation on modified representation with small feature recognition.
  • The ability to support evolving geometry is limited by the geometry representation available.
  • The integration of simulation in the design process is not effectively addressed.
  • Differences in algorithms and tolerances between modeling engines make it impossible to exactly exchange data between them. Results and robustness vary dramatically with different CAD systems.

4.2 Discrete Representations

The Discrete Representations technique is based on the generation of a faceted model by the CAD system and accessing the resulting faceted model for mesh generation. This is most commonly done based on simple facets generated by the CAD system faceter but may also use subdivision surfaces [17], [16] or higher order triangular patches [18].

This technique is often used to attempt to eliminate dirty geometry and to resolve differences between different CAD systems. There are some remaining concerns regarding robustness since the simple facet representations are designed for visualization and may not close as illustrated in figure 9. These facet representations are often done on a face-by-face basis and may not be incompatible across face boundaries.

The successful use of the simple facets in Discrete Representations technique is highly dependent on the faceter used by the originating CAD system. All Discrete Representation techniques result in an approximation of the geometry and do not retain the intended feature data, and geometry of the CAD model.

Figure 9. Facet representations from major CAD system modeling engines may not close
Evaluation of Discrete Representations as related to geometry access issues presented in this paper is as follows:

- Defeaturing of any type is difficult since all intended feature information is lost.
- Simple facet representations are designed for visualization and may still have some problems with dirty geometry.
- Simple facet representations cannot support curved meshing, curvature based refinement and geometry based mesh adaptivity.
  - More sophisticated discrete representations such as subdivision surfaces and higher order triangular patches can support an approximate version curved meshing, curvature based refinement and geometry based mesh adaptivity.
  - It is difficult to support adaptive geometry representation on modified representation with small feature recognition.
- The definition of evolving geometry can be supported.
- The integration of simulation in the design process is not effectively addressed.
- Handles data from different systems in a consistent manner but results may vary dramatically due to differences in CAD System faceters.

**4.3 Direct Geometry Access**

Direct Geometry Access is a technique that is growing in popularity based on accessing CAD geometry directly through CAD system toolkits such as CATIA CAA and Pro/Toolkit [21]. Use of the CAD system toolkits requires that a seat of the CAD system is available for geometry access.

Since many CAD systems use geometric modeling kernels this approach can also achieved by licensing the same geometric modeling kernel as the CAD system and accessing the geometry through the modeling kernel APIs [1], [3], [22], [23].

The main theme of this approach is to leave the data in the native modeling engine and to use that native modeling engine to access geometry so that the native tolerances and methods are used for geometry access and wherever possible the intended feature data is retained.

Evaluation of Direct Geometry Access as related to geometry access issues presented in this paper is as follows:

- Defeaturing is an issue for artifact features that cannot be suppressed.
  - Small features, slivers and multiple faces cannot be suppressed.
- Native geometry is not dirty.
- Can support curved meshing, curvature based refinement and geometry based mesh adaptivity.
- Adaptive geometry representation with small feature recognition is extremely difficult (if not impossible) to support.
- The ability to support evolving geometry is limited by the geometry representation available.
- The integration of simulation in the design process can be effectively addressed with unique solutions for each CAD modeling source.
- Requires multiple direct interfaces for a broad range of geometry support.
  - Each CAD system has a different geometry and topology representation to interrogate for meshing.
  - Each CAD system has different tolerances and methods to understand.
  - Each CAD system has a different toolkit for accessing geometry and topology data.

**4.4 Unified Topology Accessing Geometry Directly**

The final geometry access technique to be considered is Unified Topology Accessing Geometry. This is a natural extension of the Direct Geometry Access technique with enhancements to overcome the shortfalls that are associated with multiple CAD sources and defeaturing of artifact features. This approach is based on an abstraction of the geometry that allows multiple sources of geometry to be treated the same by the mesh generator [1], [3], [22], [23]. For the purposes of this paper this abstraction of the geometry will be referred to as the Unified Topology Model.

The Unified Topology Model is a representation of the model for simulation purposes that retains it connection to the original CAD system geometry and topology. This approach provides a separate topology data structure that allows for multiple forms of defeaturing while retaining the original geometry & topology. This approach also facilitates the use of geometry from multiple sources.

The geometry is directly accessed from the native modeling system as per the Direct Geometry Access technique, however, a common description of the topology is created that is well suited for mesh generation. This Unified Topology Model accounts for the topology of the original modeling systems and enhances this representation to make it more suitable for analysis. These enhancements may include; support for multi-dimensional models, non-manifold model (extremely useful for assemblies), defeaturing of unwanted features, and support for models from multiple CAD sources for a single analysis.

One important aspect of the Unified Topology Model is to maintain a relationship between the Unified Topology Model and the topology of the original CAD model. This may be a one to one relationship, or a one to many
relationship. Maintaining these relationships allows the Unified Topology Model to be modified for analysis without affecting the underlying CAD model while still maintaining the Direct Geometry Access for all geometric queries.

One example of a Unified Topology Model is the Simulation Modeling Suite provided by Simmetrix, another example is the CGM provided by Sandia National Laboratories. In the Simmetrix example the Unified Topology Model builds on top of the CAD topology to present a standard representation for all modeling sources (non-manifold topology similar to Radial Edge Data Structure [24]). The Unified Topology Model is built from the CAD topology and geometric queries are passed through to the CAD system via direct access to APIs or modeling kernels. The implementations by Simmetrix and Sandia also support discrete geometry as a modeling source. The resulting Unified Topology Model used is illustrated in Figure 10.

![Unified Topology Model](image)

**Figure 10. Unified Topology Model**

Evaluation of Unified Topology Accessing Geometry as related to geometry access issues presented in this paper is as follows:

- Allows for various forms of defeaturing.
  - *Slivers and small* features can be addressed as a function of global and local target mesh sizes. Figure 11 illustrates the effect on meshing results related to defeaturing of the *small* features illustrated in Figure 3.

![Small feature removed from Unified Topology Model](image)

**Figure 11. Small feature removed from Unified Topology Model**

- *Simple* features can be suppressed in Unified Topology Model for meshing purposes.
- *Complex* features may be addressed either by suppression of *intended* features in the CAD system or as *small* features in the Unified Topology Model.

- Uses native system tolerances and methods.
  - Native geometry is not *dirty*.
- Curved meshing, curvature based refinement and geometry based mesh adaptivity can be supported.
- Can support adaptive geometry representation with *small* feature recognition.
- Creation of new topology in the Unified Topology Model based on a discrete geometry basis provides support for evolving geometry problems.
- Proven effective to address issues related to integration of simulation in the design process.
  - Used in large Simulation-Based Design initiatives and commercial CAE Software (Visteon, John Deere, Blue Ridge Numerics, CFD Research Corporation, ESRD, Coventor, PVM Corporation and many others)
- Provides a single interface for a broad range of geometry support.
  - Geometry abstraction layer handles all CAD systems specific issues.
- Mesh generation algorithms access a consistent Unified Topology Model.

5. SUMMARY

The desire to use simulation as an integral part of the design process has necessitated an evaluation of the issues and techniques associated with CAD geometry access for mesh generation. A broad range of issues was highlighted in this paper and four techniques for CAD geometry access were reviewed with respect to these issues.

Translation and Healing was the initial technique reviewed and was found to lack the reliability and robustness necessary to support design/analysis integration. The Translation and Healing technique does not address several of the geometry access issues outlined.

The second technique reviewed was Discrete Geometry Representations. This technique does address some of the geometry access robustness issues but does not address well those issues related to feature representations, curvature based meshing and design integration.

The third technique reviewed was Direct Geometry Access. This technique does address many of the geometry access issues but does not address well those issues related to defeaturing of *artifact* features and multiple CAD systems.

The final technique reviewed was Unified Topology Accessing Geometry Directly. This technique provides an effective means to address to the geometry access issues outlined in this paper. The Unified Topology Model Accessing Geometry Directly technique is the most flexible technique for addressing issues related to accessing CAD geometry for mesh generation.
Unified Topology Accessing Geometry Directly can provide a single environment to effectively deal with integration to CAD from multiple sources, along with integration with various discrete models, and defeating of artifact features providing a firm foundation for design/analysis integration.

REFERENCES


PARALLEL GENERATION OF UNSTRUCTURED SURFACE GRIDS

Udo Tremel\textsuperscript{1} Frank Deister\textsuperscript{1} Oubay Hassan\textsuperscript{2} Nigel P. Weatherill\textsuperscript{2}

\textsuperscript{1}EADS, Munich, Germany {udo.tremel|frank.deister}@m.eads.net
\textsuperscript{2}University of Wales Swansea, Swansea, U.K. {o.hassan|n.p.weatherill}@swansea.ac.uk

ABSTRACT

In this paper a new grid generation system is presented for the parallel generation of unstructured triangular surface grids. The object-oriented design and implementation of the system, the internal components and the parallel meshing process itself are described. Initially in a rasterisation stage, the geometry to be meshed is analysed and a smooth distribution of local element sizes in 3-D space is set up automatically and stored in a Cartesian mesh. This background mesh is used by the advancing front surface meshing as spacing definition for the triangle generation. Both the rasterisation and the meshing are MPI-parallelised. The underlying principles and strategies will be outlined together with the advantages and limitations of the approach. The paper will be concluded with examples demonstrating the capabilities of the presented approach.

Keywords: unstructured surface mesh generation, geometry rasterisation, MPI-parallel, automatic, object-orientation

1. INTRODUCTION

The generation of unstructured surface grids is still one of the most interactive tasks to be performed during a numerical CFD analysis requiring a considerable amount of human effort. For the complex configurations currently in a routine use, the time to get a well suited surface grid tends to become a bottleneck\textsuperscript{[1]} compared to the later stages such as volume mesh generation and numerical simulation. Two main reasons can be identified causing this.

One problem is the preparation of the geometry definition itself. Time constraints prohibit the detailed examination and clean up of the full geometry. Due to undetected and unrepairable features, failures generally occur during the meshing process. These errors stipulate human repair and modification steps until the geometry definition can be meshed successfully. To accelerate this process, the response time of the surface grid generator should be as short as possible.

The second reason is the, mostly interactive, definition of the local element sizes to be applied for different parts of the geometry. For example, for a well suited aerodynamic surface grid the leading and trailing edges have to be resolved accurately from the beginning, compared to other regions such as fuselage surfaces, etc. Even in the case of mesh adaptation cycles during the simulation, the initial surface grid should be able to resolve the major flow characteristics, so that only a few adaptation steps are applied in order to obtain an acceptable result. This implies that reasonable element sizes have to be defined for the (initial) surface grid. Standard means are tetrahedral background grids and sources\textsuperscript{[2]} together with octree-based approaches \textsuperscript{[3, 4]}. For complex configurations such as complete aircrafts, hundreds of sources are not uncommon, imposing on the user hours of tedious work.

In the following sections a recently developed surface mesh generation system is presented. Its objective is to reduce the above mentioned problems as far as possible. The object-oriented design and implementation
is described first, followed by a section focusing on the automatic definition of the local element sizes in 3-D space. In section 4 the parallelisation of the time-consuming tasks is illustrated and in section 5 examples demonstrate the capabilities of the approach. The paper is concluded by a summary.

2. OBJECT-ORIENTED DESIGN AND IMPLEMENTATION

Based on the existing FLITE-ST surface mesh generator[5] from the School of Engineering of the University of Wales Swansea, a new object-oriented (OO) surface mesh generator system, the ST++-system, has been developed at EADS M. The OO-methodology has been chosen because the OO concepts of polymorphism, inheritance and encapsulation[6][7] inherently enable and support the building and maintenance of large and complex software systems. Compared to the procedural structured programming in, for example, Fortran 77 or C, the resulting code is generally better understandable, maintainable, extensible and reusable. This is caused by the powerful features supported in OO languages such as strongly typed interfaces, templates, design patterns, etc.[8][9] C++ has been selected as implementation language due to the rich OO features the language offers[1]. Another reason is the downward compatibility to C which enables the integration and reuse of already available, validated and highly optimised C and Fortran 77/90 routines.

2.1 ST++ System Overview

In Figure 1 the three major components of the ST++-system are shown. These objects interact together and perform the following steps to generate a surface mesh:

1. The geometry definition imports the geometrical and topological items from CAD data.
2. The mesh size specification initialises the prescribed means to define the spacings.
3. The surface mesher starts the advancing front triangulation of the geometry, controls the mesh enhancements and prepares and exports the surface mesh later to be used in the volume mesh generation.

2.2 Geometry Definition

All the geometrical and topological entities are encapsulated in the geometry definition object illustrated in Figure 2. The implemented boundary representation (B-Rep) structure consists of geometrical points, curves and surfaces referenced by topological vertices, edges and faces respectively, defining the outer hull of the body to be meshed. All types of items can be grouped together and the grouping information can be kept in multiple tables to allow non-unique identifiers[2]. Specific attributes can be assigned to each group, which enables, for example, the use of different tolerances for the configuration and the outer fairfield parts. Import filters are available for the industrial relevant data exchange formats STEP[10], NASA-IGES[11] and the traditional FLITE format[5].

All the geometrical entities are derived from a generic point, curve or surface object defining a common interface for all derived types. For example, the following methods are part of the interface for curves:

![Figure 1: Surface mesh generator design.](image1)

1Strong type checks, single and multiple inheritance, templates, abstract classes and interfaces, streams, exceptions, the standard template library (STL), …

![Figure 2: Geometry definition component.](image2)

2Depending on the CAD database the geometry was imported from.
• Evaluate(doubleT u, doubleT *outXYZ, doubleT *outTangent=0)
• CalculateProjection(const doubleT *xyz, doubleT &outU)
• CalculateArcLength(doubleT u1, doubleT u2, doubleT &outL)

Algorithms working only with such an abstract interface are independent from the real mathematical representation of the underlying parametric spline composite curves and tensor-product surfaces. Implemented are various types of curves and surfaces such as Ferguson-splines[12], Bezier-splines[13] or NURBS[14].

For dynamic modifi cations the geometry defi nition object offers methods for the insertion and deletion of geometric entities, for the transformation of existing entities, etc. One important capability is the extraction of arbitrary subgeometries, which is heavily used by the parallel meshер to extract subparts transmitted to and meshed by another process.

2.3 Mesh Size Specifi cation

Another important aspect of mesh generation, besides the handling of geometrical CAD data, is the control over the spatial distribution of size and shape of the elements to be generated. Inside the ST++, this is the task of the mesh size specifi cation object shown in Figure 3. The local element size at a certain point in 3-D space is determined by the minimum size defi ned by all active objects. For that purpose, a Cartesian background grid, a tetrahedral background grid and sources are available, which can be used independently from each other depending on the requirements of the user. Additional types of application specifi c sources can easily be added, because all sources are derived from a generic source object defi ning the following common lean interface for all sources:

• GetMinSpacing(doubleT &outMinSpacing)
• GetSpacing(const doubleT *xyz, doubleT &outSpacing)

These methods only have to be specialised and implemented to add a new source type transparently to the objects working with the mesh size specifi cation object.

Several methods are offered to achieve dynamic modifi cations of the local edge lengths. Sources can be added, deleted or modifi ed dynamically, all spacings can be scaled in common and spatial transformations can be applied to all entities. These operations are intended to be used during dynamic mesh modifi cation processes originating from transient simulations, design optimisation, etc.

The automatic determination of a well suited Cartesian background mesh based on a rasterisation of the CAD geometry is presented in section 3. This approach differs from the octree based approaches presented in [3, 4], because the sizes of the Cartesian cells are largely independent from the local element sizes calculated. Only the scalar quantities defi ned in each Cartesian cell are used to derive the mesh size at a certain point in space. Experience has shown that the Cartesian cell size is generally two to eight times larger than the calculated lengths, hence, the Cartesian meshes created are much smaller than those grids generated in [3, 4].

2.4 Surface Mesher

The kernel of the ST++ system is the surface mesher component illustrated in Figure 4. Its task is the control and the exchange of data between the different objects responsible for the different steps of the
meshing procedure. Based on the advancing front algorithm\cite{15} \cite{16} the different steps are spread across the curve discretiser, the initial front generator and the advancing front triangulator object. For each topological face to be meshed, the curve discretiser discretises all the topological edges connected to the face if not already done. This set of straight sides is given to the initial front generator which builds up the initial advancing front. The triangulator uses this starting front to generate the triangles added to the surface mesh. When no sides are left in the front, the triangulation is finished and the surface mesh is transferred to the mesh enhancer for optimisation. After optional postprocessing steps, such as removal of duplicate nodes and edges, correcting the orientation of the facets, checks, etc., the merged surface meshes can act as the starting point for a volume triangulation.

3. GEOMETRY RASTERISATION

A fully automatic and parallel feature-based rasterisation of native CAD data has been developed. The local curvature and characteristic length are investigated along CAD curves and inside trimmed CAD surfaces in order to define local sample lengths. A locally refined Cartesian background mesh (octree data-structure) is constructed to prolongate and therewith smooth the sample lengths. Additionally, Cartesian cells inside the geometry may be blanked out in order to avoid length prolongation through solid bodies. During the surface mesh generation, the Cartesian background mesh serves as the mesh size specification. Details are provided in \cite{17}.

3.1 Rasterisation of Native CAD Curves

The rasterisation of CAD curves are controlled mainly by three sampling parameters specified by the user: minimal arc length $L_{\text{min}}$, maximal arc length $L_{\text{max}}$ and maximal curvature angle $\alpha_{\text{max}}$. Now a CAD curve is subdivided into consecutive curve segments applying the following three sampling criteria: the arc length of a curve segment must not be smaller than the specified minimal arc length $L_{\text{min}}$. Conversely, the curve segment length must be smaller than the maximal arc length $L_{\text{max}}$. Finally, the curvature angle must be smaller than the maximal curvature angle $\alpha_{\text{max}}$. This last sampling criterion is only applied, if the arc length is larger than the minimal arc length $L_{\text{min}}$. The curvature angle is taken as the angle between the tangential vectors at the two end points of a curve segment. The sample length cannot be larger than the length of the corresponding CAD curve. Finally, all curve segments are approximated by straight lines. For each straight line a bounding box is determined, termed raster box, which controls the local resolution of the later to be generated Cartesian mesh.

3.2 Rasterisation of Native CAD Surfaces

The rasterisation of CAD surfaces requires the same three user specified sampling parameters $L_{\text{min}}$, $L_{\text{max}}$ and $\alpha_{\text{max}}$, which are used also for curve rasterisation. Because only the part inside a trimmed CAD surface is considered, a scan-line algorithm\cite{18} \cite{19} from computer graphics is applied. As a first step, the trimming CAD curves are approximated by sequences of straight lines. For this, they are rasterised as described earlier. The resulting straight lines in physical (Cartesian) space are transformed to the $(u,v)$-parameter space of the CAD surface, in which the remaining computation takes place. This discrete representation of corresponding trimming curves must not intersect each other because of the scan-line algorithm.

The second step consists of computing the stencil point distribution, where the local surface curvature will be investigated later. For both $u$- and $v$-direction equally distributed iso-curves (probes) of the CAD surface are rasterised applying again the previous curve rasterisation algorithm. However, this time the curvature angle is defined as the angle between the surface normal vectors at the end points of a curve segment. The end points of the evaluated curve segments are taken as the desired stencil points. For each direction, the final stencil point distribution is extracted from these probes. In Figure 5 the final stencil point distribution in $u$- and $v$-direction are represented by the circles.

Now the scan-line algorithm is applied separately for the $u$- and $v$-iso-curves, which are defined by the stencil points (third step). It identifies the parts of the iso-curves which are inside the polygon constituted by the sequences of straight lines and thus inside the trimmed surface. The demarcations are drawn as squares in Figure 5. However, these inner curve parts are rasterised again with the presented approach. Afterwards, the computed sample lengths are related to the stencil points located inside the trimmed surface (circles drawn with thick lines in Figure 5). Every stencil

![Figure 5: Rasterisation of trimmed surface.](image-url)
point inside the trimmed surface gets a raster box according to the stored sample length. Additional raster boxes are created if the distance between two stencil points is larger than their sample lengths. In this way, the trimmed surface is completely enclosed by raster boxes, which are used for the generation of the Cartesian background mesh.

3.3 Cartesian Background Mesh

The locally refined Cartesian background mesh specifies the mesh size required by the surface mesh generator. It is based on the hierarchical octree-data structure describing the connectivity between the Cartesian cells[20][21][22]. The raster boxes along the CAD curves and CAD surfaces determine the local resolution of the Cartesian background mesh: all Cartesian cells which intersect a raster box are identified. These Cartesian cells must not be larger than the current raster box. Besides, the sample length of the corresponding curve segment, or rather, the surface stencil point is stored in every intersected Cartesian cell. At the end, the Cartesian background mesh is smoothed accordingly to a one-level difference rule: it is not allowed that two neighbouring Cartesian cells differ by more than one refinement level.

The stored sample lengths are prolonged through the Cartesian mesh. The rate of change between adjacent Cartesian cells is limited by an user-defined slope. In this way, a smoothed sample length distribution is achieved in the complete flow domain. Moreover, the user is able to control the rate of coarsening of the triangulation by modifying this slope parameter. Finally, the gradient of the sample length is calculated using a least square method.

During generation of the surface triangle mesh, the Cartesian background mesh specifies the local mesh size. For each point, the local mesh size is required. First, the Cartesian cell is identified applying the hierarchical octree data-structure enclosing this point. Then the sought mesh size is interpolated linearly using the sample length and its gradient, which is stored in the Cartesian cell. The smoothed mesh size specification is also available in space and therefore usable by a volume mesh generator.

3.4 Blanking out of Solids

The penetration of the sample length through solids is avoided. This means, that the local mesh size of the lower side of a thin geometry does not influence the mesh size on the upper side. Therewith, unneeded refinement is avoided and the resulting surface mesh is locally more homogenous. Especially this affects the quality of a possible quasi-prismatic mesh, because the local prismatic mesh height strongly depends on the corresponding (underlying) surface triangle. Figures 6 and 7 show the impact of blanking out solids for the nacelle of a generic transport aircraft: the smaller sample lengths of the engine (Figures 15 and 16) penetrate through the solid nacelle coating and reduce the sample lengths there. Figure 7. In contrast, Figure 6 illustrates that this does not occur, if the prolongation through solids is avoided. For thick geometries, blanking out of solids is not necessary.

![Figure 6: Without blanking out solids for nacelle (generic transport aircraft).](image)

![Figure 7: With blanking out solids for nacelle (generic transport aircraft).](image)
closed initial surface mesh of the geometry is required. If an initial surface mesh is not available (for example STL output format of the CAD system), the initial surface mesh is generated without blanking out solids. Here - after rasterisation of the geometry and before length prolongation - the smoothed Cartesian mesh together with the initially stored sample lengths are stored. Therewith, it is avoided to raster the geometry again for generating the final surface mesh.

After finishing the initial surface mesh, the sample length is prolonged again using the Cartesian mesh previously stored. But this time, all Cartesian cells inside the body are not considered for the length prolongation. In order to blank out inner cells, all Cartesian cells are identified, which are intersected by the initial surface mesh. The locations (inside / outside) of the remaining cells are found using a ray-tracing and a coloring algorithm[19] [22]: first the location of a cell (with undefined location) is determined by ray tracing. Afterwards, all neighbor cells obtain the same location recursively, which are not marked to be intersected by the geometry. Figure 8 presents all Cartesian cells which are inside the generic transport aircraft, whereas the intersected cells are drawn in Figure 9. Here, the small pictures show the regions marked by the black circles.

**Figure 8:** Cartesian cells inside the geometry (generic transport aircraft).

**Figure 9:** Cartesian cells intersected by the geometry (generic transport aircraft).

used successfully for parallel surface meshing as will be shown in section 5.

4.1 Parallel Geometry Rasterisation

The first computational intensive part is the rasterisation of the geometry. Here the main loop over all edges and faces is parallelised in a pipeline approach, which is illustrated in Figure 10. At the beginning the n par-

**Figure 10:** Parallel rasterisation of the geometry.
sent to the collector process for an adaptation of the Cartesian mesh. At the end the collector process performs all further postprocessing operations (smoothing, length prolongation, I/O, etc.) on the Cartesian mesh in sequential mode. Hence, the scalability of the parallelisation is limited, but up to a modest number of processes a sufficient speedup of more than one order of magnitude is obtained as demonstrated in section 5.

4.2 Parallel Surface Meshing

For the parallel surface meshing the loop over the faces to be meshed is parallelised similar to the parallel rasterisation. This is shown in Figure 11. After each process has determined its role (distributor, merger or worker), the necessary initialisation steps are executed. The distributor reads in the complete geometry and all workers are initialising the mesh size specification. Then the parallel meshing starts. Each worker queries for the next set of faces to be meshed to enable an automatic load balancing. Depending on whether or not the distributor pre-discretises the face boundary edges, the next set of geometric entities is sent to the corresponding worker with or without the (already) pre-discretised edges. This pre-discretisation is mandatory due to floating point roundoff errors, which may cause an edge to be discretised with one side more or less, which prevents the recombination of the submeshes into one consistent mesh due to non-matching boundaries. Heterogeneous clusters running different types of processors are candidates for this parallel pitfall. When the worker has finished the advancing front triangulation and the mesh enhancements, the surface mesh is transmitted to the merger process. After all faces have been meshed, the merger assembles the complete mesh by combining all the received submeshes. On the final grid further postprocessing operations (unifying the orientation, etc.) are performed in sequential mode.

As with any pipelining strategy, it is important to:

- keep the pipeline filled,
- select the optimal length of the pipeline,
- optimise the chunks of data to be processed between different stages.

Hence, a CAD model of reasonable size should be used if also many processors are involved, otherwise a reasonable scalability will not be obtained. Additionally, faces requiring more computational effort should be processed first during the parallel meshing to balance the throughput. Especially at the end, such faces should not occur, otherwise some processors continue working while most of the others have already finished their work, which can heavily limiting the scalability of the approach. Therefore a heuristic meshing weight is calculated for each face during the rasterisation. This weight is estimated by the sum over all inverse edge lengths times the surface areas of the corresponding raster boxes. Such a weight is approximately proportional to the number of triangles to be generated for the face and enables the above mentioned optimisations concerning the order in which the faces are meshed.

The size of the subgeometries can be adapted accordingly to the capabilities of the underlying communication network. Smaller subsets can be used for low-latency/high-bandwidth networks, whereas larger subgeometries might be more favourable for less performant interconnects reducing the number of messages to be exchanged. However, this also depends to a large extent on other factors such as the size of the geometry, the processors, etc.

It is clear that the maximum speedup achievable is inverse proportional to the maximum time needed to raster/mesh a single face. If some faces consume a large amount of computational time, the parallel execution will not perform as expected. In such cases a fine-grain parallelisation of the underlying algorithms will have to be used to further speed up the process. Nevertheless, if the meshing time can reduced more than an order of magnitude as will be shown in section 5, the result is worth the effort inherent with the presented approach.

4.3 Parallel Surface Remeshing

The parallel surface remeshing presented in Figure 12 is characterised by a three stage approach:

1. Parallel surface mesh analysis
2. Parallel curve rediscrtisation

3. Parallel surface remeshing

It starts with the initialisation of the modified mesh size specification in every process. Modifications can be caused by the addition/removal of sources, by a modified raster length prolongation, etc. For a parallel analysis the existent surface mesh is partitioned across the available processes with one element overlap across the domains by means of the MeshLib-library\(^3\). In the first stage all elements (triangles) containing edges not respecting the mesh size are marked within each partition. To ensure that both the sequential and the parallel marking algorithms result in the same set of markers, the states of the external elements\(^4\) have to be exchanged after each marking loop in the parallel version. After this parallel analysis step, the partitions including the markers are collected again on the single distributor process, which extracts the parts to be remeshed and forwards the unselected parts directly to the merger process. In the second stage, the parallel remeshing of all internal edges lying on curves of the geometry is performed. Start- and endpoints of such segments are determined by the distributor and sent, together with the underlying geometric entity, to the next worker waiting for data. The remeshed segments are collected and recombined by the merger and returned back to the distributor when all remeshings have been performed. In the third stage the distributor extracts the outer boundary of each hole and sends away these edges, together with the underlying surface definitions, to the next worker querying for data. The remeshed holes are collected at the merger and inserted again into the existent mesh.

It depends on the application area whether a complete regeneration from scratch or a local remeshing based on the already existent surface mesh is the better approach. Concerning dynamic CFD simulations the remeshing will be the alternative of choice because most of the volume mesh is normally kept fixed and only a small subset of the mesh has to be modified. With a complete new surface mesh also the entire volume mesh would have to be regenerated causing much more overhead compared to the local remeshing. During the generation of the initial surface mesh a complete regeneration might be the best option due to the normally similar time required and the slightly better quality.

5. EXAMPLES

5.1 Generic Transport Aircraft

The presented concept of fully automatic surface mesh generation is demonstrated with a generic transport aircraft. The CAD description is imported via the STEP format and consists of 864 NURBS curves and 346 NURBS surfaces. The only input parameters (for CAD surfaces) specified by the user are the minimal arc length \(L_{\text{min}}\) (default value 0.25 [mm], for wing 0.1 [mm], for nacelle 0.3 [mm]), the maximal arc length \(L_{\text{max}}\) (default value 50 [mm], for nacelle 20 [mm], for farfield/symmetry plane 1 [m]) and the maximal curvature angle \(\alpha_{\text{max}}\) (default 10 [°]). Additionally, the following CAD curves are rastered:

\(^3\)An OO library developed by the first author for the MPI-parallel handling of hybrid unstructured meshes including partitioning, decomposition, communication, etc.

\(^4\)Duplicated elements in the overlap area of a partition owned from another process
- wing trailing edge ($L_{\text{min}} = 0.75$ [mm], $L_{\text{max}} = 1.5$ [mm]),
- wing root edge ($L_{\text{min}} = 2.5$ [mm], $L_{\text{max}} = 5.0$ [mm]),
- and wing tip edge ($L_{\text{min}} = 0.5$ [mm], $L_{\text{max}} = 1.0$ [mm]).

We emphasize that not a single source is specified for this case. Figure 13 presents the surface mesh of the generic transport aircraft: the half configuration consists of 322204 triangles (final surface mesh). The initial surface mesh without blanking out solids contained 330320 triangles. Details of the wing tip region are shown in Figure 14. Furthermore, in Figures 15 and 16 the nacelle coating is opened, thus the inner part of the engine is revealed. Here, Figure 16 shows the detail of Figure 15, which is marked by the black circle. In Figure 17 timing measurements are given for parallel rasterisation and surface meshing. All runs had been performed on a PC cluster system running with XEON 2.67 GHz processors connected via gigabit ethernet. Although only a parallel meshing speed-up of about four can be obtained for 16 processors, the

Figure 13: Surface triangulation for generic transport aircraft.

Figure 14: Wing tip of generic transport aircraft.

Figure 15: Inside view of nacelle for generic transport aircraft.

Figure 16: Detail of inside view of nacelle for generic transport aircraft.

Figure 17: Performance figures for generic transport aircraft.
important total time (also including I/O operations, ... ) to get a surface mesh for the complete configuration starting from a geometry without any sources is reduced down to less than five minutes.

5.2 Advanced Fighter-Type Aircraft

The geometry definition of this example consists of 6788 NURBS curves and 2749 NURBS surfaces and is imported via the STEP-format. Input parameters (for CAD surfaces) specified by the user were the minimal arc length \( L_{\text{min}} \) (default value 4 [mm]), for air data sensors 0.5 [mm], for nearfield 0.5 [m]), the maximal arc length \( L_{\text{max}} \) (default value 100 [mm]), for air data sensors 5 [mm], for nearfield 5 [m] ) and the maximal curvature angle \( \alpha_{\text{max}} \) (default 15 [°]). Again, no sources were used. The surface grid (containing 1.6 million triangles) shown in shaded mode in Figure 18 is thus solely based on the edge lengths calculated by the rasterisation process. In Figure 19 timing me-

Figure 18: Front view of advanced fighter-type aircraft.

surements are given for parallel rasterisation and surface meshing. All runs had been performed on a PC cluster system running with XEON 2.67 GHz processors connected with a QUADRICS network. Speed-ups of more than one order of magnitude are obtained for both the rasterisation and the surface meshing. The most important result is the reduction of the total time needed to get a surface grid starting from the watertight geometry. For 32 processors, the turnaround time can be reduced from about 1:10 hours down to 10 minutes.

6. CONCLUSIONS

The recently developed ST++ system has been presented. The OO design and implementation of the system was described together with the three major components, the geometry definition, the mesh size specification and the surface meshing itself. Based on a rasterisation of the geometry an automatic way of determining a smooth distribution of the element sizes in 3-D space was highlighted. To achieve fast turnaround times, the computationally intensive parts can be executed in parallel. Especially for large complex configurations containing thousands of geometric entities the turnaround time can be reduced greatly by the presented approach as shown in the examples.

ACKNOWLEDGEMENTS

We would like to thank EADS Military Aircraft, Ottobrunn, and all colleagues in the numerical simulation department for the support.

Special acknowledgments have to be given to Luciano Fornasier, Stephan M. Hitzel, Kaare A. Sorensen and Herbert Rieger for the great support concerning different topics of this work.

References


INTERPOLATION FROM A CLOUD OF POINTS

Timothy J. Baker

Dept. of MAE, Princeton University, Princeton, NJ 08544, U.S.A. baker@tornado.princeton.edu

ABSTRACT

Let \( V = \{P_1, P_2, \ldots, P_n\} \) be a set of points in either 2D or 3D space and let \( \{q_1, q_2, \ldots, q_m\} \) be scalar values associated with the points. This paper presents a method for interpolating values of the scalar variable \( q \) at any position \( X \) in the convex hull of \( V \). The interpolant consists of the sum of the linear interpolant for a simplex \( T \) that contains \( X \) and a least squares estimate of the higher order terms. The least squares fit is made through the cloud of \( m \) points in \( V \) that are closest to \( X \) and are not already vertices of \( T \). Conditions that determine the invertibility of the least squares system are examined and related to geometric constraints on the positions of points in the cloud.

Keywords: interpolation, solution projection, least squares estimation

1. INTRODUCTION

Data interpolation from a discrete set of points in either 2D or 3D space is required in many situations that arise when solving partial differential equations on a mesh of points. Time dependent problems, for example, often require the projection of a solution at a given time instant from one mesh onto a new, modified mesh that has been adapted for the computation at the next time step. Another example arises with the overset method which uses several overlapping meshes to represent a given domain with a consequent need to transfer the solution data between the component meshes. Graphical interpretation and feature detection for large data sets such as flowfield measurements also relies on accurate interpolation from scattered data \cite{1}. If the donor mesh possesses a high degree of regularity (i.e. with a locally well defined set of coordinate directions) then interpolation by tensor splines or transfinite interpolation is possible. If the donor mesh consists of arbitrarily placed points, however, it will be necessary to devise data interpolation or reconstruction schemes that do not make any \textit{a priori} assumptions about the underlying mesh geometry.

This problem has received much attention among protagonists of meshless methods \cite{2, 3, 4, 5, 6} (also known as \( h p \) clouds and partition of unity methods). Data estimation is typically carried out by a moving least squares method \cite{7, 8} which assigns an influence function to each mesh point. The domain of influence, over which this function is non-zero, extends a finite distance from the mesh point. Data reconstruction at any given position in space is then achieved by summing the contributions from of all domains of influence that enclose the particular position.

The method proposed in this paper starts from a representation of the interpolated value as a linear interpolant over a simplex whose vertices are points in the donor mesh. The linear interpolant is then augmented by a higher order estimate that is obtained from nearest point neighbors outside this simplex. Data projection is specifically designed to interpolate the exact value at each donor mesh point.

2. PROBLEM STATEMENT AND FORMULATION

Given a set of randomly distributed points \( V = \{P_1, P_2, \ldots, P_n\} \) with associated scalar values \( \{q_1, q_2, \ldots, q_m\} \) interpolate a value \( q(x) \) at any given position \( x \) within the convex hull of \( V \). It is required that \( q(x_i) = q_i \) for \( i = 1,\cdots,n \) where \( x_i \) is the position vector associated with point \( P_i \).
Let $T$ be a containing simplex whose vertices are points in $V$ and such that $T$ contains the point $\mathbf{x}$. If there is a triangulation associated with the donor mesh points $V$ then this can be searched to find the unique simplex containing $\mathbf{x}$. If no triangulation has been defined, a containing simplex can be constructed by searching through the point cloud until a suitable set of vertices has been found. In either case, a fast search procedure (e.g., use of an octree data structure [9]) will enable the closest point $P \in V$ to be found in $O(\log n)$ time where $n = \text{card} V$. If a triangulation of $V$ has been defined then it is possible to find the containing simplex in an additional $O(1)$ time. If no a priori triangulation of the convex hull of $V$ is given then one may first create a triangulation of $V$, a procedure that is reasonable if the set $V$ of mesh points is not too large. If $n = \text{card} V$ is extremely large and the number of positions at which interpolated values are needed is relatively small, it may be preferable to create a containing simplex for each interpolated position $\mathbf{x}$ by a gift wrapping procedure.

Let

$$q(\mathbf{x}) = q_{loc}(\mathbf{x}) + f(\mathbf{x})$$

(1)

where $q_{loc}(\mathbf{x})$ is the interpolant obtained by a linear fit through the vertices of the containing simplex $T$. The function $f(\mathbf{x})$ is a higher order estimate of the error between the true function value and the linear interpolant. This estimate is obtained from a least squares fit through the nearest neighbors among the point set $V$. The interpolation can be carried out to arbitrarily high order. Although continuity of the derivatives is not guaranteed, tests of the reconstruction procedure indicate that the requisite degree of smoothness is obtained in practice.

### 2.1 Linear Interpolant

Considering first the planar case, let the containing triangle $T$ be defined by the vertices $R_1, R_2, R_3 \in V$ and let $X$ be the point with coordinates $(x, y)$ at which an interpolated value $q(x, y)$ is required. Let $R_i$ have coordinates $(x_i, y_i)$, $i = 1, 2, 3$ and define the linear basis functions $\phi_1, \phi_2, \phi_3$ such that $\phi_i(x_i, y_i) = \delta_{ij}$, $i, j = 1, 2, 3$ where $\delta_{ij}$ is the Kronecker delta

$$\delta_{ij} = \begin{cases} 
1, & i = j \\
0, & i \neq j
\end{cases}$$

(2)

The basis functions correspond to the barycentric coordinates associated with the triangle $T$. Thus $\phi_i(x, y) = A_i/A$ where (see figure 1) $A_i$ is the area of triangle $XR_iR_j$, with $A_1$ and $A_3$ defined in a similar way, and $A = A_1 + A_2 + A_3$ is the area of triangle $R_1R_2R_3$. The linear interpolant is then given by

$$q_{loc}(x, y) = q_1\phi_1(x, y) + q_2\phi_2(x, y) + q_3\phi_3(x, y)$$

(3)

Since the linear interpolant must be exact if $q(x, y)$ is a constant, it follows that

$$\phi_1(x, y) + \phi_2(x, y) + \phi_3(x, y) = 1$$

(4)

Similarly, the requirement that $q(x, y) = x$ and $q(x, y) = y$ be represented exactly by the linear interpolant leads to the equations

$$x_1\phi_1(x, y) + x_2\phi_2(x, y) + x_3\phi_3(x, y) = x$$

(5)

and

$$y_1\phi_1(x, y) + y_2\phi_2(x, y) + y_3\phi_3(x, y) = y$$

(6)

It follows that explicit expressions for the basis functions can be determined by inverting the system of equations

$$\begin{pmatrix}
1 & 1 & 1 \\
x_1 & x_2 & x_3 \\
y_1 & y_2 & y_3
\end{pmatrix}
\begin{pmatrix}
\phi_1 \\
\phi_2 \\
\phi_3
\end{pmatrix}
= 
\begin{pmatrix}
1 \\
x \\
y
\end{pmatrix}$$

(7)

The determinant of the above matrix equals twice the area of triangle $R_1R_2R_3$. Thus, the basis functions are well defined provided the area of the triangle is nonzero.

**Figure 1:** Definition of areas $A_1, A_2$ and $A_3$ associated with the point $X$.

### 2.2 Estimation of Quadratic Error Terms

In order for the interpolant to remain exact at the vertices of $T$, it is necessary to obtain an estimate of
the quadratic error \( f(x, y) \) that is zero at each vertex of \( T \) (i.e., such that \( f(x_j, y_j) = 0 \), \( j = 1, 2, 3 \)). This can be achieved in terms of the linear basis functions by representing the error as
\[
f(x, y) = a\phi_1\phi_2 + b\phi_2\phi_3 + c\phi_3\phi_1 \tag{8}
\]
Since each basis function is a linear function of \( x \) and \( y \) it follows that any product of two basis functions must be a quadratic function of \( x \) and \( y \). The three pairs of basis functions that appear in equation (8) represent the three distinct pairs which are identical zero at each vertex of \( T \). To see this, note that \( \phi_1 = 0 \) on the extended edge \( R_1R_2 \) while \( \phi_2 = 0 \) on the extended edge \( R_1R_3 \) so that \( \phi_1\phi_2 \) is zero at each vertex of \( T \). In a similar way, it can be seen that \( \phi_2\phi_3 \) and \( \phi_3\phi_1 \) are zero at each vertex of \( T \). It follows that \( f(x_j, y_j) = 0 \), \( j = 1, 2, 3 \).

Now let \( S_j \in V - \{R_1, R_2, R_3\}, j = 1, \ldots, m \) be the next \( m \) donor mesh points that are closest to \( X \). Let \( \phi_i(j) \) represent the value of \( \phi_i \) at the data point \( S_j \). Similarly, let \( q(j) \), respectively \( q_{i,n}(j) \), be the values of the data, respectively linear interpolant, at the data points \( S_j \), \( j = 1, \ldots, m \). Define the matrix
\[
B = \begin{pmatrix}
\phi_1(1)\phi_2(1) & \phi_2(1)\phi_3(1) & \phi_3(1)\phi_1(1) \\
\phi_1(2)\phi_2(2) & \phi_2(2)\phi_3(2) & \phi_3(2)\phi_1(2) \\
\vdots & \vdots & \vdots \\
\phi_1(m)\phi_2(m) & \phi_2(m)\phi_3(m) & \phi_3(m)\phi_1(m)
\end{pmatrix}
\tag{9}
\]
The coefficients \( a, b, c \) are determined by computing the least squares approximation of the error terms for the \( m \) extra points. Thus, the coefficients are found by inverting the \( 3 \times 3 \) system [10]
\[
B^T B a = B^T w \tag{10}
\]
where \( a = (a, b, c)^T \) and
\[
w = \begin{pmatrix}
q(1) - q_{i,n}(1) \\
q(2) - q_{i,n}(2) \\
\vdots \\
q(m) - q_{i,n}(m)
\end{pmatrix}
\tag{11}
\]
The interpolation procedure generalizes in a straightforward manner to 3D. In this case,
\[
q(x, y, z) = q_{i,n}(x, y, z) + f(x, y, z) \tag{12}
\]
and
\[
q_{i,n}(x, y, z) = q_1\phi_1(x, y, z) + q_2\phi_2(x, y, z) \\
+ q_3\phi_3(x, y, z) + q_4\phi_4(x, y, z) \tag{13}
\]
where \( \phi_i(x, y, z), i = 1, 2, 3, 4 \) are the linear basis functions associated with the tetrahedron \( T \) that contains the point \( X \). In an analogous manner, explicit expressions for these four linear basis functions can be found by inverting the system of equations
\[
\begin{pmatrix}
1 & 1 & 1 & 1 \\
x_1 & x_2 & x_3 & x_4 \\
y_1 & y_2 & y_3 & y_4 \\
z_1 & z_2 & z_3 & z_4
\end{pmatrix}
\begin{pmatrix}
\phi_1 \\
\phi_2 \\
\phi_3 \\
\phi_4
\end{pmatrix}
= \begin{pmatrix}
1 \\
x \\
y \\
z
\end{pmatrix} \tag{14}
\]
The quadratic error is now represented as
\[
f(x, y, z) = a\phi_1\phi_2 + b\phi_2\phi_3 + c\phi_3\phi_1 \\
+ d\phi_2\phi_3 + e\phi_2\phi_4 + f\phi_3\phi_4 \tag{15}
\]
By a similar argument to that given earlier, it can be seen that the six pairs of basis functions appearing in equation (15) are all zero at each of the four vertices of the tetrahedron \( T \). The least squares approximation leads to the system,
\[
C^T a = C^T w \tag{16}
\]
where \( C \) is an \( m \times 6 \) matrix, each of whose rows is formed from the six distinct products of two basis functions evaluated at one of the \( m \) points \( S_j \). The vector \( a \) is the column vector containing the 6 coefficients \( a, b, c, d, e, f \) and \( w \) is defined as before. Inversion of the \( 6 \times 6 \) matrix \( C^T C \) is required to obtain the coefficients.

### 2.3 Higher Order Interpolants

The procedure generalizes in a natural way to permit cubic and higher order estimation of the error terms. In the planar case, the error up to and including third order terms is given by
\[
f(x, y) = a\phi_1\phi_2^2 + b\phi_2\phi_3^2 + c\phi_3\phi_1^2 + d\phi_2\phi_3 + e\phi_2\phi_4 + f\phi_3\phi_4 \tag{17}
\]
where the seven triple products of basis functions are the set of distinct products that are cubic in \( x \) and \( y \) and are identically zero at the vertices of the triangle \( T \). Determination of the coefficients by least squares leads to a \( 7 \times 7 \) system of equations to invert. In a similar way one can estimate the error terms up to fourth order accuracy with an expression for \( f(x, y) \) that is formed as a linear combination of the 12 distinct quadruple products of the basis functions that are zero at each vertex of \( T \). It follows in this case that there are 12 coefficients to be computed by solving a \( 12 \times 12 \) system of equations. Third order and fourth order accuracy in 3D requires the inversion of a \( 16 \times 16 \) and a \( 31 \times 31 \) system respectively.

### 3. Size of Point Cloud

The number \( m \) of mesh points \( S_j, j = 1, \ldots, m \) that are used to determine the least squares estimate should not be too large in order to maintain a compact support for the evaluation of the error term. Too few
points, on the other hand, will result in a covariance matrix $B^TB$ (or $C^TC$ in 3D) that is singular. Although the least squares system of equations is consistent and therefore always has a solution, a non-singular covariance matrix ensures that the least squares solution is unique.

In principle, one could handle the singular case by choosing the minimum length solution, or which is equivalent, by taking the pseudo inverse $a = B^+w$ [11]. Evidently, $m$ cannot be larger than $n = \text{card } V$ so that the pseudo inverse should be used if the point set $V$ is extremely small and one is, in effect, trying to interpolate through an insufficiently large set of data points. In general, however, the size of $V$ will not be a limitation and the number $m$ of data points used in the least squares fit should be chosen to be sufficiently large to ensure invertibility of the covariance matrix.

In order to determine conditions under which the covariance matrix $B^TB$ will be singular it should first be noted that $B^TB$ has the same null-space as $B$ [10]. This follows from the observation the nullspace of $B$ is contained in the nullspace of $B^TB$ and vice versa. First $Bx = 0 \Rightarrow B^TBx = 0 \Rightarrow N(B) \subset N(B^TB)$. Conversely, $B^TBx = 0 \Rightarrow x^TB^TBx = 0$ so that $||Bx||^2 = 0 \Rightarrow Bx = 0$. Hence $N(B^TB) \subset N(B)$ and these two inclusions show that $N(B^TB) = N(B)$.

It follows that $\text{rank } B^TB = \text{rank } B$ and the number of linearly independent columns of $B^TB$ is therefore the same as the number of linearly independent columns of $B$. For the planar case with quadratic error estimation, the matrix $B$ is given by equation (9) and hence invertibility of $B^TB$ requires $m \geq \text{rank } B = 3$.

### 3.1 Condition for a Diagonal Covariance Matrix

An example for which $m = 3$ is sufficient is shown in figure 2. The point $S_1$ lies on the extended edge $R_1R_1$ so that $\phi_1(1) = 0$. Similarly, $\phi_2(2) = 0$ since $S_2$ lies on the extended edge $R_2R_2$ and $\phi_3(3) = 0$ since $S_3$ lies on the extended edge $R_3R_3$. Hence

$$B = \begin{pmatrix}
\phi_1(1) & \phi_2(1) & 0 & 0 \\
0 & \phi_2(2) & \phi_3(2) & 0 \\
0 & 0 & \phi_3(3) & \phi_1(3)
\end{pmatrix}
$$

and

$$B^TB = \begin{pmatrix}
\phi_1^2(1) & \phi_2^2(1) & 0 & 0 \\
0 & \phi_2^2(2) & \phi_3^2(2) & 0 \\
0 & 0 & \phi_3^2(3) & \phi_1^2(3)
\end{pmatrix}
$$

In this particular case the matrix $B^TB$ will be diagonal, a property that does not hold unless the three extra data points $S_1, S_2, S_3$ all lie on extended edges of the containing triangle $T$. More generally, if the point $S_j, j = 1, \ldots, m$ lies on the extended edge of the containing triangle $T$ opposite vertex $R_i$ where $i = 1, 2$ or $3$ then $\phi_i(j) = 0$. Hence each row of $B$ has two zero entries and just one non-zero entry. It follows that the columns of $B$ are orthogonal so that $B^TB$ will be diagonal.

**Remark 1:** If every point $S_j, j = 1, \ldots, m$ lies on an extended edge of the containing triangle $T$ then $B^TB$ will be diagonal. If, in addition, there is at least one point $S_j$ on each of the three extended edges then $B^TB$ will be invertible.

**Figure 2:** A case when three points suffice for invertibility.

### 3.2 A Condition when Three Extra Points is Insufficient

The more interesting situation, or at least the situation of greater concern, is associated with $B^TB$ being singular. Figure 3 illustrates a situation when three extra points $S_1, S_2, S_3$ are not sufficient to make $B$ and hence $B^TB$ nonsingular. If two data points, say $S_1$ and $S_2$, lie on the same extended side, say $R_1R_1$, then $\phi_1(1) = 0$ and $\phi_2(2) = 0$. It follows that

$$B = \begin{pmatrix}
\phi_1(1) & \phi_2(1) & 0 & 0 \\
\phi_1(2) & \phi_2(2) & 0 & 0 \\
\phi_1(3) & \phi_2(3) & \phi_3(3) & \phi_1(3)
\end{pmatrix}
$$

Hence $B$ and therefore $B^TB$ have rank 2. The deficiency in rank occurs as a result of the fact that the linear basis function $\phi_i$ is zero on the extended edge of $T$ that is opposite vertex $R_i$.

In general, if there are $m$ extra data points of which the first $m - 1$ lie on an extended edge of $T$, say $R_1R_1$...
so that \( \phi_\lambda(j) = 0, \ j = 1, \cdots, m - 1 \) then

\[
B = \begin{pmatrix}
\phi_1(1)\phi_2(1) & 0 & 0 \\
\phi_1(2)\phi_2(2) & 0 & 0 \\
\vdots & \vdots & \vdots \\
\phi_1(m - 1)\phi_2(m - 1) & 0 & 0 \\
\phi_1(m)\phi_2(m) & \phi_2(m)\phi_3(m) & \phi_3(m)\phi_1(m)
\end{pmatrix}
\]

(21)

and

\[
B^T B = \begin{pmatrix}
ce_1 & c_2 & c_3
\end{pmatrix}
\]

(22)

where the columns are given by

\[
ce_1 = \left( \sum \phi_i^2 \phi_j^2 \right) \phi_k(m)
\]

(23)

\[
ce_2 = \left( \phi_1(m)\phi_2(m)^2 \phi_3(m) \right) \\
\phi_2(m)^2 \phi_3(m) \\
\phi_1(m)\phi_2(m)\phi_3(m)
\]

(24)

\[
ce_3 = \left( \phi_1(m)^2 \phi_2(m)\phi_3(m) \right) \\
\phi_2(m)^2 \phi_3(m) \\
\phi_1(m)^2 \phi_2(m)\phi_3(m)
\]

(25)

As expected the covariance matrix \( B^T B \) has rank 2 since columns 2 and 3 are linearly dependent.

![Figure 3: A example when three points do not suffice for invertibility.](image)

We summarize this result as

**Remark 2:** The covariance matrix \( B^T B \) will be singular if more than \( m - 2 \) of the \( m \) extra data points \( S_j, j = 1, \cdots, m \) lie on one extended edge \( e \) of the containing triangle \( T \).

### 3.3 A Condition for Matrix \( B \) to have linearly dependent columns

It is also possible for \( B \) and hence \( B^T B \) to be singular if any two columns of \( B \) are linearly dependent. This can only arise if all \( m \) points lie on a line through a vertex of the containing triangle \( T \). This possibility is illustrated in figure 4. Suppose, for example, that the extra data points all lie on a straight line \( L \) through a vertex of the containing triangle \( T \). In particular, as shown in figure 4, let \( R_3 \) be the vertex through which \( L \) passes. Then, if \( S \) is any data point on \( L \), it follows that

\[
\phi_1(S) = \frac{A_1}{A}, \quad \phi_2(S) = \frac{A_2}{A}
\]

(26)

with

\[
A_1 = \frac{l_1}{2} l_1, \quad A_2 = \frac{l_2}{2} l_2
\]

(27)

Here, \( A_1 \) is the area of triangle \( R_3 SR_3 \), \( A_2 \) is the area of triangle \( R_3 SR_1 \), \( l_1 \) is the length of edge \( R_2R_3 \), \( l_2 \) is the length of edge \( R_3R_1 \) and \( h_2 \), respectively \( h_2 \), is the length of the perpendicular from \( S \) to the extended edge \( R_2R_3 \), respectively \( R_2R_1 \). Now let \( \theta \) be the angle between line \( L \) and the extended edge \( R_3R_1 \) and let \( \psi \) be the angle between \( L \) and the extended edge \( R_2R_3 \). It follows that

\[
h_1 = d \sin \psi \quad \text{and} \quad h_2 = d \sin \theta
\]

(28)

Hence

\[
\phi_2(S) = \alpha \phi_1(S) \quad \text{where} \quad \alpha = \frac{l_2 \sin \theta}{l_1 \sin \psi}
\]

(29)

Since \( \alpha \) does not depend on the distance \( d \) of \( S \) from the vertex \( R_3 \), this relation must be true for any position \( S \) on \( L \). It follows that the second and third columns of \( B \) are linearly dependent so that \( B^T B \) will be singular.

![Figure 4: Colinearity condition for a rank deficient matrix.](image)

**Remark 3:** The matrix \( B^T B \) will be singular if the \( m \) points \( S_j, j = 1, \cdots, m \) are colinear and lie on a line passing through a vertex of the containing triangle \( T \). For a general arrangement of points it is unlikely that these pathological situations will arise. It is possible, however, that the covariance matrix \( B^T B \) will be badly conditioned if the number \( m \) of extra data points
is small and the arrangement of donor mesh points has a lattice organization as illustrated in figure 9. In practice, a well conditioned covariance matrix is usually assured by taking \( m \) equal to twice the number of columns in the matrix \( B \). Since the system of normal equations (eqns. (10) or (16)) is non-negative definite, inversion can be accomplished by a Cholesky decomposition and the determinant and/or condition number of \( B^TB \) monitored to detect singular behavior. If this does occur, further data points can be acquired until the system of equations does become well conditioned.

\[ \phi_1(x, y) = 1 - x, \quad \phi_2(x, y) = x - y, \quad \phi_3(x, y) = y \] (30)

whence

\[ q_{\phi}(x, y) = 1 - x \] (31)

The linear interpolant over the other seven triangles is easily obtained and is displayed for the entire mesh in figure 6.

If the nearest three extra points \( S_1, S_2 \) and \( S_3 \) are used to obtain the quadratic correction we find that

\[ B \mathbf{a} = \mathbf{w} \] (32)

where

\[ B = \begin{pmatrix} -1 & -1 & 1 \\ 0 & -2 & 0 \\ 1 & -1 & -1 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix} \] (33)

In this case \( \text{rank} \, B = 2 \) so that the system of normal equations (10) is singular. Using Householder transformations \([11]\) we obtain

\[ \mathbf{R} \mathbf{y} = \mathbf{g} \] (34)

where \( \mathbf{R} = \mathbf{Q} \mathbf{B} \mathbf{K} \) and \( \mathbf{R} \) has the form

\[ \mathbf{R} = \begin{pmatrix} R_{11} & 0 \\ 0 & 0 \end{pmatrix} \] (35)

where \( R_{11} \) is a \( 2 \times 2 \) upper triangular matrix. \( \mathbf{Q} \) and \( \mathbf{K} \) are orthogonal matrices and \( \mathbf{y} = \mathbf{K}^T \mathbf{a}, \quad \mathbf{g} = \mathbf{Q} \mathbf{w} \).

For this particular example

\[ R_{11} = \begin{pmatrix} -2 & 0 \\ 0 & \sqrt{6} \end{pmatrix}, \quad \mathbf{g} = \begin{pmatrix} 0 \\ \sqrt{2} \end{pmatrix} \] (36)

and

\[ \mathbf{K} = \begin{pmatrix} -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{pmatrix} \] (37)

Let \( \mathbf{y} = (y_1, y_2, y_3) \). Solving the system

\[ \begin{pmatrix} -2 & 0 \\ 0 & \sqrt{6} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \sqrt{2} \end{pmatrix} \] (38)

gives \( y_1 = 0, \ y_2 = \frac{2}{\sqrt{3}} \). To obtain the pseudo inverse we set \( y_3 = 0 \) and then compute \( \mathbf{a} = \mathbf{K} \mathbf{y} \) to give

\[ a = c = 0, \quad b = \frac{1}{3}. \] (39)

Substituting the values of these coefficients into equation (8) and using the linear basis functions given in equations (30) we find that the quadratic correction based on the three nearest extra points is

\[ f_3(x, y) = \frac{1}{3} y(x - y) \] (40)

![Figure 5: Mesh for the nine point test example.](image)

![Figure 6: Linear interpolant for nine point test example.](image)
and the corresponding second order interpolant is

\[ q_2(x, y) = 1 - x + \frac{1}{3}y(x - y) \]  

(41)

The corresponding interpolants for the remaining seven triangles can be easily found by symmetry considerations. Figure 7 shows this interpolant for the entire mesh.

![Figure 7: Quadratic interpolant using three extra points and the pseudo-inverse.](image)

When four extra points \( S_1, S_2, S_3 \) and \( S_4 \) are used we find that

\[
B = \begin{pmatrix}
-1 & -1 & 1 \\
0 & -2 & 0 \\
1 & -1 & -1 \\
-4 & -2 & 2 \\
\end{pmatrix}
\]  

(42)

which has rank 3 so that \( B^TB \) is invertible. In this case we have

\[
a = c = \frac{2}{3}, \quad b = \frac{1}{3}
\]

leading to the quadratic interpolant over triangle \( R_1R_2R_3 \) given by

\[ q_4(x, y) = 1 - x + \frac{2}{3}(1 - x) + \frac{1}{3}y(x - y) \]  

(44)

The corresponding interpolant \( q_5(x, y) \) based on five extra points has the coefficients

\[
a = \frac{13}{14}, \quad b = \frac{2}{7}, \quad c = \frac{15}{14}
\]

(45)

while the interpolant \( q_6(x, y) \) based on all six extra points has the coefficients

\[
a = \frac{48}{53}, \quad b = \frac{15}{53}, \quad c = \frac{54}{53}
\]

(46)

The interpolant \( q_4(x, y) \) based on four extra points is displayed for the entire mesh in figure 8.

![Figure 8: Quadratic interpolant using four extra points for the least squares fit.](image)

![Figure 9: Point set \( V \) (filled in circles) and interpolation positions (open circles).](image)

5. A SMOOTHLY VARYING TEST CASE

A simple test case, using a smoothly varying function to define the data values, illustrates the efficacy of the procedure for different orders of interpolation. A donor mesh (see figure 9) was defined by a regular lattice of points covering a square whose sides have unit length. The minimum spacing, or lattice width, \( h \) between mesh points was varied in order to investigate how well the accuracy of the interpolation schemes improved as \( h \) was reduced in size. The data values assigned to the mesh points were given by

\[ q(x, y) = \left( \sin\frac{\pi}{2}x \sin\frac{\pi}{2}y \right)^2 \]

(47)

a function that varies smoothly between 0 and 1.

Interpolated values were obtained at a series of points along a line that ran across the mesh and such that the interpolation positions were as far as possible from the donor mesh points. The interpolated values were compared with the exact values to determine the interpolation error at each sample point and the root mean
square (rms) value of the error at all sample points was computed. Figure 10 shows the rms error versus mesh width \( h \) for the linear interpolation as well as for second order, third order and fourth order interpolation. The rate at which the error diminishes as \( h \) becomes smaller shows clearly how the higher order interpolation schemes provide superior performance albeit by requiring the inversion of larger matrix systems than the lower order schemes.

A similar comparison in 3D was made for a three dimensional lattice of points on the unit cube whose data values were given by

\[
q(x, y) = (\sin \frac{\pi}{2} x \sin \frac{\pi}{2} y \sin \frac{\pi}{2} z)^2
\]

The root mean square error versus mesh width \( h \) for the linear interpolation as well as for second order, third order and fourth order interpolation is shown in figure 11. The trend of error reduction versus lattice spacing \( h \) is similar to that demonstrated for the 2D case in figure 10.

6. INTERPOLATION THROUGH A STEP FUNCTION

A more severe test is provided by data values which represent a discontinuous jump. In the 2D case the following step function was therefore used to define the data values \( q \) at each lattice point on the unit square.

\[
q(x, y) = \begin{cases} 
1 & \text{if } x < \frac{1}{2} \\
0 & \text{if } x \geq \frac{1}{2}
\end{cases}
\]

A cut through the interpolating surface along the line \( y = 0.5 \) is shown in figure 12 over part of the \( x \) axis for data defined on the donor mesh at a lattice spacing of \( h = 0.025 \). The linear interpolant decreases from a value of 1 at \( x = 0.475 \) to zero at \( x = 0.5 \). The second order interpolant displays an overshoot ahead of the step jump and an undershoot after the step jump whose magnitude is around 10% of the step height. The overshoots extend roughly one half of the lattice spacing \( h \) on either side of the discontinuity. (Note that the symbols shown on the curves in figures 12 and 13 do not represent actual mesh positions which are much more widely spaced and at which the function values \( q(x, y) \) are, of course, interpolated exactly). The cubic interpolant is smoother with an overshoot and undershoot that is about half the amplitude of that for the quadratic interpolant but which extends about twice as far (i.e. a whole lattice spacing \( h \) before and after the discontinuity). Figure 13 shows the corresponding result for a lattice width \( h = 0.00625 \). The results, as one would expect, are similar to the previous comparison showing overshoots and undershoots.
of comparable magnitude and extending the same distance when scaled by the lattice spacing.

7. ACKNOWLEDGMENT

Part of this research was carried out in the course of consulting activity for Combustion Research and Flow Technology, Inc. and was funded by the Air Force Research Laboratory, Wright-Patterson AFB, OH under contract F33615-02-C-3215, Phase II SBIR (technical monitors Frank Witzeman and Matthew Grismer). The support is gratefully acknowledged.

References


Session 1B
Mesh Refinement
MESH REFINEMENT BASED ON THE 8-TETRAHEDRA LONGEST-EDGE PARTITION

Ángel Plaza1 Marı́a-Cecilia Rivara2

1 University of Las Palmas de Gran Canaria, Spain, aplaza@dmat.ulpgc.es
2 DCC, University of Chile, Santiago de Chile, Chile, mcrivara@dcc.uchile.cl

ABSTRACT

The 8-tetrahedra longest-edge (8T-LE) partition of any tetrahedron is defined in terms of three consecutive edge bisections, the first one performed by the longest-edge. The associated local refinement algorithm can be described in terms of the polyhedron skeleton concept using either a set of precomputed partition patterns or by a simple edge-midpoint tetrahedron bisection procedure. An effective 3D derefinement algorithm can be also simply stated. In this paper we discuss the 8-tetrahedra partition, the refinement algorithm and its properties, including a non-degeneracy fractal property. Empirical experiments show that the 3D partition has analogous behavior to the 2D case in the sense that after the first refinement level, a clear monotonic improvement behavior holds. For some tetrahedra a limited decreasing of the tetrahedron quality can be observed in the first partition due to the introduction of a new face which reflects a local feature size related with the tetrahedron thickness.

Keywords: mesh refinement, longest-edge bisection, longest-edge algorithms, tetrahedral meshes

1. INTRODUCTION

Skeleton algorithms for local mesh refinement/derefinement of triangular and tetrahedral meshes have been proposed by Plaza and Carey [10, 11, 12]. In two dimensions, the algorithm is an alternative formulation of the 4-triangles longest-edge algorithm [14, 15]. The 2-dimensional skeleton algorithm [10, 11] works over the edges wireframe mesh affected by the refinement (target triangles and some neighbors to assure the construction of a conforming mesh) by performing midpoint bisection of the involved edges. Then this information is used to select the appropriate triangle partition pattern (between a set of three patterns) to refine each individual triangle. This idea was then generalized to 3-dimensions [11, 12] by introducing an 8-tetrahedra partition which induces the 4-triangles partition of its faces.

The 3-dimensional skeleton algorithm performs: (1) the refinement of the 3-dimensional edges wireframe mesh affected, (2) the refinement of the faces surface mesh (by using the 4-triangles partition and associated partial partitions), and (3) the volume refinement of each tetrahedron either by using a simple edge bisection procedure or according to an appropriate pattern, selected between a set of precomputed partition patterns.

In this paper we study the properties of the 8-tetrahedra partition showing that each full partition pattern is equivalent to a sequence of seven tetrahedron edge bisections by the midpoint of the tetrahedron edges, the first one being performed by the tetrahedron longest-edge. Then we take advantage from the improvement and fractal properties of the 4-triangles longest-edge partition to show some non-degeneracy properties in 3-dimensions. We also show that for the meshes globally refined by using the 8-tetrahedra partition, the asymptotic average number of tetrahedra sharing a fixed vertex is equal to 24.

An empirical study about the behavior of the 8-tetrahedra partition is also included. This shows that consistently, from the second refinement level, both
the distribution of quality tetrahedra, and the volume percentage covered by better tetrahedra tend to be improved as the 8T-LE partition proceeds.

1.1 Previous and related work

Refinement algorithms based on longest-edge partitions, including Lepp based algorithms, have been extensively discussed [14, 15, 18, 17, 16], as well as skeleton based algorithms [10, 11, 12, 13].

In two-dimensions it has been shown that these algorithms improve the point distribution by maintaining some small-angled triangles which depend on the quality of the initial mesh, in the following senses: the iterative global refinement of any triangle produces triangles whose minimum angle is bounded as a function of the quality of the initial triangle, the process produces a finite number of similarly distinct triangles, and both the percentage of good-quality triangles and the area covered by these triangles increases as the refinement proceeds.

In [18] a pure three dimensional longest-edge refinement method was considered. Empirical experimentation showing that the solid angle decreases slowly with the refinement iterations and that a quality-element improvement behavior, analogous to the 2-dimensional case holds in practice, were provided. However, there has not been mathematical results available guaranteeing the non-degeneracy properties of the 3-dimensional mesh.

In the last 12 years other triangle-bisection and tetrahedron-bisection refinement algorithms have been proposed. Between them we can cite the newest-vertex insertion method of Michell [9] in two dimensions, the tetrahedron-bisection algorithm of Bänisch [2] and the 8-tetrahedra bisection algorithm of Liu and Joe [7]. These algorithms essentially consist on performing edge based partitions in such a way that triangles or tetrahedra similar to those of the first refinement levels are obtained throughout the process. In particular, Liu and Joe have obtained a bound on the mesh quality as a function of the initial geometry for their algorithm [6].

Other studies report somewhat equivalent algorithms. A recursive approach which imposes certain restrictions and pre-processing in the initial mesh is proposed by Kossaczký [4]. Maubach [8] has developed and algorithm for n-simplicial grids generated by reflection. Although the algorithm is valid in any dimension and the number of similarity classes is bounded, this cannot be applied to a general tetrahedral grid, since an additional closure refinement is needed to avoid incompatibilities. Arnold et al. [1] have presented an algorithm equivalent to those discussed in [2, 4] proving its equivalence with [8].

All these algorithms however, do not take practical advantage of the element-quality improvement properties of longest-edge and skeleton algorithms. These algorithms, in exchange, can be applied to any valid initial triangulation without any restriction on the shape of the tetrahedra.

In what follows we specifically discuss the skeleton algorithms of Plaza and Carey [10, 11, 12]. This three dimensional approach is based on the application of the 2-dimensional algorithm over the skeleton of the 3D triangulation, that is to the set of the triangular faces of the tetrahedra. Being this a longest-edge based algorithm we expect for it analogous behavior to that reported in [18] for pure 3-dimensional longest-edge refinement algorithm.

2. THE 4-TRIANGLES ALGORITHM AND PREVIOUS RESULTS

The 4-Triangles algorithm can be described in terms of the three refinement patterns of Figure 1, where \( P \) is the midpoint of the longest-edge. The algorithm consists on two basic steps: (1) refinement of target triangles by using the partition pattern (a) of Figure 1, and (2) a local propagation step to assure a conforming mesh which uses the partition patterns of (b) and (c) of Figure 1.

\[
\text{4-Triangles-Refinement-Algorithm}(\tau, t)
\]

/* Perform the 4-Triangles partition of \( t \) for each edge \( e \) of \( t \), of associated neighbor \( t^* \) do
neighbor-refinement(\( t^* \), \( e \))
\( t \leftarrow t^* \)
while \( t \) is non-conforming do
find the unique non-conforming edge \( e \in t \) with associated neighbor \( t^* \)
neighbor-refinement(\( t^* \), \( e \))
\( t \leftarrow t^* \)
end while
end for

\[
\text{Figure 1: 4-Triangles-refinement patterns.}
\]
neighbor-refinement($t^*$, $e$)
if $e$ is longest-edge of $t^*$ perform LE bisection of $t^*$
else perform 3-Triangles partition of $t^*$ by edge $e$

For an illustration see Figure 2. Note that in the general case, the refinement should propagate to neighbor triangles by the edges $AC$ and $CB$.

Figure 2: Example of 4-Triangles-Refinement-Algorithm.

The 4-Triangles-Algorithm produces a subset of the triangles obtained by longest-edge bisection and the following theorem holds [14, 15]:

**Theorem 2.1** Over any conforming triangulation $\tau_0$, the iterative application of the 4-Triangles-Algorithm: (1) produces nested triangulations in the sense that each new triangle is embedded in its parent; (2) every triangle $t$ generated in the process has smallest angle greater or equal to $\frac{\alpha_0}{2}$, where $\alpha_0$ is the smallest angle of the triangle $t_0$ in $\tau$ which embeds $t$; (3) produces a finite number of similarity distinct triangles; (4) the triangulations obtained tend to be improved in the sense that both the percentage of the good-quality triangles and the area covered by these triangles increases as the refinement proceeds.

Furthermore, for obtuse triangles the following monotone improvement behavior holds [17]:

**Theorem 2.2** For any obtuse triangle $t_0$ of smallest angle $\alpha_0$ and largest angle $\gamma_0$, the 4-Triangles partition of $t_0$ produces a unique similarly distinct triangle $t_1$, whose 4-Triangles partition in turn produces a new similarly distinct triangle $t_2$, and so on, until a last non-obtuse triangle $t_n$ is obtained. Furthermore, the smallest angles $\alpha_i$ and the largest angles $\gamma_i$ of each triangle $t_i$ satisfy the following improvement relations:

$$\alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_n$$
$$\gamma_0 > \gamma_1 > \gamma_2 > \ldots > \gamma_n$$

where $\gamma_i = \gamma_{i-1} - \alpha_i$.

For the 4T-LE algorithm, a fractal property analogous to that proved for the LE-bisection algorithm [17] also holds:

**Theorem 2.3** After a finite number of iterative (local) applications of the 4-triangles algorithm around any vertex $P$ of any conforming triangulation $\tau$, a stable molecule around $P$ is obtained, in the sense that the next iteration of the algorithm do not divide the angles of vertex $P$, but only introduce new vertices along the edges of the stable molecule. Furthermore, each new triangle of vertex $P$ produced throughout the next iterations will be similarly equal to a preceding triangle.

Figure 3: Fractal behavior of 4-triangles partition and stable molecule.

2.1 The skeleton algorithms in two and three dimensions

The skeleton version of the 4-Triangles refinement algorithm performs the refinement task by using two sequential steps: (1) Identifying and bisecting the edges (not the triangles) involved throughout the overall refinement process; and (2) partitioning each individual triangle involved in the refinement process by using the triangles partitions of Figure 1 according to its bisected edges.

The 3D-skeleton algorithm in exchange generalizes the 4-Triangles refinement algorithm to 3-dimensions by
making use of the skeleton concept which in turn generalizes the graph usually associated with the polygonal faces of any polyhedron [10, 11, 3]:

**Definition 2.4** For any conforming 3D triangular mesh \( \tau \) (tetrahedral mesh), the 2D-Skeleton of \( \tau \) is the conforming surface mesh defined by the triangular faces of the elements of \( \tau \). In addition, the 1D-Skeleton of \( \tau \) is the conforming wire mesh defined by the edges of the elements of \( \tau \).

By using the preceding concepts the algorithm can be schematically described as follows:

**3D-Skeleton Refinement-Algorithm** \((\tau, t)\)
- Find and Partition involved Edges over 1-skeleton mesh
- Partition involved Faces over 2-skeleton mesh
- Partition involved Tetrahedra according appropriate partition patterns

Note that with minor changes, both procedures (over the 1-skeleton mesh and the 2-skeleton mesh) together correspond to the application of the 4-Triangles-Skeleton-Refinement Algorithm to the surface triangulation formed by the faces of the tetrahedra of the initial 3-dimensional triangulation. The Partition Tetrahedra procedure in exchange performs the volume partition of the set of tetrahedra whose faces were refined by the preceding procedures.

In the next Section we shall introduce and discuss the 8-Tetrahedra LE partition, proving the following properties: the 8-tetrahedra LE partition of every tetrahedron \( t \) in the mesh produces both a conforming volume mesh and a conforming surface mesh where the surface mesh is obtained by the 4-Triangles partition of the faces of \( t \).

### 3. THE 8-TETRAHEDRA PARTITION AND PROPERTIES

At this point, some definitions are in order:

**Definition 3.1** For any tetrahedron \( t \) of unique longest-edge, the primary faces of \( t \) are the two faces of \( t \) that share the longest-edge of \( t \). In addition, the two remaining faces of \( t \) are called secondary faces of \( t \). Furthermore, the secondary edges of \( t \) are the longest edges of the secondary faces of \( t \) (1 or 2 secondary longest edges). In addition, the 3 or 4 remaining edges of \( t \) are called third-class edges of \( t \).

Note that for any tetrahedron \( t \) of unique longest-edge, the primary faces of \( t \) have a common longest-edge equal to the longest-edge of \( t \). In order to avoid ambiguity in the general case, we always suppose that for each tetrahedron \( t \) having either a non-unique longest-edge, or non-unique secondary edges, a unique selection for each of such edges is performed a priori in such a way that the longest-edge of the tetrahedron coincides with the longest-edge of the primary faces of \( t \), and this selection is consistently maintained throughout the overall refinement process.

The 8-Tetrahedra longest-edge partition can be defined as follows:

**Definition 3.2** For any tetrahedron \( t \) of unique longest-edge and unique secondary edges, the 8-Tetrahedra Longest-Edge (8T-LE) partition of \( t \) is defined as follows:

1. LE-bisection of \( t \) producing tetrahedra \( t_1, t_2 \);
2. bisection of \( t_i \) by the midpoint of the unique edge of \( t_i \) which is also a secondary edge of \( t \), producing tetrahedra \( t_{ij} \) for \( i, j = 1, 2 \);
3. bisection of each \( t_{ij} \) by the midpoint of the unique edge equal to a third-class edge of \( t \), for \( i, j = 1, 2 \).

In order to study the 8-tetrahedra partition, we need to consider an intermediate 4-tetrahedra partition characterized by the following proposition:

**Proposition 3.3** Let \( t \) be any tetrahedron of unique longest-edge \( AB \) and associated midpoint \( P \) (see Figure 5. Then the 4-tetrahedra partition described by the two ordered steps (1) and (2) of Definition (3.2) produces a 4-tetrahedra volume triangulation of \( t \) satisfying the following properties:

a) The volume triangulation induces the longest-edge bisection of each triangular face of \( t \).

b) The volume triangulation of \( t \) will be a conforming triangulation if and only if the distribution of the longest-edge and secondary edges of \( t \) corresponds to either the cases (a), (b), or (c) in Figure 5.

c) The volume triangulation will not be a conforming triangulation if and only if the secondary edges share a vertex and one of these edges is opposite to the longest-edge of \( t \) (Figure 5 (d)).
The proof of part a) follows directly from the definition of the 4-tetrahedra partition, while the proof of parts b) and c) are based on the study of the possible relative positions of the longest-edge of \( t \) and the secondary edges of \( t \). Clearly, only 4 relative configurations, invariant under translation, rotation, reflection and uniform scaling are possible:

i) longest-edge of \( t \) opposite to the unique (common) secondary edge of the two secondary faces of \( t \) (Figure 5 (a)).

ii) the secondary longest-edges and longest-edge of \( t \) form a triangular face of \( t \) (Figure 5 (b)).

iii) opposite secondary edges, where each of such edges shares a vertex with the longest-edge of \( t \) (Figure 5 (c)).

iv) the secondary edges share a vertex and one of the secondary edges is opposite to the longest-edge of \( t \) (Figure 5 (d)).

**Corollary 3.4** The 4-tetrahedra partition of Theorem 3.3 produces four tetrahedra \( t_{ij} \) for \( i, j = 1, 2 \) such that each \( t_{ij} \) has a unique edge equal to a third-class edge of \( t \).

The next proposition proves that, for the 4 cases of Proposition 3.3 (Figure 6), the midpoint edge bisection of the new tetrahedra (by the non-bisected edge of \( t \)) produces a conforming volume triangulation of \( t \).

**Proposition 3.5** Let \( t \) be any tetrahedron having a unique longest-edge and unique secondary edges. Then if after applying the 4-tetrahedra partition defined in Proposition 3.3, each of the tetrahedra \( t_{ij} \) produced by this partition is in turn bisected by the midpoint of the (unique) edge equal to a third-class edge of \( t \), a conforming volume triangulation is obtained having the following properties:

a) The volume triangulation induces the 4-triangles partition of each face of \( t \).

b) Only an interior edge \( P^*P \) is produced, where \( P \) and \( P^* \) are respectively the midpoint of the longest-edge of \( t \), and the midpoint of the edge opposite to the longest-edge of \( t \).

c) Eight new internal faces appear inside the tetrahedron \( t \).

The results of previous proposition allow us to state Theorem 3.6:

**Theorem 3.6** The 8-tetrahedra longest-edge partition of any tetrahedron \( t \) produces both a conforming volume triangulation of \( t \) and a conforming surface triangulation of \( t \) such that:

1. The conforming surface triangulation of \( t \) is identical to the surface triangulation obtained by the 4-triangles partition of the faces of \( t \).

2. Four different triangulation patterns are obtained (Figure 7) according with the relative position of the longest-edge and the secondary edges of \( t \). Each one of these 4 patterns produces only one new internal edge \( P^*P \) (where \( P \) is the midpoint of the longest-edge of \( t \), and \( P^* \) is the midpoint of the edge opposite to the longest-edge) and 8 new internal faces.

Note that under the assumption that the longest-edge and the secondary edges are unique, there is a univocal correspondence between the four volume partition
patterns produced by the 8-tetrahedra partition of any tetrahedron $t$ and the four surface partition patterns obtained by the 4-triangles partition of the faces of $t$.

The careful study of the possible $n$-point partition patterns produced for the different relative positions of the longest-edge and secondary edges of $t$, for $n = 1, 2, \ldots, 6$ (which includes the four global 8-tetrahedra partition patterns) allows us to obtain the set of partial partition patterns involved in the mesh refinement algorithm. It can be proved that there are exactly 30 different partition patterns (invariant under translation, rotation, reflection, and uniform scaling) associated to the 8-tetrahedra partition of any tetrahedron.

## 4. THE 3D-SKELETON REFINEMENT/DEREFINEMENT ALGORITHM

The refinement algorithm for refining any tetrahedron $t$ in any conforming tetrahedral mesh $\tau$ can be formulated as follows:

**3D-Skeleton-Refinement-Algorithm($\tau$, $t$)**

`/* Find involved edges, faces, and tetrahedra */`

Initialize $S_E$, $S_F$, and $S_T$, respectively sets of involved edges, faces, and tetrahedra

Initialize $P_E$ set of processing edges

for each edge $E$ of $t$ do
  add edge $E$ to set $S_E$
  add edge $E$ to set $P_E$
endfor

While $P_E \neq \emptyset$, do

pick $E$ from $P_E$

for each tetrahedron $t'$ sharing edge $E$ do

  for each face $F$ of $t'$ having an edge in $S_E$ do
    find longest-edge $E^*$ of $F$
    if $E^*$ is not in $S_E$ do
      add $E^*$ to $S_E$
      add $E^*$ to $P_E$
      add $F$ to $S_F$
    endif
  endfor

add $t'$ to $S_T$
endfor

`/* Partition involved edges */`

for each edge $E$ in $S_E$ do

  create vertex $P$ midpoint of $E$
  bisect $E$
endfor

`/* Partition involved faces */`

for each edge $E$ in $S_F$ do

  partition $F$ according its bisected edges
endfor

`/* Partition involved tetrahedra */`

for each tetrahedron $T$ in $S_T$ do

  partition $T$ according to the partition of its faces
end for

The 3-dimensional skeleton refinement algorithm generalizes the 2-dimensional 4T-LE algorithm in the following sense:

**Theorem 4.1** The refined volume mesh obtained by the use of the 3D-Skeleton-Refinement-Algorithm induces the surface refinement of the associated 2D-Skeleton mesh and vice versa. Furthermore, the surface refined mesh is identical to the mesh obtained by applying the 4-triangles mesh refinement to the faces of $t$.

**Corollary 4.2** The 3-dimensional skeleton refinement algorithm is finite.

Note that the tetrahedra partition step can be implemented either by successive application of a basic tetrahedron bisection operation by an edge midpoint, or by precomputing a set of partition patterns. Also, an alternative algorithm working directly with the volume mesh (without using the mesh-skeleton concept) can be developed.

The derefinement algorithm works on the finite sequence of nested meshes obtained by the refinement algorithm application of the 4T-LE derefinement algorithm to the skeleton (working firstly both over the wireframe mesh and the triangular surface mesh), then followed by the
redetermination of the interior of the tetrahedra, for which a slight variation of the 3D refinement algorithm is used. For a further discussion see Plaza et al. in [12].

5. ON THE NON-DEGENERACY PROPERTIES OF THE 8T-LE REFINEMENT ALGORITHMS

Theorem 5.1  Let \( \tau_0 \) be any initial conforming tetrahedral mesh having a number of vertices, edges, faces, and tetrahedra respectively equal to \( N_0, E_0, F_0, \) and \( T_0 \); and consider the global use of the 3D-Skeleton Mesh Refinement algorithm producing a sequence of globally refined meshes \( \tau_1, \tau_2, \ldots, \tau_n \), and so on. Then the average number of tetrahedra sharing a vertex in the mesh is asymptotically equal to \( 2^4 \), the average number of faces sharing a vertex is asymptotically equal to \( 2^4 \), and the average number of edges per vertex tends to \( 14 \).

The proof is based on the resolution of the recurrence equations associated to the 8-tetrahedra longest-edge partition. Note that the global refinement of each mesh \( \tau_{n-1} \) reduces to the 8-tetrahedra longest-edge partition of all the tetrahedra of \( \tau_{n-1} \) which directly produces a conforming mesh \( \tau_n \). By Theorem 3.6, the number of vertices, edges, faces, and tetrahedra of the mesh \( \tau_n \), respectively equal to \( N_n, E_n, F_n, \) and \( T_n \), satisfy the following recurrence relations as a function of the values \( N_{n-1}, E_{n-1}, F_{n-1}, \) and \( T_{n-1} \) of the previous mesh:

\[
\begin{align*}
N_n &= N_{n-1} + E_{n-1} + T_{n-1} \\
E_n &= 2 \cdot E_{n-1} + 3 \cdot F_{n-1} + T_{n-1} \\
F_n &= 4 \cdot F_{n-1} + 8 \cdot T_{n-1} \\
T_n &= 8 \cdot T_{n-1}
\end{align*}
\]

(1)

where \( N_0, E_0, F_0, \) and \( T_0 \) are given from the initial mesh \( \tau_0 \).

Furthermore, since each tetrahedron has exactly four vertices, the average number of tetrahedra sharing a given vertex in the mesh \( \tau_n \) reduces to:

\[
\text{Av\#(tetrahedra per node)} = \frac{4 \cdot T_n}{N_n}
\]

And, in a similar way, the rest of the non-constant adjacency relations are:

\[
\begin{align*}
\text{Av\#(tet per edge)} &= \frac{6 \cdot T_n}{E_n} \\
\text{Av\#(faces per edge)} &= \frac{3 \cdot F_n}{E_n} \\
\text{Av\#(faces per node)} &= \frac{3 \cdot F_n}{N_n}
\end{align*}
\]

Once the recurrence relations (1) are solved, the asymptotic values are obtained taking limits when \( n \) tends to infinity. See reference [13] for details.

The following theorem summarizes geometrical and fractal properties of the 8T-LE refinement algorithm.

Theorem 5.2  Both for the 8T-LE partition and for the 3D-Skeleton Refinement algorithm the following mesh quality properties hold:

a) The 8T-LE partition of any tetrahedron \( t \) always partitions the largest planar angles of the two faces sharing the longest-edge of \( t \).

b) The 8T-LE partition never partitions a solid angle such that, each one of the three associated planar angles is non-obtuse and different from the largest angle of the corresponding triangular face.

c) Over each triangular obtuse face \( F \) of any tetrahedron \( t \), the iterative 8T-LE partition of \( t \) produces a finite number of different faces, such that each new face produced is better than the preceding one in the sense that the smallest angle and the largest angle of the new face are respectively greater than and less than those corresponding to the preceding face generated in the preceding iteration.

d) Property c) extends to each new obtuse face produced throughout the 8T-LE refinement process (self-corrective behavior).

The theorem proof is essentially based on the 2-dimensional properties of the 4T-LE refinement (Theorem 2.2).

Theorem 5.3  (Fractal behavior) For any conforming tetrahedral mesh \( \tau_0 \), after a finite number of local 3D-Skeleton refinements around any vertex \( P \), a finite number of tetrahedra sharing vertex \( P \) is obtained whose associated solid angles are never refined again as the refinement around \( P \) proceeds.

At this point some remarks are in order:

1. Part (b) of Theorem 5.2 implies that whenever a solid angle having non-obtuse planar angles (each one not opposite to the longest-edge of the corresponding triangular face) is obtained throughout the process, this solid angle remains untouched forever in the mesh. In other words, only new vertices along the edges of this solid angle are added as the refinement proceeds.
2. Parts (c) and (d) of Theorem 5.2 together state that the strong quality improvement properties of the 2-dimensional 4-Triangles partition hold over each triangular face of the 2D-Skeleton mesh, including the new faces.

3. Theorems 5.2 and 5.3 together do not certainly guarantee that the size of the molecules (set of tetrahedra sharing a given vertex of the mesh) do not increase as new vertices are added in the refinement process. However, empirical experimentation shows that a rather constant standard deviation around the average size of the molecules is obtained through the refinement steps, while the maximum size of them remains rather constant (equal to 64) in the last three levels.

6. EMPIRICAL RESULTS

In this section we report empirical evidence that supports the conjecture on the non-degeneracy property of both the 8T-LE partition and the mesh refinement algorithms based on this partition.

Here three numerical examples are presented. In every case the 8T-LE partition has been applied 7 times to an initial tetrahedron and its descendants, so the last level of division (T_7) contains 366, 145 vertices and 2,097, 152 tetrahedra. For each test tetrahedron a set of 3 tables have been produced: the first one contains the values Φ^{max}_{T_i} and third ones summarize statistical information for the coordinates of the vertices, while that the second of 3 tables have been produced: the first one contains value for the solid angles of tetrahedron T_i and Φ_i^R of Rivara and Levin [18], where Φ_i are the planar angles associated to vertex P_i used by Rivara and Levin [18], where Φ_T is the minimum Φ-value for the solid angles of tetrahedron T_i, and Φ_{min} and Φ_{max} are respectively equal to the minimum and maximum Φ-values attained for the mesh at level n. Note that 0 ≤ Φ_T ≤ 45° and Φ = 0 implies a totally degenerate tetrahedron. For a discussion on tetrahedron shape measures see [5].

It should be pointed out here that the improvement behavior of any tetrahedron T_i will be in general studied relative to the quality of the tetrahedra of the first volume partition of T_i. This is due to the fact that the quality measures Φ_{T_1}, associated to the tetrahedron T_1, (i = 1, . . . , 8), of the first partition of a tetrahedron T_i in general describe better the local feature sizes of T_i than the Φ_T measure itself. Consider for instance a cap (very flat) tetrahedron having four quality acceptable faces which clearly do not reflect well the tetrahedron quality; the first 8-tetrahedra partition of T_i in exchange, introduces at least a bad quality face that describe well the thickness of T_i.

Table 1: Right-tetrahedron vertices

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>4.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>4.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

In the first test problem the initial tetrahedron is a right-tetrahedron, with a vertex in the origin of the coordinate system, and three vertices over the axes of the coordinate system to equal distance from the origin. The evolution of the shape for the tetrahedra as the partition proceeds is shown in Table 5. Note that, as expected, the minimum solid angle remains constant since the second global partition, while the percentage of volume covered by bad-shaped elements decreases monotonicaly from the third global partition. Figures 8, 9, and 10 show the evolution of the average number of tetrahedra per vertex as the global refinement (partition) proceeds. Note that the distribution seems to tend to a bimodal distribution, with concentrations between 15 and 20, and between 45 and 50, with average around 24, which is the asymptotic average number for this partition. Also the maximum number of tetrahedra per vertex is included in the figures.

The second example considers a needle tetrahedron. Table 6 shows the evolution of the minimum and maximum angles, and the % of volume covered by bad-shaped elements, while Figure 9 shows the evolution of tetrahedra per node for this needle tetrahedron where the distribution also approaches the mean value 24. Note that in this case, since the faces of T reflect well the local feature sizes of this needle tetrahedron, the worst solid angle remains constant throughout the process.

Table 2: Needle tetrahedron vertices

<table>
<thead>
<tr>
<th></th>
<th>-0.5</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>7.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The third example corresponds to a flat tetrahedron. Table 7 shows for this example the evolution of the shape of the elements and meshes obtained at global partitioning. Note that the minimum solid angle remains constant from the second global refinement, while the percentage of volume covered by bad-shaped tetrahedra improves when the partitioning proceeds.
Finally, Table 4 shows the evolution of the average number of tetrahedra per vertex in the first 10 steps of global iterative application of the 8T-LE partition to any initial tetrahedron.

It should be pointed out here that the 3D partition seems to have similar behavior to the 2D case in the sense that for needle tetrahedra (equivalent to one small-angled triangle), a clear monotonic improvement behavior holds, while that for quality tetrahedra and cap tetrahedra a limited decreasing of the tetrahedron quality can be observed in the first partition. Note that in 2-dimensions, the 4-triangles partition of an equilateral triangle produces some 30 degrees triangles, and this is the only case where the bound in part (2) of Theorem 2.1 is attained. For the cap tetrahedron in exchange, the first 3D partition introduces an acute face that captures the thickness of this tetrahedron (a local feature size not described by its four faces). Note that this is not the case of a needle tetrahedron where its faces fully describe its local feature sizes.

Table 4: Statistical Measures

<table>
<thead>
<tr>
<th>Level</th>
<th>Num. Tets.</th>
<th>Avg(#\text{tets per node})</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>512</td>
<td>12.41</td>
</tr>
<tr>
<td>5</td>
<td>4,096</td>
<td>16.90</td>
</tr>
<tr>
<td>6</td>
<td>32,768</td>
<td>20.03</td>
</tr>
<tr>
<td>7</td>
<td>262,144</td>
<td>21.88</td>
</tr>
<tr>
<td>8</td>
<td>2,097,152</td>
<td>22.91</td>
</tr>
<tr>
<td>9</td>
<td>16,777,216</td>
<td>23.45</td>
</tr>
<tr>
<td>10</td>
<td>134,217,728</td>
<td>23.72</td>
</tr>
<tr>
<td>11</td>
<td>1,073,741,824</td>
<td>23.86</td>
</tr>
<tr>
<td>12</td>
<td>8,589,934,592</td>
<td>23.93</td>
</tr>
<tr>
<td>13</td>
<td>68,719,476,736</td>
<td>23.96</td>
</tr>
</tbody>
</table>

7. CONCLUDING REMARKS

Although in the last 15 years the longest-edge refinement algorithms have become well-known and useful techniques which guarantee the construction of quality refined meshes in 2-dimensions, equivalent non-degeneracy properties had not been proved yet in 3-dimensions. The question was essentially centered before either on finding a lower bound on the minimum solid angle or on looking for results on the number of similarly distinct tetrahedra produced. This last approach is a rather difficult path to follow because of the combinatorial issues involved in 3-dimensions.

In this paper we see that stronger improvement and fractal properties proved for 2-dimensional longest-edge based algorithms [17], also hold over the triangular faces of the volume meshes. The use of these properties seems to be a fruitful path for obtaining mathematical results on the 3-dimensional longest-edge algorithms.

This paper include theoretical and empirical results on this direction: We have discussed a longest-edge based volume algorithm which induces the 4-Triangles partition of the faces of the tetrahedra. The improvement and fractal properties of the 4-Triangles longest-edge partition have been in turn used to prove statistical and fractal properties over the 8-Tetrahedra longest-edge refinement algorithm: (1) the asymptotic average number of tetrahedra surrounding each vertex is equal to 24; (2) the number of tetrahedra surrounding each fixed vertex remains constant, after a few local iterative refinement around such a vertex; and (3) the algorithm improves each triangular face produced as the refinement proceeds.

Empirical study carried out here not only supports these results but also shows that, consistently throughout the refinement levels the distribution of quality tetrahedra improve and the volume percentage covered by better tetrahedra increase as the refinement proceeds.
### Table 5: Shape evolution for a right-tetrahedron

<table>
<thead>
<tr>
<th>Level</th>
<th>Num. of Nodes</th>
<th>Num. of Elems.</th>
<th>$\Phi_{\text{min}}$</th>
<th>Planar angles associated to $\Phi_{\text{min}}$</th>
<th>$\Phi_{\text{max}}$</th>
<th>% bad elems. ($\Phi_T &lt; 10$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>30.00</td>
<td>45.00 $\neq$ 60.00 $\neq$ 45.00</td>
<td>90</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>9.59</td>
<td>19.47 $\neq$ 35.26 $\neq$ 30.00</td>
<td>90</td>
<td>25.00</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>64</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>25.00</td>
</tr>
<tr>
<td>4</td>
<td>165</td>
<td>512</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>20.31</td>
</tr>
<tr>
<td>5</td>
<td>969</td>
<td>4096</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>15.62</td>
</tr>
<tr>
<td>6</td>
<td>6545</td>
<td>32768</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>11.82</td>
</tr>
<tr>
<td>7</td>
<td>47905</td>
<td>262144</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>8.89</td>
</tr>
<tr>
<td>8</td>
<td>366145</td>
<td>2097152</td>
<td>9.59</td>
<td>30.00 $\neq$ 35.26 $\neq$ 19.47</td>
<td>90</td>
<td>6.67</td>
</tr>
</tbody>
</table>

![Table 5 Diagram](image.png)

**Figure 8:** Distribution of vertices versus number of tetrahedra per vertex. Right-Shaped Tetrahedron.

### Table 6: Shape evolution for a needle tetrahedron.

<table>
<thead>
<tr>
<th>Level</th>
<th>Num. of Nodes</th>
<th>Num. of Elems.</th>
<th>$\Phi_{\text{min}}$</th>
<th>Planar angles associated to $\Phi_{\text{min}}$</th>
<th>$\Phi_{\text{max}}$</th>
<th>% bad elems. ($\Phi_T &lt; 0.24$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0.23</td>
<td>8.00 $\neq$ 4.36 $\neq$ 4.28</td>
<td>43.58</td>
<td>100.00</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>0.23</td>
<td>8.00 $\neq$ 4.26 $\neq$ 4.28</td>
<td>67.84</td>
<td>75.00</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>64</td>
<td>0.22</td>
<td>4.26 $\neq$ 4.36 $\neq$ 8.00</td>
<td>68.14</td>
<td>68.75</td>
</tr>
<tr>
<td>4</td>
<td>165</td>
<td>512</td>
<td>0.22</td>
<td>4.28 $\neq$ 4.36 $\neq$ 8.01</td>
<td>68.14</td>
<td>67.19</td>
</tr>
<tr>
<td>5</td>
<td>969</td>
<td>4096</td>
<td>0.22</td>
<td>4.23 $\neq$ 4.35 $\neq$ 7.96</td>
<td>68.14</td>
<td>66.80</td>
</tr>
<tr>
<td>6</td>
<td>6545</td>
<td>32768</td>
<td>0.22</td>
<td>4.23 $\neq$ 4.35 $\neq$ 7.96</td>
<td>68.14</td>
<td>66.71</td>
</tr>
<tr>
<td>7</td>
<td>47905</td>
<td>262144</td>
<td>0.22</td>
<td>7.96 $\neq$ 4.23 $\neq$ 4.35</td>
<td>68.14</td>
<td>66.63</td>
</tr>
<tr>
<td>8</td>
<td>366145</td>
<td>2097152</td>
<td>0.22</td>
<td>7.96 $\neq$ 4.23 $\neq$ 4.35</td>
<td>68.14</td>
<td>66.63</td>
</tr>
</tbody>
</table>
Figure 9: Distribution of vertices versus number of tetrahedra per vertex. Needle Tetrahedron.

Table 7: Shape evolution for a flat tetrahedron.

<table>
<thead>
<tr>
<th>Level</th>
<th>Num. of Nodes</th>
<th>Num. of Elems.</th>
<th>$\Phi_{\text{min}}$</th>
<th>Planar angles associated to $\Phi_{\text{min}}$</th>
<th>$\Phi_{\text{max}}$</th>
<th>$%$ bad elems. ($\Phi_T &lt; 10$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>6.31</td>
<td>24.90, 24.74, 46.78</td>
<td>24.94</td>
<td>100.00</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>8</td>
<td>3.68</td>
<td>40.11, 6.08, 38.42</td>
<td>33.30</td>
<td>62.50</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>64</td>
<td>3.12</td>
<td>46.68, 27.74, 19.70</td>
<td>75.29</td>
<td>45.31</td>
</tr>
<tr>
<td>4</td>
<td>165</td>
<td>512</td>
<td>3.12</td>
<td>46.68, 27.74, 19.70</td>
<td>75.29</td>
<td>37.50</td>
</tr>
<tr>
<td>5</td>
<td>969</td>
<td>4096</td>
<td>3.12</td>
<td>46.68, 27.74, 19.70</td>
<td>75.29</td>
<td>30.03</td>
</tr>
<tr>
<td>6</td>
<td>6545</td>
<td>32768</td>
<td>3.12</td>
<td>46.68, 27.74, 19.70</td>
<td>75.29</td>
<td>22.97</td>
</tr>
<tr>
<td>7</td>
<td>47905</td>
<td>262144</td>
<td>3.12</td>
<td>46.68, 27.74, 19.70</td>
<td>75.29</td>
<td>17.13</td>
</tr>
<tr>
<td>8</td>
<td>366145</td>
<td>2097152</td>
<td>3.12</td>
<td>19.70, 46.78, 27.74</td>
<td>74.85</td>
<td>12.57</td>
</tr>
</tbody>
</table>

Figure 10: Distribution of vertices versus number of tetrahedra per vertex. Flat Tetrahedron.
References


PROPAGATION PATH PROPERTIES IN ITERATIVE LONGEST-EDGE REFINEMENT

J.P. Suárez\textsuperscript{1} \hspace{1cm} A. Plaza\textsuperscript{1} \hspace{1cm} G.F. Carey\textsuperscript{2}

\textsuperscript{1}University of Las Palmas de Gran Canaria, Canary Islands, Spain, jsuarez@dcegi.ulpgc.es, aplaza@dma.ulpgc.es
\textsuperscript{2}University of Texas at Austin, Austin, Texas, U.S.A. carey@cfdlab.ae.utexas.edu

ABSTRACT

In this work we investigate the refinement propagation process in longest-edge based local refinement algorithms for unstructured meshes of triangles. The conformity neighborhood of a triangle, the set of additional triangles that is needed to be refined to ensure mesh conformity is introduced to define the propagation path. We prove that asymptotically the propagation path extends on average to a few neighbor adjacent triangles. We also include numerical evidence which is in complete agreement with the theoretical study reported.

Keywords: mesh refinement, longest edge bisection, propagation path

1. INTRODUCTION

Mesh Generation plays a central role in the Finite Element Method [1, 2, 3], and is a basic tool in many other fields such as Computational Geometry and Computer Graphics. A related problem that is also of considerable interest is refinement of a mesh. The refinement problem can be described as any technique involving the insertion of additional vertices in order to produce meshes with desired features: well shaped triangles, mesh conformity and smoothness. The presence of thin triangles can lead to undesirable behavior affecting numerical stability and accuracy. Mesh conformity refers to the requirement that the intersection of adjacent triangles is either a common vertex or an entire side. Mesh smoothness implies that the transition between small and large elements should be gradual.

Certain longest-edge refinement algorithms [4, 5, 6] guarantee the construction of non-degenerate and smooth unstructured triangulations. In these schemes the longest edges are progressively bisected and hence all angles in subsequent refined triangulations are greater than or equal to half the smallest angle in the initial triangulation [7]. However, the extent of secondary refinements induced in neighboring elements by the initiating element edge bisection is not known [6, 8]. One can construct pathological cases where refinement of a single element propagates through the entire mesh (Figure 1). However experience indicates that this is an exception and that in practice the refinement propagates through only a few neighbors on average. Our goal here is to address this question. We provide both theoretical results and empirical evidence showing that successive application of refinement to an arbitrary unstructured triangular mesh produces meshes in which the average propagation path is reduced in each refinement stage, and asymptotically approaches the constant 5.

2. PRELIMINARIES. THE REFINEMENT AND THE PROPAGATION PROBLEM

The refinement of triangular meshes involves two main tasks. The first is the partition of the target

\footnote{Throughout this work, neighbor triangles are triangles sharing an edge}
triangles and the second is the propagation to successive neighbor triangles to preserve conformity. Several approaches for partitioning triangles have been studied. The simplest is Bisection into two subtriangles by connecting the midpoint of one of the edges to the opposite vertex. If the Longest Edge (LE) is chosen for the bisection, then this is called Longest Edge Bisection, see Figure 2 (a). The Four Triangles Longest Edge Partition, (4T-LE) bisects a triangle into four subtriangles where the original triangle is first subdivided by its longest edge as before and then the two resulting triangles are bisected by joining the new midpoint of the longest edge to the midpoints of the remaining two edges of the original triangle, as in Figure 2 (d), [5].

In order to ensure the conformity of the arising mesh, the refinement must be extended to additional triangles. This is made for the 4T-LE partition by the use of partial division patterns given in Figure 2 (a)-(c).

**Definition 1** (Longest Edge Neighbor triangle) The longest edge neighbor of a triangle \( t \) is the neighbor triangle \( t' \) which shares with \( t \) the longest edge of \( t \).

In the case of an isosceles or equilateral triangle, we may assume a ‘roundoff level’ perturbation to yield a single longest edge. This can be random and hence uniqueness is not implied. However, in the cases in which one of the longest edges has been already identified for bisection in a neighbor triangle, this edge is chosen as the longest edge to get the refinement as local as possible.

**Definition 2** (LE Propagation Path [5, 9]) The Longest Edge Propagation Path (LEPP) of a triangle \( t_0 \) is the ordered finite list of all adjacent triangles \( \text{LEPP}(t_0) = \{ t_0, t_1, \ldots, t_n \} \) such that \( t_i \) is the longest edge neighbor triangle of \( t_{i-1} \).

Throughout this work \( \tau \) denotes a 2D conforming triangulation. If longest edge bisection is used to refine a given triangle \( t \in \tau \), then the LEPP(\( t \)) provides the list of triangles to be refined, (see Figure 3). Note that if the 4T-LE partition is used to refine a given triangle \( t \), then the LEPP’s of the neighbor triangles of \( t \) in the mesh \( \tau^* = \tau - t \) provide the lists of triangles to be refined (see Figure 4 and Table 1). As a consequence, the LEPP’s provide the main adjacency lists used by the algorithms.

**Definition 3** (Boundary and Interior triangle) Let \( \tau \) be a two dimensional triangulation for a bounded domain \( \Omega \). A triangle \( t \in \tau \) is said to be a boundary triangle if \( t \) has an edge coincident with the boundary \( \partial \Omega \) of \( \Omega \). Otherwise, \( t \) is an interior triangle of \( \tau \).
Definition 4 (Pair of Terminal triangles) Two neighbor triangles $(t, t')$ will be called a ‘pair’ of terminal triangles if they share a common longest edge. If a triangle $t$ does not belong to a pair of terminal triangles, $t$ is said to be a ‘single’ triangle.

For any triangle $t_0$, if $\text{LEPP}(t_0) = \{t_0, \ldots, t_{n-1}, t_n\}$ then for triangle $t_n$ either: (i) $t_n$ has its longest edge coincident with the boundary or (ii) $t_{n-1}$ and $t_n$ are a pair of terminal triangles that share a common longest edge, [9].

If all the triangles in a mesh are pairs of terminal triangles, then all the LEPP lists are comprised only of two triangles.

Definition 5 (LEPP-balanced mesh) Triangulation $\tau$ is said to be LEPP-balanced if it is comprised of pairs of terminal triangles. Otherwise, it is said to be a non LEPP-balanced mesh.

Remark: In [10] the terminology ‘balanced’ is applied to angles and areas. This is relevant to triangle shape but not directly related to our LEPP study here.

Definition 6 (LEPP-balancing degree) Let $\tau$ contain $N$ triangles of which $T$ triangles are in pairs of terminal triangles. Then, the LEPP-balancing degree of $\tau$, noted as $B(\tau)$, is defined as follows:

$$B(\tau) = \frac{T}{N}$$  \hspace{1cm} (1)

Note that $0 \leq B(\tau) \leq 1$ and in the case $B(\tau) = 1$, the mesh is LEPP-balanced.

Remark: If $\tau$ is such that the LEPP-balancing degree is 0, then the conformity process when refining any triangle $t_0 \in \tau$ extends to the boundary of $\tau$.

Figure 5 shows a LEPP-balanced mesh in (a) and a non balanced mesh in (b). Here and in subsequent figures we represent the longest-edges with a dashed line.

A simple example of a LEPP-balanced mesh is any mesh comprised entirely of pairs of right triangles sharing the longest-edges. In such a mesh, if one applies uniform $4T$-LE refinement, then all triangles are pairs of terminal triangles and they are similar to the original right triangles.

3. PROPAGATION PROPERTIES OF RECURSIVE 4T-LE REFINEMENT

We are particularly interested in the average and maximum lengths of the propagation paths generated by longest-edge refinement since they are important in assessing algorithm efficiency.

First we introduce the Conformity Neighborhood associated with the application of $4T$-LE local refinement. This concept will be useful in the study of the propagation properties.

Definition 7 (Conformity Neighborhood) When refining a triangle $t \in \tau$, the Conformity Neighborhood $V_c(t)$ of $t$, is the set of triangles in $\tau' = \tau - t$ that need to be refined due to the conformity process for $t$.

Definition 8 (M1) When refining a triangle $t \in \tau$, $M_1(t)$ is said to be the size of $V_c(t)$: $M_1(t) = |V_c(t)|$.

Note that $M_1(t)$ measures the extent of the propagation refinement zone for triangle $t$, in number of triangles.

Figure 5: (a) LEPP-Balanced mesh, (b) Non balanced LEPP mesh.
Proposition 1 For each $t \in \tau$, $M_1(t)$ is the sum of the lengths of the LEPP’s of the neighbors of $t$ in the mesh $\tau^* = \tau - t$. 

Figure 1 shows that it always is possible to construct meshes in which $M_1(t)$ is $O(N)$, where $N$ is the number of elements. Here, the average of meshes in which $M_1(t)$ is $\mu(M_1) = \sum_{k=0}^{N-1} M_1(t) / N = \frac{\sum_{k=0}^{N-1} k}{N} = \frac{N-1}{2}$. On the other hand, if $B(\tau) = 1$ as in Figure 5 (a), then $M_1(t) \leq 5 \forall t \in \tau$. 

Definition 9 (M2) For each $t \in \tau$, $M_2(t)$ is the maximum length of the LEPP’s of the neighbor triangles of $t$ in the mesh $\tau^* = \tau - t$: $M_2 = \max\{V_r(t) \cap LEPP(t_a) | t_a \text{ neighbor to } t\}$. 

Since the conformity process extends at most by the three edges of $t$ the propagation defines at most three lists of ordered triangles. $M_2(t)$ is the maximum number of triangles of the three resulting lists. For example, in Figure 4, $M_2(t) = 2$ because the maximum number of triangles among $\{t_0, t_1\}, \{t_2, t_3\}, \{t_4\}$ is 2, see Table 1. 

Proposition 2 Let $t$ be LEPP-balanced. Then, for each interior triangle $t \in \tau$, $M_1(t) = 5$ and $M_2(t) = 2$. 

Proof: 
Let $t$ be an interior triangle of $\tau$. Since $t$ is a LEPP-balanced mesh, $t$ is adjacent to another triangle $t_1$ by their common longest edge. Let $t_2$ and $t_3$ be the two other adjacent triangles to $t$. Again, since $t$ is a LEPP-balanced mesh, $t_2$ and $t_3$ are adjacent to other triangles $t_2'$ and $t_3'$ by their respective common longest edges, and $t_2' \neq t \neq t_3'$. Considering the mesh $\tau^* = \tau - t$ we have that $LEPP(t_1) = \{t_1\}$, $LEPP(t_2) = \{t_2, t_2'\}$ and $LEPP(t_3) = \{t_3, t_3'\}$. Hence $V_r(t) = \{t_1, t_2, t_3, t_2', t_3'\}$ so $M_1(t) = 5$ and $M_2(t) = 2$. 

Our next goal is to prove that the uniform application of the 4T-LE partition will produce a sequence of meshes with increasing LEPP-balancing degree approaching 1. We shall also prove that the mean of $M_1$ and the mean of $M_2$ tend to 5 and 2 respectively, when the number of refinements applied tends to infinity. 

Proposition 3 (4) (a) The first application of the 4T-LE partition to a given triangle $t_0$ introduces two new triangles that are similar to the original triangle $t_0$. Moreover, these two triangles have their longest edges coincident with the longest edge of the original triangle. The remaining two new triangles $t_1$, are similar to each other but not necessarily to the original triangle $t_0$. Triangles $t_1$ may be a terminal pair or not. (b) The iterative application of the 4T-LE partition to a given triangle $t_0$ introduces at most one new distinct (up to similarity) triangle in each iteration. 

Proof: 
The proof follows from the angle properties of similar subtriangles obtained by 4T-LE quadrisection as seen in Figures 6 and 7 for the acute and obtuse triangles respectively. 

Proposition 4 If the 4T-LE partition to an initial triangle $t_0$ introduces a pair of terminal triangles $t_1$, then the iterative application of the 4T-LE partition introduces pairs of terminal triangles excepting the triangles located at the longest edge of $t_0$. Moreover, in this case only two classes of similar triangles are generated, corresponding to $t_0$ and $t_1$ respectively (see Figure 6). 

Proof: 
The hypothesis of Proposition 4 is depicted in Figure 6 (b). The proof follows trivially from the angle properties of parallel lines in the nested triangles. 

To demonstrate that recursive uniform 4T-LE refinement introduces meshes with relatively more pairs of terminal triangles for any arbitrary triangular mesh we consider right, acute and obtuse triangles respectively. 

We begin in the next Proposition with the right and acute triangle cases: 

Proposition 5 (Right and acute triangle cases) The application of the 4T-LE partition to an initial right or acute triangle $t_0$ produces two new single triangles similar to the original one (located at the longest edge of $t_0$) and a pair of terminal triangles $t_1$. These triangles $t_1$ are also similar to the original one $t_0$ in the case of right triangle $t_0$, and they are similar to each other but non-similar to the initial one in the case of acute triangle $t_0$. (See Figure 6). 

The obtuse triangle case offers a different situation: 

Proposition 6 (Obtuse triangle case) The application of the 4T-LE partition to an initial obtuse triangle $t_0$, produces two new single subtriangles similar to the original one (located at the longest edge of $t_0$) and a pair of subtriangles $t_1$. These subtriangles $t_1$ either
1. are a pair of similar terminal triangles, as in Figure 6 (b) ($t_0$ is said to be a Type 1 obtuse triangle), or
2. a pair of similar single triangles, as in Figure 7 (b) ($t_0$ is said to be a Type 2 obtuse triangle).

It should be noted that the 4T-LE partition always produces two new single triangles similar to the original one (located at the longest edge of $t_0$) and excluding for Type 2 obtuse triangles, a pair of terminal triangles (similar or non-similar to the original one). Moreover, in this scenario, the single triangles generated by the iterative 4T-LE partition are those located at the longest edge of the initial triangle, Proposition 4 (see Figure 6).

The following Proposition states the recursive improvement property of the 4T-LE partition for obtuse triangles [4]:

Proposition 7 If the 4T-LE partition of an obtuse triangle $t_0$ introduces a pair of similar single triangles $t_1$ of largest angle $\gamma_1$, then
1. $\gamma_1 = \sigma$
2. $\gamma_1 = \gamma_0 - \epsilon \leq \gamma_0 - \alpha_0$
hold for the new angles of the new triangle $t_1$, see Figure 7.

It is worth noting, in relation to Proposition 7 above, that after a finite number of applications of the 4T-LE partition to triangle $t_0$ and its successors a non-obtuse triangle is obtained. This is a straightforward consequence of statement 2 in Proposition 7. Furthermore, after a first non-obtuse triangle is obtained then the successive application of the 4T-LE partition does not generate new non-similar triangles.

In view of the previous properties, we have:

Proposition 8 Let $\Gamma = \{\tau_0, \tau_1, \ldots, \tau_n\}$ be a sequence of nested meshes obtained by repeated application of 4T-LE partition to the previous mesh. Then, the LEPP-balancing degree of the meshes tends to 1 as $n \to \infty$.

Proof:
It suffices to prove the result for the case in which the initial mesh $\tau_0$ only contains a single triangle $t_0$. Then, the number of generated triangles associated with the 4T-LE partition at stage $n$ of refinement is:

$$N_n = 4^n$$

First, we prove the proposition for initial right, acute, and the Type 1 obtuse triangles. In this situation, the number of triangles in pairs of terminal triangles
$T_n$ generated at stage $n$ of uniform 4T-LE partition satisfies (see Proposition 4 and Figure 6):

$$T_n = 4T_{n-1} + 2(N_{n-1} - T_{n-1}) = 2(T_{n-1} + N_{n-1}) \quad (3)$$

with $N_0 = 1$ and $T_0 = 0$.

Solving the recurrence relations 2, 3 we get:

$$T_n = 4^n - 2^n \quad (4)$$

Therefore,

$$\lim_{n \to \infty} B(\tau_n) = \lim_{n \to \infty} \frac{T_n}{N_n} = 1$$

To complete the proof, we now consider the case of an initial Type 2 obtuse triangle $t_0$. Table 2 presents the number of distinct types of triangles generated by the 4T-LE iterative refinement of $t_0$. We denote by $t_j^*$ the number of triangles of similarity class $t_j$, $j = 0, 1, 2, \cdots, k$ at stage $n$ of refinement. For example, after the second refinement 4 triangles are similar to $t_0$, 8 triangles similar to $t_1$ and 4 new triangles similar to $t_2$.

From Proposition 6 (2) and Figure 7 we derive Table 2, in which the following relation holds:

$$t_j^* = 2(t_{j-1}^* + t_{j-2}^*), \quad j = 1, 2, 3, \cdots, k \quad (5)$$

The solution to Equation (5) with initial condition $t_0^* = 1$ can be easily expressed in terms of binomial coefficients as follows:

$$t_j^* = 2^n \binom{n}{j} \quad (6)$$

On the other hand, from Proposition 7, the iterative 4T-LE partition of any obtuse triangle $t_0$ produces a finite number of distinct (up to similarity) triangles, $t_j^*$, $0 < j \leq k$. After $k$ refinement stages there will no longer be distinct new generated triangles different from those already generated, (see proof of Proposition 6). Therefore, the number of triangles in pairs of terminal triangles $T_n$ after the $k$ refinement stage with $n > k$ satisfy:

$$T_n \geq 2^n \sum_{m=k}^{n} \binom{n}{m}$$

It follows that:

$$1 \geq B(\tau_n) \geq \frac{2^n \sum_{m=k}^{n} \binom{n}{m}}{2^n \sum_{m=0}^{n} \binom{n}{m}} = \frac{\sum_{m=k}^{n} \binom{n}{m}}{2^n}$$

Table 2: Triangle evolution in the 4T-LE partition.

<table>
<thead>
<tr>
<th>Ref.</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>k</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>...</td>
<td>$t_0^*$</td>
<td>...</td>
<td>$t_0^*$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>2</td>
<td>8</td>
<td>24</td>
<td>64</td>
<td>$t_1^*$</td>
<td>...</td>
<td>$t_1^*$</td>
<td>...</td>
<td>$t_1^*$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>4</td>
<td>24</td>
<td>96</td>
<td>256</td>
<td>$t_2^*$</td>
<td>...</td>
<td>$t_2^*$</td>
<td>...</td>
<td>$t_2^*$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>8</td>
<td>64</td>
<td>$t_3^*$</td>
<td>...</td>
<td>$t_3^*$</td>
<td>...</td>
<td>$t_3^*$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>16</td>
<td>$t_4^*$</td>
<td>...</td>
<td>$t_4^*$</td>
<td>...</td>
<td>$t_4^*$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| ... | ... | ... | ... | ... | ... | ... | ...
| $t_k$ | ... | ... | ... | ... | ... | ... | $t_k^*$ | ...

Taking limits:

$$1 \geq \lim_{n \to \infty} B(\tau_n) \geq \lim_{n \to \infty} \frac{2^n \sum_{m=k}^{n} \binom{n}{m}}{2^n \sum_{m=0}^{n} \binom{n}{m}}$$

Since

$$\sum_{m=k}^{n} \binom{n}{m} = 2^n - \sum_{m=0}^{k-1} \binom{n}{m} \geq 2^n - \binom{n}{k-1}$$

we have

$$1 \geq \lim_{n \to \infty} B(\tau_n) \geq \lim_{n \to \infty} \frac{2^n - \binom{n}{k-1}}{2^n} = 1$$

So, $\lim_{n \to \infty} B(\tau_n) = 1$. \hfill \Box

**Proposition 9** For iterative application of 4T-LE uniform refinement to an initial triangular mesh $\tau_0$, the means of M1 and of M2 tend to 5 and 2 respectively, as the number of refinements tends to infinity.

**Proof:**

If the initial mesh is LEPP-balanced the result is trivial. Let us suppose that the initial mesh contains non-terminal triangle.

In any subsequent mesh we have pairs of terminal triangles and non terminal triangles. First, we prove the proposition for any right, acute, or Type 1 obtuse non terminal triangles arranged in such a way that M1 and M2 are the largest. That is, all the non-terminal triangles constitute a unique LEPP. Figure 8 (a) reproduces a possible situation within a mesh. After a few refinement steps it is observed that new non terminal triangles are located at the longest edges of the initial triangles. We represent in bold the longest edges of the initial polyline, see Figure 8 (d). Among them, those having...
Similarly for $M_2$: the number of total triangles is $N$, number of non-rectangular triangles is $X$. After refinement of less than or equal to $n$ steps, the number of non-rectangular triangles is $X_n = 2^n X_0$, and the number of total triangles is $N_n = 4^n N_0$. Hence, upper and lower bounds for the average of $M_1$ are as follows:

$$\frac{6X_n + 5(N_n - X_n)}{N_n} \leq M_1 \leq \frac{7 \cdot 2X_n + 5(N_n - 2X_n)}{N_n}$$

Similarly for $M_2$:

$$\frac{2X_n + 2(N_n - X_n)}{N_n} \leq M_2 \leq \frac{3 \cdot 2X_n + 2(N_n - 2X_n)}{N_n}$$

Taking limits we find that the means of $M_1$ and of $M_2$ tend to 5 and 2 respectively, as the number of refinements $n$ tends to infinity.

To complete the proof, we should also consider the case of Type 2 obtuse triangles. As pointed out after Proposition 7, in repeated 4T-LE refinement, largest angles of Type 2 obtuse triangles clearly decrease, and after a finite number $k$ of 4T-LE partitions the new generated triangles will be no longer obtuse. Hence, the first part of the proof for right or acute triangle cases then applies.

4. NUMERICAL EXPERIMENTS

In this section we present numerical results showing that the practical behavior of the 4T-LE partition is in concordance with the reported theory in this work, mainly Propositions 8 and 9.

We next consider a Delaunay mesh in a rectangle (Figure 9 (a)) and an irregular mesh in a pentagon (Figure 12 (a)) with five stages of uniform refinement. It should be noted that the triangles in the Delaunay mesh are almost regular in terms of the angles moreover, the mean of the minimum angles and of the maximum angles are 48.91 and 72.91 degrees respectively. The initial value of $B(\tau_0)$ is 0.4833. On the other hand, for the irregular mesh in the pentagon the mean of the minimum angles and of the maximum angles are 9.18 and 120.41 degrees respectively, and $B(\tau_0) = 0$.

The refined meshes for the initial Delaunay mesh are presented in Figures 9 (b)-(d). The light-shading in Figures 9 and 12 illustrate the triangles in terminal pairs. In Table 3 it can be noted that the number of triangles in terminal pairs increase as the refinement stage grows, and as result, so does the LEPP-balancing degree. Table 4 reports the means and standard deviations of $M_1$ and $M_2$ and respective histograms are graphed in Figure 11. It is observed that both means tend to 5 and 2 respectively, as the refinement continues. The asymptotic behavior is graphed in Figure 10.

Similarly, the refined meshes for the ‘pentagonal’ domain are shown in Figure 12 and the asymptotic behavior for the means $\mu(M_1), \mu(M_2)$ graphed in Figure 13. The evolution of the LEPP-balancing degree is summarized in Table 5 and a comparison graphed in Figure 15. Note that in both meshes the LEPP-balancing degree tends to 1 when the number of refinements increases, even in the Pentagonal mesh, which exhibits an initial LEPP-balancing degree of $B(\tau_0) = 0$ (see Figure 15). Table 6 reports the means and standard deviations of $M_1$ and $M_2$ and respective histograms are graphed in Figure 14.

These results are also applicable to local refinement. In order to empirically demonstrate this we consider application of 4T-LE local refinement on a domain corresponding to the Gran Canaria Island (Figure 16). The initial mesh is a Delaunay mesh and one refinement step is applied on respective disjoint subregions $S_1$, $S_2$ and $S_3$ with $S = S_1 \cup S_2 \cup S_3$, for innermost region $S_1$, intermediate region $S_2$ and outermost region $S_3$. Table 7 and Figure 17 confirm similar behavior to that observed for uniform refinement with $\mu(M_1)$ and $\mu(M_2)$ approaching 5 and 2 respectively and the LEPP-balancing degree approaching 1. Figure 18 graphs $M_1$ and $M_2$ histograms for the initial mesh and refinement steps 3 and 6.
Figure 9: Delaunay mesh. Uniform 4T-LE refinement.

Table 3: Statistics for the refinement of the Delaunay mesh (Figure 9). R=Refinement step, T=Triangles in Terminal Pairs, N=Triangles, B=LEPP-Balancing Degree.

<table>
<thead>
<tr>
<th>R</th>
<th>T</th>
<th>N</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>58</td>
<td>120</td>
<td>0.48333</td>
</tr>
<tr>
<td>1</td>
<td>356</td>
<td>480</td>
<td>0.74166</td>
</tr>
<tr>
<td>2</td>
<td>1672</td>
<td>1920</td>
<td>0.87083</td>
</tr>
<tr>
<td>3</td>
<td>7184</td>
<td>7680</td>
<td>0.93541</td>
</tr>
<tr>
<td>4</td>
<td>29728</td>
<td>30720</td>
<td>0.96770</td>
</tr>
<tr>
<td>5</td>
<td>121477</td>
<td>122880</td>
<td>0.98821</td>
</tr>
</tbody>
</table>

Table 4: M1 and M2 statistics for the refinement of the Delaunay mesh (Figure 9). Average (μ) and Standard Deviation (σ).

<table>
<thead>
<tr>
<th>R</th>
<th>N</th>
<th>μ(M1)</th>
<th>σ(M1)</th>
<th>μ(M2)</th>
<th>σ(M2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>120</td>
<td>5.233</td>
<td>1.873</td>
<td>2.625</td>
<td>0.888</td>
</tr>
<tr>
<td>1</td>
<td>480</td>
<td>5.112</td>
<td>1.35</td>
<td>2.381</td>
<td>0.582</td>
</tr>
<tr>
<td>2</td>
<td>1920</td>
<td>5.057</td>
<td>0.772</td>
<td>2.195</td>
<td>0.421</td>
</tr>
<tr>
<td>3</td>
<td>7680</td>
<td>5.028</td>
<td>0.535</td>
<td>2.096</td>
<td>0.303</td>
</tr>
<tr>
<td>4</td>
<td>30720</td>
<td>5.014</td>
<td>0.374</td>
<td>2.048</td>
<td>0.216</td>
</tr>
<tr>
<td>5</td>
<td>122880</td>
<td>5.007</td>
<td>0.275</td>
<td>2.028</td>
<td>0.040</td>
</tr>
</tbody>
</table>
Figure 10: Evolution of $\mu(M1)$ and $\mu(M2)$ for the refinement of the Delaunay mesh (Figure 9).

Figure 11: $M1$ and $M2$ histograms for the refinement of the Delaunay mesh (Figure 9).

Figure 12: Pentagonal mesh. Uniform 4T-LE refinement.
Table 5: Statistics for the refinement of the Pentagonal mesh (Figure 12). R=Refinement step, T=Triangles in Terminal Pairs, N=Triangles, B=LEPP-Balancing Degree.

<table>
<thead>
<tr>
<th>R</th>
<th>T</th>
<th>N</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>125</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>246</td>
<td>500</td>
<td>0.492</td>
</tr>
<tr>
<td>2</td>
<td>1088</td>
<td>2000</td>
<td>0.544</td>
</tr>
<tr>
<td>3</td>
<td>4778</td>
<td>8000</td>
<td>0.597</td>
</tr>
<tr>
<td>4</td>
<td>21240</td>
<td>32000</td>
<td>0.664</td>
</tr>
<tr>
<td>5</td>
<td>103970</td>
<td>128000</td>
<td>0.812</td>
</tr>
</tbody>
</table>

Table 6: M1 and M2 statistics for the refinement of the Pentagonal mesh (Figure 12). Average (\(\mu\)) and Standard Deviation (\(\sigma\)).

<table>
<thead>
<tr>
<th>R</th>
<th>N</th>
<th>(\mu(M1))</th>
<th>(\mu(M2))</th>
<th>(\sigma(M1))</th>
<th>(\sigma(M2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>125</td>
<td>26.544</td>
<td>14.392</td>
<td>16.569</td>
<td>7.156</td>
</tr>
<tr>
<td>1</td>
<td>500</td>
<td>6.910</td>
<td>3.800</td>
<td>2.204</td>
<td>1.668</td>
</tr>
<tr>
<td>2</td>
<td>2000</td>
<td>6.200</td>
<td>3.048</td>
<td>1.699</td>
<td>1.103</td>
</tr>
<tr>
<td>3</td>
<td>8000</td>
<td>5.997</td>
<td>2.831</td>
<td>1.553</td>
<td>0.883</td>
</tr>
<tr>
<td>4</td>
<td>32000</td>
<td>5.482</td>
<td>2.412</td>
<td>1.122</td>
<td>0.800</td>
</tr>
<tr>
<td>5</td>
<td>128000</td>
<td>5.370</td>
<td>2.204</td>
<td>0.947</td>
<td>0.757</td>
</tr>
</tbody>
</table>

Figure 13: Evolution of \(\mu(M1)\) and \(\mu(M2)\) for the refinement of the Pentagonal mesh (Figure 12).

Figure 14: M1 and M2 histograms for the refinement of the Pentagonal mesh (Figure 12).

Figure 15: LEPP-balancing degree evolution for the refinement of Delaunay and Pentagonal meshes.

Table 7: M1 and M2 statistics for the refinement of the Gran Canaria mesh (Figure 16). Average (\(\mu\)) and Standard Deviation (\(\sigma\)).

<table>
<thead>
<tr>
<th>R</th>
<th>N</th>
<th>(\mu(M1))</th>
<th>(\mu(M2))</th>
<th>(\sigma(M1))</th>
<th>(\sigma(M2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>592</td>
<td>6.619</td>
<td>3.451</td>
<td>2.764</td>
<td>1.660</td>
</tr>
<tr>
<td>1</td>
<td>736</td>
<td>6.690</td>
<td>3.539</td>
<td>2.577</td>
<td>1.663</td>
</tr>
<tr>
<td>2</td>
<td>1230</td>
<td>6.505</td>
<td>3.383</td>
<td>2.265</td>
<td>1.601</td>
</tr>
<tr>
<td>3</td>
<td>2624</td>
<td>6.019</td>
<td>2.918</td>
<td>1.679</td>
<td>1.051</td>
</tr>
<tr>
<td>4</td>
<td>9258</td>
<td>5.513</td>
<td>2.426</td>
<td>1.162</td>
<td>0.601</td>
</tr>
<tr>
<td>5</td>
<td>30730</td>
<td>5.247</td>
<td>2.367</td>
<td>0.681</td>
<td>0.509</td>
</tr>
<tr>
<td>6</td>
<td>41448</td>
<td>5.212</td>
<td>2.189</td>
<td>0.564</td>
<td>0.328</td>
</tr>
</tbody>
</table>
(a) Refinement step 1, 326 terminal triangles (lighter shade triangles), 736 total triangles.

(b) Refinement step 3, 1588 terminal triangles (lighter shade triangles), 2624 total triangles.

(c) Refinement step 4, 7020 terminal triangles (lighter shade triangles), 9258 total triangles.

Figure 16: Gran Canaria mesh. Local 4T-LE refinement.

Figure 17: M1 and M2 evolution for the refinement of the Gran Canaria mesh (Figure 16).

Figure 18: M1 and M2 histograms for the refinement of the Gran Canaria mesh (Figure 16).

Table 8: Statistics for the refinement of the Gran Canaria mesh (Figure 16). \( R \) = Refinement step, \( T \) = Triangles in Terminal Pairs, \( N \) = Triangles, \( B \) = LEPP-Balancing Degree.

<table>
<thead>
<tr>
<th>R</th>
<th>T</th>
<th>N</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>248</td>
<td>592</td>
<td>0.41891</td>
</tr>
<tr>
<td>1</td>
<td>326</td>
<td>736</td>
<td>0.44293</td>
</tr>
<tr>
<td>2</td>
<td>672</td>
<td>1230</td>
<td>0.54634</td>
</tr>
<tr>
<td>3</td>
<td>1588</td>
<td>2624</td>
<td>0.60518</td>
</tr>
<tr>
<td>4</td>
<td>7020</td>
<td>9258</td>
<td>0.75826</td>
</tr>
<tr>
<td>5</td>
<td>26282</td>
<td>30730</td>
<td>0.85525</td>
</tr>
<tr>
<td>6</td>
<td>35746</td>
<td>41448</td>
<td>0.86243</td>
</tr>
</tbody>
</table>
5. CONCLUSIONS

In this work we have studied the propagation problem associated with longest edge based refinement algorithms in 2D. We have theoretically proved in the paper that the propagation path asymptotically extends on average to a few neighbor adjacent triangles. This result has also been numerically demonstrated for repeated local refinement. The extent of refinement for triangle $t$ defines a Conformity Neighborhood characterized by two parameters ($M_1(t)$ and $M_2(t)$).

When repeated uniform refinement is applied to an initial arbitrary triangular mesh, the average of the parameters $M_1(t)$ tends to 5 and the average of $M_2(t)$ tends to 2. This implies for local refinement of practical applications that on average the propagation of secondary refinements induced by specified refinements will be limited to a proportionally small number of elements with a confined limit. In view of this, an asymptotic estimate of the cost is easily determined: since the cost of refinement of a single triangle is bounded by a small constant $c$ and the number of triangles in the conformity neighborhood of any such triangle is 5 on average, the asymptotic estimate of the cost to refine a triangle is obviously $6c$.

We also have introduced the concept of LEPP-balancing degree (ratio between triangles in terminal pairs and total triangles in a mesh) for longest edge refinement of meshes and have proved that the LEPP-balancing degree asymptotically tends to 1. These results are also a global measure of the improvement of the generated meshes on the size of the conformity neighborhood.

The counterpart 3D propagation problem needs a more complex study because the number of connectivity patterns are considerably higher than in 2D. It should be noted that the 4T-LE refinement uses three partial division patterns while the extension to three dimensions, the 8T-LE partition may involve more than fifty partial divisions for the sake of conformity [11]. We have made some exploratory numerical studies of LEPP behavior for refinement of tetrahedral meshes and this topic is the subject of our continuing work.

Acknowledgements

This research has been partially supported by Sandia grant #56522 and ULPGC grant UNI2002/22.

References

WHEN AND WHY RUPPERT’S ALGORITHM WORKS

Gary L. Miller 1 Steven E. Pav 2 Noel J. Walkington 2

1 Carnegie Mellon University, Pittsburgh, PA 15213 glmiller@cs.cmu.edu
2 Carnegie Mellon University, Pittsburgh, PA 15213 [spav|noelw]@andrew.cmu.edu

ABSTRACT

An “adaptive” variant of Ruppert’s Algorithm for producing quality triangular planar meshes is introduced. The algorithm terminates for arbitrary Planar Straight Line Graph (PSLG) input. The algorithm outputs a Delaunay mesh where no triangle has minimum angle smaller than $26.45^\circ$ except “across” from small angles of the input. No angle of the output mesh is smaller than $\arctan\left(\frac{\sin \theta^*}{2 - \cos \theta^*}\right)$ where $\theta^*$ is the minimum input angle. Moreover no angle of the mesh is larger than $137.1^\circ$. The adaptive variant is unnecessary when $\theta^*$ is larger than $36.53^\circ$, and thus Ruppert’s Algorithm (with concentric shell splitting) can accept input with minimum angle as small as $36.53^\circ$. An argument is made for why Ruppert’s Algorithm can terminate when the minimum output angle is as large as $30^\circ$.

Keywords: mesh generation, Ruppert’s Algorithm, computational geometry, triangular

1. INTRODUCTION

The Delaunay Refinement Algorithm, first described by Ruppert, accepts a set of points and a set of segments, augments the point set with Steiner points, and returns the Delaunay Triangulation of the augmented set. For suitable input, the triangulation conforms to the input, has no angle smaller than some parameterizable $\kappa$ (which is no larger than $\arcsin \frac{1}{\sqrt{3}} \approx 20.7^\circ$), and exhibits “good grading,” i.e., short edges in the triangulation are attributable to nearby input features which are close together. The number of triangles in the output is within a constant of optimal [1].

The algorithm has the advantage of being relatively easy to state and implement, and has been the object of great scrutiny and interest. Since its introduction, the algorithm and the analysis of the algorithm have been improved and modified: the class of known acceptable input has been expanded [2]; a variant algorithm has been developed to handle small input angles [3]; the algorithm has been adapted to accept curved input [4]; it also has been generalized to higher dimensions [2, 5, 6, 7].

Ruppert’s original analysis required that no input segments meet at acute angles, and guaranteed that no angle in the output was smaller than a parametrizable $\kappa < \arcsin \frac{1}{\sqrt{3}}$. As $\kappa \leq \arcsin \frac{1}{\sqrt{3}}$, the proved bound on the number of Steiner Points approaches infinity [1], though this behaviour is not seen experimentally; rather, the Delaunay Refinement Algorithm is often run with $\kappa$ as great as $\pi/6$ or greater without diverging. The input condition has been relaxed to a $\pi/3$ lower bound on input angles [3, 5]. The algorithm has been observed to terminate on some input with smaller (in some cases much smaller) input angles.

Shewchuck demonstrated an alteration of the algorithm, the so-called “Terminator,” which accepts input with arbitrary minimum angle, $\theta^*$, producing Delaunay meshes with no output angle smaller than $\arcsin \left(\frac{\sin \theta^*}{\frac{\theta^*}{2}}\right)/\sqrt{2}$. This variant is adaptive in the sense that it leaves some small angles in the output mesh, while most angles are larger than $\arcsin \frac{1}{\sqrt{3}}$. The location of the small output angles cannot be determined very much beyond the statement that they are “near input angles less than . . . 60°.” Moreover, the
analysis of this scheme comes without grading guarantees, and thus no optimality claim [3].

We here demonstrate an alteration of the algorithm which outputs meshes where all output angles are greater than $\arcsin 2^{-7/6} \approx 26.45^\circ$, except those whose shortest edge is "opposite" an input angle $\theta < 36.53^\circ$; in this case, the output angle is no less than $\arctan \left( \frac{\sin \theta}{2 \cos \theta} \right)$. Moreover, in spite of the potential of arbitrarily small output angles, this algorithm can guarantee that no output angle is larger than around $\pi - 2 \arcsin \sqrt{\frac{2}{3}} \approx 137.1^\circ$. In this sense the algorithm contrasts favorably with the Terminator, which has no upper bound other than the naive one of $\pi - 2 \arcsin \left[ \frac{\sin \left( \frac{\theta}{2} \right)}{\sqrt{2}} \right]$, which deteriorates when $\theta^*$ is small. Moreover, our algorithm comes with grading and optimality guarantees, and is fairly simple.

In the case where $\theta^* \geq 36.53^\circ$, our analysis shows that the variant algorithm is unnecessary, and that Ruppert’s original algorithm with circular shell splitting comes with the same output and optimality guarantees.

In this work we employ the strategy of Shewchuck [2], i.e., termination is proved without showing good grading. This is done since a relatively accessible and complete proof of the “termination-only” result may be given in the limited amount of available space. The proof of good-grading is quite a bit more involved [8].

The restriction that $\theta^* \leq \pi/3$ is merely for convenience; asserting a larger lower bound does not give any better results.

**Assumption 2.2 (Output).** The algorithm outputs sets of points, segments, triangles, $P'$, $S'$, $T'$, respectively, satisfying:

(a) **Complex:** The output collectively forms a simplicial complex, i.e., $\{\emptyset\} \cup P' \cup S' \cup T'$ is closed under taking boundaries, and under intersection.

(b) **Delaunay:** Each triangle of $T'$ has the Delaunay property with respect to $P'$.

(c) **Conformality:** $P' \subseteq P'$, and for every $s \in S$, $s$ is the union of segments in $S'$.

(d) **Quality:** There are few or no “poor-quality” triangles in $T'$.

(e) **Cardinality:** Few Steiner points have been added, i.e., $|P' \setminus P|$ is small.

One passable definition of item (d) is that there are some reasonably large constants $0 < \alpha \leq \omega < 2 + \pi\alpha$ such that for every triangle $t \in T'$, no angle of $t$ is smaller than $\alpha$ or larger than $\pi - 2\omega$. However, such a guarantee is not consistent with conformality of the triangulation (item (c)) when the input contains angles less than $\alpha$. A weaker definition is that most triangles satisfy the above condition, and those that do not (a) are describably near an input angle of size $\theta$, (b) have no angle smaller than $\theta - \mathcal{O}(\theta^2)$, and (c) have no angle larger than $\pi - 2\omega$.  

**3. THE ALGORITHM**

We describe a whole class of algorithms, which we collectively refer to as “the” Delaunay Refinement Algorithm. This class contains Ruppert’s original formulation [1], as well as the incremental version [5].

We suppose that the algorithm maintains a set of “committed” points, initialized to be the set of input points, $P$. The algorithm also maintains a set of “current” segments, initialized as the input set, $S$. The algorithm will “commit” points to the set of committed points. At times the algorithm will choose to “split” a current segment; this is achieved by removing the segment from the set of current segments, adding the two half-length subsegments which comprise the segment to the set of current segments, and committing to the midpoint of the segment. The word “midpoint” should be taken to mean one of these segment midpoints for the remainder of this work, to distinguish them from the other kind of Steiner Point, which will be called “circumcenters.”

The algorithm has two high-level operations, and will continue to perform these operations until it can no longer do so, at which time it will output the committed points, the current segments and the Delaunay

---

**2. THE MESHING PROBLEM**

The meshing problem is described in terms of the input to the algorithm and the expected conditions on the output. The input to the mesh is defined as follows:

**Assumption 2.1 (Input).** The input to the meshing problem consists of a finite set of points, $\mathcal{P} \subseteq \mathbb{R}^2$, and a set of segments $\mathcal{S}$ such that

(a) the two endpoints of any segment in $\mathcal{S}$ are in $\mathcal{P}$,

(b) any point of $\mathcal{P}$ intersects a segment of $\mathcal{S}$ only at an endpoint,

(c) two segments of $\mathcal{S}$ meet only at their endpoints, and

(d) the boundary of the convex hull of $\mathcal{P}$ is the union of segments in $\mathcal{S}$.

Let $\Omega$ denote the convex hull of the input, and let $0 < \theta^* \leq \pi/3$ be a lower bound on the angle between any two intersecting segments of the input.

Items (a)-(c) characterize $(\mathcal{P}, \mathcal{S})$ as a Planar Straight Line Graph (PSLG); item (d) can always be satisfied by augmenting an arbitrary PSLG which does not satisfy it with a bounding polygon (typically a rectangle).
The heuristics involved with determining which operation to perform when and on which segment or poor-quality triangle are not relevant to our discussion. This is not to say that they might not affect ease of implementation, running time, cardinality of the final set of committed points, parallelizability, etc. A common heuristic (and the one chosen by Ruppert and others) is to prefer conformity operations over quality operations, which likely results in a smaller output, and which simplifies detecting that a circumcenter is outside of $\Omega$. A description of a member of this class of algorithms would have to include some discussion of how to figure out which current segments are encroached, which triangles are suitable for removal via the quality operation, how to deal with degeneracy, etc. We do not concern ourselves with these details (though see [9, 10, 11, 5, 12, 2, 13]).

### 3.1 When is Adaptivity Necessary?

We here make the claim that the Delaunay Refinement Algorithm is as good as its adaptive variant when the latter is used with a small output angle parameter $\hat{\kappa}$. The claim is formalized as follows:

**Claim 3.1.** Suppose that we can guarantee that if the Adaptive Delaunay Refinement Algorithm is run with output angle parameter $\kappa$, on any appropriate input with minimum input angle $\theta^*$, that (a) the algorithm terminates, (b) no angle of the output mesh is smaller than $\kappa$, and (c) no angle is larger than $\pi - 2\omega$.

Then if the Delaunay Refinement Algorithm is run on any appropriate input with minimum input angle $\theta^*$, using output angle parameter $\kappa = \hat{\kappa}$, then (a) the algorithm terminates, (b) no angle of the output mesh is smaller than $\kappa$, and (c) no angle is larger than $\pi - 2\omega$.

**Proof.** The Adaptive Delaunay Refinement Algorithm only attempts to remove a Delaunay triangle if it has minimum angle smaller than $\hat{\kappa}$. Moreover, it produces meshes with no angle smaller than $\hat{\kappa}$. Then the \{(Quality')\} operation could be rewritten as follows:

\[(\text{Quality'}) \text{ If } a, b, c \text{ are committed points, the circumcircle of the triangle } \Delta abc \text{ contains no committed point, } \angle abc < \hat{\kappa}, \text{ the circumcenter, } p, \text{ of the triangle is inside } \Omega \text{ and either (i) both } a, b \text{ are midpoints on distinct non-disjoint input segments, sharing input endpoint } x, \text{ and } \angle axb > \pi/3, \text{ or (ii) } a, b \text{ are not midpoints on adjoining input segments, then attempt to commit } p. \text{ If, however, the point } p \text{ encroaches any current segment, then do not commit to point } p, \text{ rather in this case split one, some, or all of the current segments which are encroached by } p.\]

In summary, the algorithm removes angles smaller than $\hat{\kappa}$ except when the opposite edge spans a small angle in the input, in which case the small output angles are ignored. For this variant we call $\hat{\kappa}$ the output angle parameter; the output mesh may well contain angles smaller than $\hat{\kappa}$. We will let $\alpha$ be the minimum angle in the output mesh.

The heuristics involved with determining which operation to perform when and on which segment or poor-quality triangle are not relevant to our discussion. This is not to say that they might not affect ease of implementation, running time, cardinality of the final set of committed points, parallelizability, etc. A common heuristic (and the one chosen by Ruppert and others) is to prefer conformity operations over quality operations, which likely results in a smaller output, and which simplifies detecting that a circumcenter is outside of $\Omega$. A description of a member of this class of algorithms would have to include some discussion of how to figure out which current segments are encroached, which triangles are suitable for removal via the quality operation, how to deal with degeneracy, etc. We do not concern ourselves with these details (though see [9, 10, 11, 5, 12, 2, 13]).
Thus we will first examine the adaptive variant, then use the results to analyze the regular Delaunay Refinement Algorithm.

The analysis that follows should be read with a tacit understanding that it can be applied to the Delaunay Refinement Algorithm as well, if $\kappa$ is set properly. For example, it will be shown that if an input with $\theta^* \approx 36.53^\circ$ conforms to Assumption 4.2, then the Adaptive Delaunay Refinement Algorithm with $\kappa = 26.45^\circ$ will terminate leaving no angle in the output mesh smaller than $\kappa$, and no angle larger than $\pi - 2\kappa$. Then we can immediately claim that the Delaunay Refinement Algorithm (i.e., Ruppert’s Algorithm) with $\kappa = 26.45^\circ$ will also terminate on the same input, and with the same grading guarantees.

So the adaptive variant is only necessary if $\theta^*$ is small, say smaller than about 36.53°. When $\theta^*$ is small, the adaptive variant will remove small angles where this is possible, i.e., away from small input angles.

4. PRELIMINARIES

Some preliminary definitions and results are essential to the exposition. First there is the matter of terminology: if $p$ is a committed point that was the midpoint of a segment, we say this segment is the “parent” segment (or parent subsegment) of $p$; the “radius” of a segment is half its length, while the radius associated with a midpoint is the radius of its parent segment; any segment derived from a segment $s \in \mathcal{S}$ by splitting is a “subsegment” of (or on) $s$; segments in $\mathcal{S}$ which share an endpoint are nondisjoint; distinct nondisjoint segments are said to be “adjoining.”

Throughout this work, we let $|x - y|$ denote the Euclidian distance between points $x$ and $y$. For a segment $S$, we let $|S|$ denote the length of the segment. Local feature size is defined in terms of the input, and is the classical definition due to Ruppert:

Definition 4.1 (Local Feature Size). For a point $x \in \mathbb{R}^2$, the local feature size at $x$, relative to an input PSLG, $(\mathcal{P}, \mathcal{S})$, is the minimum $r$ such that a closed ball of radius $r$ centered at $x$ intersects at least two disjoint features of $\mathcal{P} \cup \mathcal{S}$. The local feature size is a Lipschitz function, i.e., $\text{lfs}(x) \leq |x - y| + \text{lfs}(y)$.

This definition is illustrated in Figure 1.

For the proof we require an extra condition on the input:

Assumption 4.2. In addition to those of Assumption 2.1 we make the following assumption:

(a) If $S_1, S_2$ are two adjoining input segments that meet at angle other than $\pi$, then they have the same length modulo a power of two, that is $|S_1| = 2^k$ for some integer $k$.

It is simple to show that this assumption can be satisfied by the addition of no more than $2 |\mathcal{S}|$ augmenting points, effectively redefining the input $|\mathcal{S}|$. Later we will argue that Ruppert’s strategy of splitting on concentric circular shells obviates this additional assumption [1].

5. MIDPOINT-MIDPOINT INTERACTIONS

Ruppert noted that one way his algorithm could fail was due to infinite cascades of segment midpoints each encroaching on an adjoining subsegment; the prescribed cure was concentric shell splitting [1], which puts input into a form which satisfies Assumption 4.2 on an as-needed basis. To simplify the proof, we assume the input satisfies this assumption up-front, then ease the restriction later. In this section we show how this assumption can prevent infinite cascades of midpoints.

The classic result on Ruppert’s Algorithm for input satisfying a $\pi/3$ angle condition can be proven with the following purely geometric lemma [8].

Lemma 5.1. Given two rays, $R$ and $R'$ from a point $x$ with angle $\theta$ between them, suppose there is a ball of radius $r$ with center $p$ on ray $R$ such that the ball does not contain $x$ but does contain a point $q$ of $R'$. Then if $\pi/4 \leq \theta < \pi/2$, $|q - x| / |p - x| < |q - x| / |p - q| \leq 2 \cos \theta$. 

Figure 1: For a number of points in the plane, the local feature size with respect to the given input is shown. About each of the points $u, v, w, x, y, z$ is a circle whose radius is the local feature size of the center point. The point $u$ is an input point.
Given the $\pi/3$ angle condition, the right hand side of the inequality in the lemma is no greater than 1. Roughly this guarantees that radii do not “dwindle,” or in terms of Shewchuk’s dataflow diagrams, the midpoint-midpoint loop does not admit a decrease in insertion radius [2].

The following lemma makes the same guarantees, but for input which satisfy Assumption 4.2. The lemma explicitly states that the radii are non-dwindling, though note these are actual segment radii, not Shewchuk’s insertion radii, which is also known as nearest neighbor distance. Using the non-dwindling property of segment radii, we will prove termination of the algorithm by demonstrating a lower bound on a segment’s radius at time of splitting.

This lemma takes care of the case where a midpoint encroaches a segment on a nondisjoint input feature. In the following sections, we consider another way in which a midpoint can trigger such a segment split, namely via sequences of triangle circumcenters.

**Lemma 5.2.** Suppose that the input conforms to Assumption 4.2. Let $p$ be the midpoint of a segment which is encroached by a committed point, $q$, on an adjoining input segment. Let $r_p$ be the radius associated with $p$, and $r_q$ that of $q$. Then $r_q \leq r_p$, and moreover,

$$|p - q| \geq 2r_q \sin \frac{\theta}{2},$$

where $\theta$ is the angle between the two input segments.

**Proof.** Let $(x, y), (x, z)$ be the two input segments containing, respectively, $p, q$. Let $(a, b)$ be the subsegment of which $p$ is midpoint. Let $(c, d)$ be that for which $q$ is midpoint. Assume that $a$ is closer to $x$ than $b$ is, and assume $c$ is closer to $x$ than $d$ is. It may be the case that $x = a$, or $x = c$.

It is easy to show that, $\log_2 \frac{|x-a|}{|a-b|}$, and $\log_2 \frac{|x-a|}{|c-d|}$ are nonnegative integers. By Assumption 4.2, and since $\theta \neq \pi$, $\log_2 \frac{|x-a|}{|a-b|}$ is an integer. Thus $\log_2 \frac{|x-a|}{|a-b|} = \log_2 \frac{r_p}{r_q} = j$ is also an integer. We wish to show that it is nonnegative.

A geometric argument gives $|x-a| < |x-q| < |x-b|$, so that $|x-a| < |x-c| + r_q < |x-a| + 2r_p$. It can then be shown that $k = \frac{|x-c|}{|x-a|} = \frac{|x-q|}{2r_p}$ is a nonnegative integer, as is, $mutatis mutandis$, $l = \frac{|x-c|}{2r_q}$. Thus

$$2kr_p < (2l+1)r_q < 2(k+1)r_p, \text{ or } 2^{j+1}k < (2l+1) < 2^{j+1}(k+1), \text{ and so }$$

$$2l + 1 < k < \frac{2l+1}{2^{j+1}}.$$

If $j$ is a negative integer, then $2^{j+1}$ is a power of two no greater than 1, in particular it divides any integer, thus $\frac{2l+1}{2^{j+1}} = m$ is an integer. This gives the contradiction that $m - 1 < k < m$ for integer $m, k$. Thus $j$ is a nonnegative integer, or $r_p \geq r_q$.

For the second part, we first show that $|p - q| \geq 2(|x-q| \wedge |x-p|) \sin \frac{\theta}{2}$. We consider the case where $|x-q| \leq |x-p|$; the other case follows mutatis mutandis.

Let $L = \frac{|x-p|}{|x-q|} \geq 1$. Using the cosine rule on $\Delta xpq$,

$$|p - q|^2 = |x - p|^2 + |x - q|^2 - 2|x - p||x - q| \cos \theta.$$

$$= (1 + L^2)|x - q|^2 - 2L|x - q|^2 \cos \theta$$

$$\geq 2L|x - q|^2 - 2L|x - q|^2 \cos \theta$$

$$= 2L|x - q|^2 (1 - \cos \theta),$$

where we have used that $1 + L^2 \geq 2L$. Using $L \geq 1$, we obtain $|p - q|^2 \geq 2 \sin \frac{\theta}{2}$. It is a simple exercise to show that $2\sin \frac{\theta}{2} = \sqrt{2(1 - \cos \theta)}$. Now, clearly $|p - q| \geq r_p \geq r_q$, and $|x-q| \leq r_q$, so the result $|p - q| \geq 2r_q \sin \frac{\theta}{2}$ holds, as desired. \qed

## 6. CIRCUMCENTER SEQUENCES

We now consider sequences of triangle circumcenter additions.

**Definition 6.1.** A circumcenter sequence is a sequence of points, $\{b_i\}_{i=0}^{l-1}$ such that for $i = 1, 2, \ldots, l-1$, $b_i$ is the circumcenter of a triangle in which $b_{i-1}$ is the more recently committed endpoint of an edge opposite an angle less than $\bar{\alpha}$. The point $b_0$ may be an input point or segment midpoint.

For $i = 0, 1, \ldots, l-2$, let $a_i$ be the other endpoint of the short edge of which $b_i$ is the more recently committed endpoint. In the case where $a_0, b_0$ are both input points, they are committed simultaneously; we imagine a total order on input points which determines the tie. Both $a_0, b_0$ may be midpoints on distinct, nondisjoint input segments. In this case we assume that the triangle with circumcenter $b_1$ was removed by a (QUALITY’) operation because of a small angle opposite $a_0, b_0$. In particular this means that we assume the angle subtended by the input segments containing $a_0, b_0$ is at least $\pi/3$ in this case.

When talking about such sequences, for $i = 1, 2, \ldots, l-1$, let $r_i$ be the circumradius of the triangle associated with $b_i$. Note that $r_i = |b_i - b_{i-1}| = |b_i - a_{i-1}|$, and that $|a_i - b_i| \geq r_i$. We let $r_0 = |b_0 - a_0|$, i.e., the length of the first short edge.

Note that for a circumcenter sequence, $\{b_i\}_{i=0}^{l-1}$ the points $b_1, b_2, \ldots, b_{l-2}$ are circumcenters which have been committed, $b_{l-1}$ is a circumcenter, though it may be rejected, and $b_0$ may be any type of point. If $b$ is a triangle circumcenter, there is always a circumcenter
sequence ending with b, although it may be a trivial sequence of two elements. Any circumcenter sequence whose first element, b_0, is a triangle circumcenter may be extended to a maximal sequence whose first element is either a segment midpoint or an input point.

The following geometric lemma is the key result which allows us to make the arcsin 2^{-7/6} output guarantee. It states that only circumcenter sequences longer than a certain length can “turn” around a 180° feature.

**Lemma 6.2.** Let S_1, S_2 be two segments with disjoint interiors on a common line, L. Assume that |S_2| ≤ |S_1|, i.e., S_2 is no longer than S_1. Let b_0 be the midpoint of S_1, and let a_0 be some other point. Let \{b_i\}_{i=1}^{l-1} be a circumcenter sequence such that b_{l-1} is inside the diametral circle of S_2, and such that b_1 is the circumcenter of a triangle with edge (a_0, b_0) opposite an angle smaller than \(\hat{\kappa}\). Then \(l \geq 4\).

Note that unlike in the regular terminology of circumcenter sequences, this lemma makes no assumptions about which of \(a_0, b_0\) was committed first. This is why we have chosen to index the circumcenter sequence from \(i = 1\) instead of the usual \(i = 0\).

**Proof.** The basic argument is sketched in Figure 2. The point \(b_1\) is the circumcenter of a triangle whose circumcircle does not contain the point \(x\), which is the endpoint of \(S_1\) closer to \(S_2\). However, this circumcircle has \(b_0\) on it, so \(b_1\) must be in the closed halfspace defined by the bisector of \(x\) and \(b_0\) and which does not contain \(x\), as shown in Figure 2(a). Thus \(b_1\) cannot be in the diametral circle of \(S_2\), which is in the open halfspace on the other side of this bisector. Now let \(G\) be the bisector of points \(b_1\) and \(x\). Point \(b_2\) is the center of a circle which does not contain \(x\), but has \(b_1\) on its boundary, since \(b_1\) is one of the vertices of the triangle which \(b_2\) is added to remove. Thus \(b_2\) must be either on the line \(G\), or in the open halfspace defined by \(G\) that is closer to the point \(b_1\). In Figure 2(b), this is the halfspace to the upper right of \(G\).

It then suffices to show that the closure of the diametral ball of \(S_2\) is contained in the other open halfspace defined by \(G\), and thus \(b_2\) cannot encroach \(S_2\).

Let \(m\) be the intersection of \(L\) and \(G\); take \(m\) to be the midpoint of \(S_2\), and \(m'\) is its projection onto \(G\). Let \(x'\) be the projection of \(x\) onto \(G\). Let \(y\) be the projection of \(b_1\) onto \(L\). See Figure 3. The point \(x\) is clearly between \(m\) and \(z\), otherwise \(x\) would be in the halfspace closer to \(b_1\) than to \(x\), a contradiction. Thus |\(m - z\)| = |\(m - x\)| + |\(x - z\)|.

By congruency of the three triangles of Figure 3, \(|m - m'| = |x - x'| = |x - y|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.|x - x'|.|x - z|.|x - b_1|.

Let \(r = \frac{S_2}{2} \leq \frac{|S_1|}{2}\), by assumption. Since \(S_1, S_2\) have disjoint interiors, |\(m - x\)| ≥ \(r\). Then |\(m - z\)| ≥ \(r + \)

Figure 2: The head of a circumcenter sequence is shown; the point \(b_1\) must be to the right of the bisector of \(b_0\) and \(x\), and so it cannot encroach \(S_2\), which is on the other side of this bisector, as shown in (a). In (b) the bisector of \(b_1\) and the point \(x\) is shown. Since \(b_2\) cannot be closer to \(x\) than to \(b_1\), and since the diametral circle of \(S_2\) is on the opposite side the bisector, \(b_2\) cannot encroach \(S_2\). In this case, \(a_0\) is shown to be outside the diametral circle of \(S_1\). This is not a necessary hypothesis for this lemma.

Figure 3: The geometric heart of the argument is shown, with three congruent triangles, \(\Delta mm'z, \Delta xx'z, \Delta xyb_1\).
\[ |x - z|, \]
\[
\begin{align*}
|m - m'| &= \frac{|x - x'||m - z|}{|x - z|}, \\
&\geq \frac{|x - x'| (r + |x - z|)}{|x - z|}, \\
&\geq \frac{|x - x'| r + |x - x'|}{|x - z|}, \\
&= \frac{|x - y|}{|x - b_1|} r + |x - x'|.
\end{align*}
\]

As noted above, \( b_1 \) is to the right of the bisector of \( x \) and \( b_0 \), so \( |x - y| \geq \frac{|x - b_0| = \frac{|S_1|}{4} \geq \frac{r}{2} \). Note also that \( |x - b_1| = 2|x - x'| \). Then
\[
|m - m'| \geq \frac{r^2}{4} + |x - x'|.
\]

The right hand side is minimized when \( |x - x'| = \frac{r}{2} \), where the right hand side has value \( r \). Note, however, that \( |x - x'| \geq \frac{r}{2} \geq \frac{1}{2} \sin \theta \frac{|S_1|}{4} > \frac{r}{2} \), so the right hand side will be strictly larger than \( r \).

That is, \( |m - m'| > r \), and thus the distance from \( m \) to \( G \), which is \( |m - m'| \), is greater than the radius of the diametral circle of \( S_2 \). Then the closed diametral circle of \( S_2 \) is contained in the open halfspace opposite \( b_1 \), as desired.

This lemma allows us to prove a better output angle for the Delaunay Refinement Algorithm. Previous proofs required \( 2 \sin \tilde{\kappa} \leq \frac{1}{\sqrt{2}} \); by the lemma, the following proof only requires that \( (2 \sin \tilde{\kappa})^3 \leq \frac{1}{\sqrt{2}} \). A better output angle could be guaranteed if the lemma could be improved; this would have to be via some alternative of the algorithm, as the example of Figure 4 shows the lemma cannot be extended in the naive setting. We return to this matter later.

Since \( \tilde{\kappa} < \pi/6 \), we can establish a geometric series which gives the following lemma and its corollary. The corollary describes how a segment midpoint which is not caused by a midpoint encroaching the segment is caused by some other midpoint or input point.

**Lemma 6.3.** Suppose \( \{b_i\}_{i=0}^{l-1} \) is a circumcenter sequence. For \( i > 0 \), let \( \tilde{r}_i \) be the circumradius associated with \( b_i \). Then for \( i = 1, 2, \ldots, l - 1 \),
\[
\begin{align*}
\tilde{r}_{i-1} &< 2 \tilde{r}_i \sin \tilde{\kappa} \text{ and therefore } \tilde{r}_i < (2 \sin \tilde{\kappa})^{i-1} \tilde{r}_{i-1}, \text{ and} \\
|b_{i-1} - b_i| &< \frac{\tilde{r}_{i-1}}{2 \sin \tilde{\kappa}}, \text{ and } |b_{i-1} - a_i| < \frac{\tilde{r}_{i-1}}{2 \sin \tilde{\kappa}}.
\end{align*}
\]

**Proof.** By definition, \( b_i \) is the circumcenter of a triangle of radius \( \tilde{r}_i \), which has a short edge no shorter than \( \tilde{r}_{i-1} \) opposite an angle less than \( \tilde{\kappa} \). By the sine rule, then \( 2 \tilde{r}_i \sin \tilde{\kappa} > \tilde{r}_{i-1} \).

---

**Figure 4:** A circumcenter sequence, \( \{b_i\}_{i=0}^{3} \), is displayed, which shows that Lemma 6.2 cannot be extended. The segments \( S_1, S_2 \) are shown, with their diametral circles. The points \( b_1, b_2, b_3 \) are circumcenters of triangles (shown) with an angle smaller than \( \pi/6 \). The point \( b_3 \) encroaches \( S_2 \).

Using this repeatedly gives \( \tilde{r}_i < (2 \sin \tilde{\kappa})^{i-1} \tilde{r}_{i-1} \). Since \( 2 \sin \tilde{\kappa} < 1 \), we may bound the distance from \( b_1 \) to \( b_{l-1} \) by the geometric series, as follows:
\[
|b_{l-1} - b_i| \leq |b_{l-1} - b_{l-2}| + |b_{l-2} - b_{l-3}| + \ldots + |b_{i+1} - b_i|,
\]
\[
< \tilde{r}_{l-1} + \tilde{r}_{l-2} + \ldots + \tilde{r}_{i+1},
\]
\[
< \tilde{r}_{l-1} + (2 \sin \tilde{\kappa}) \tilde{r}_{l-2} + \ldots + (2 \sin \tilde{\kappa})^{i-1} \tilde{r}_{i-1},
\]
\[
< \frac{1}{1 - 2 \sin \tilde{\kappa}} \tilde{r}_{l-1}.
\]

The bound for \( |b_{l-1} - a_i| \) follows since \( |b_{i+1} - b_i| = |b_{i+1} - b_i| = \tilde{r}_{i+1} \), and the above analysis suffices.

**Corollary 6.4.** Suppose that segment \( s_p \) with midpoint \( p \) and radius \( r \) was split, but the segment was not encroached by a committed point. Then there is some maximal circumcenter sequence \( \{b_i\}_{i=0}^{l-1} \) such that \( b_{l-1} \) “yielded” to \( p \), causing it to be committed. Moreover, \( \tilde{r}_i < (2 \sin \tilde{\kappa})^{i-1} \sqrt{2r_p} \), \( |p - b_i| \leq \eta r_p \), and \( |p - a_i| \leq \eta r_p, \) for \( i = 0, 1, \ldots, l - 1, \) with \( \eta = 1 + \frac{\sqrt{2}}{1 - 2 \sin \tilde{\kappa}} \).

**Proof.** Since \( b_{l-1} \) was the center of an empty circumcircle, but encroached \( s_p \), then \( \tilde{r}_{l-1} \leq \sqrt{2r_p} \). Using the lemma gives the desired bound on \( \tilde{r}_i \). By the lemma, and since \( \tilde{\kappa} < \pi/6 \), \( \tilde{r}_i \leq \tilde{r}_{l-1} \). Then
\[
|p - b_i| \leq |p - b_{l-1}| + |b_{l-1} - b_i| \leq r_p + \frac{\tilde{r}_{l-1}}{1 - 2 \sin \tilde{\kappa}}
\]
\[
\leq \left( 1 + \frac{\sqrt{2}}{1 - 2 \sin \tilde{\kappa}} \right) r_p = \eta r_p.
\]

The bound on \( |p - a_i| \) follows, mutatis mutandis.
7. PROVING TERMINATION

We prove termination not by showing that output mesh edges are well-graded, rather by showing that the algorithm can create no mesh edge smaller than dictated by the minimum local feature size of the input. Towards this end we define
\[ \text{lfs}_{\text{min}} = \min \{ \text{lfs}(x) \mid x \in \Omega \}. \]

**Theorem 7.1 (Radius Bounds).** Suppose that the input to the Adaptive Delaunay Refinement Algorithm conforms to Assumption 4.2. Suppose that \( \hat{r} \leq \arcsin 2^{-7/6} \). Then there is a constant, \( \mu \), depending on \( \theta^* \) and \( \hat{r} \) such that if \( p \) is the midpoint of a segment, \( s \), of radius \( r \) that is committed by the algorithm, then \( \text{lfs}_{\text{min}} \leq \mu r \).

**Proof.** We consider why the segment was split. If there was an input point or a point on a disjoint input sequence that encroached \( s \), then clearly \( \text{lfs}(p) \leq r \), so it suffices to take \( \mu \geq 1 \).

Suppose a midpoint \( q \) on a nondisjoint input sequence encroached \( s \). Using this result inductively we know that \( \text{lfs}_{\text{min}} \leq \mu r_q \), where \( r_q \) is the radius associated with \( q \). By Lemma 5.2, \( r_q \leq r \), which suffices.

Suppose that \( s \) was not encroached by an input point or midpoint, rather it was split when a circumcenter "yielded" to the segment split. Consider a maximal circumcenter sequence, \( \{b_i \}_{i=0}^{lfs} \), ending in the circumcenter \( b_{lfs} \) which yielded to the split of \( s \). By maximality, \( b_0 \) is not a circumcenter. Consider its identity.

If \( b_0 \) is an input point or a midpoint on an input feature disjoint from the segment containing \( s \), then \( \text{lfs}(p) \leq |p - b_0| \leq \eta r \), by Corollary 6.4. Thus it suffices to take \( \eta \leq \mu \).

The only remaining possibility is that \( b_0 \) is a midpoint on an input feature nondisjoint from the one containing \( s \). Let \( r_b \) be the radius associated with \( b_0 \). This radius may be larger or smaller than \( \hat{r}_0 = |b_0 - a_0| \).

We consider the possibilities:

- Suppose \( r_b \leq \hat{r}_0 \). Using this result inductively we have \( \text{lfs}_{\text{min}} \leq \mu r_b \). By Corollary 6.4, \( \hat{r}_0 \leq (2 \sin \hat{r})^{1-1} \sqrt{2r} \). If \( b_0 \) is a midpoint on the same input segment as \( p \) or on a distinct input segment subtending an angle other than \( \pi \), then by Assumption 4.2, \( \log \frac{\hat{r}_0}{r} \) is an integer. But since \( r_b \leq \sqrt{2r} \), it must be a nonnegative one, thus \( r \geq r_b \), so \( \text{lfs}_{\text{min}} \leq \mu r \). The only alternative is \( b_0 \) is a midpoint on a distinct input segment subtending angle \( \pi \) with the one containing \( p \). Then either \( r_b \leq r \), in which case immediately \( \text{lfs}_{\text{min}} \leq \mu r \), or \( r < r_b \), in which case by Lemma 6.2, \( l \geq 4 \), so \( r_b \leq (2 \sin \hat{r})^{1} \sqrt{2r} \). This yields a contradiction when \( \hat{r} \leq \arcsin 2^{-7/6} \), as assumed.

- Suppose \( r_b > \hat{r}_0 \). This means that \( a_0 \) encroached the diametral circle of the subsegment associated with \( b_0 \), and thus, since \( b_0 \) was committed after \( a_0, a_0 \) is not a circumcenter.

If \( a_0 \) is an input point or on an input segment disjoint from the one containing \( b_0 \), then \( \text{lfs}_{\text{min}} \leq |a_0 - b_0| = r_0 \), so it suffices to take \( \mu \geq 1 \).

The alternative is that \( a_0 \) is a midpoint on an input segment adjoining the one containing \( b_0 \). By the definition of the (QUALITY') operation and circumcenter sequences, it must be the case that \( \theta \), the angle between the two input segments is at least \( \pi/3 \). Using Lemma 5.2, we know that \( \hat{r}_0 = |a_0 - b_0| \geq r_a \), where \( r_a \) is the radius associated with \( a_0 \).

If the input segment containing \( a_0 \) is disjoint from the one containing \( p \), then using Corollary 6.4 again it suffices to take \( \eta \leq \mu \).

Otherwise arguments as above show that \( r \geq r_a \), and using this result inductively suffices.

In all it suffices to take \( \mu = \eta = 1 + \frac{\sqrt{2}}{1 - 2 \sin \hat{r}} |p - q| \).

The following corollary gives termination:

**Corollary 7.2.** Suppose the Adaptive Delaunay Refinement Algorithm considers committing point \( p \). Let \( q \) be the closest point that has already been committed. Then \( \text{lfs}_{\text{min}} \leq \frac{\mu}{2 \sin \hat{r}} |p - q| \).

**Proof.** Consider the identity of \( p \).

- Suppose \( p \) is a midpoint, and let \( r \) be the associated radius. If \( r \leq |p - q| \), then the theorem gives \( \text{lfs}_{\text{min}} \leq \mu |p - q| \). If, however, \( r > |p - q| \), then \( p \) encroaches the subsegment of \( p \), so it cannot be a circumcenter (which would have yielded). If \( q \) is an input point or on a disjoint input feature then \( \text{lfs}_{\text{min}} \leq |p - q| \), which suffices. Otherwise \( q \) is a midpoint on a nondisjoint input segment. Then, using, Lemma 5.2, \( |p - q| \geq 2 r_q \sin \hat{r} \), where \( r_q \) is the radius associated with \( q \). Using the theorem on \( q \), we have \( \text{lfs}_{\text{min}} \leq \mu r_q \), which gives the desired result.

- Suppose \( p \) is a circumcenter with associated radius \( r \). Then \( r = |p - q| \), since the triangle is Delaunay. Then \( p \) can be considered the last circumcenter in a circumcenter sequence, and by Lemma 6.3 \( r > \hat{r}_0 \). Then using this corollary inductively on the point \( b_0 \), the first point of the circumcenter sequence, gives the desired result.

Note that this proof entirely ignores the issue of grading. The skeptic might object that all the edges in the
8. OUTPUT QUALITY

Recall that the Adaptive Delaunay Refinement Algorithm ignores angles smaller than the parameter $\kappa$. We will show that small output angles are not too much smaller than a nearby small input angle. The following simple geometric claim gives the output quality guarantee; the idea is to use it with facts about midpoints, the definition of $\text{QUALITY}'$, and the Delaunay property to get the bound on output angles. We omit the proofs due to space constraints.

Lemma 8.1. Let $x, s, q$ be three distinct noncollinear points. Let $p$ be a point on the open line segment from $x$ to $s$. Suppose that $|p - s| \leq |x - s| \leq |x - q|$. Let $\theta = \angle pxq$, and $\phi = \angle psq$. Then

$$\phi \geq \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right).$$

Claim 8.2 (Edge-Apex Rule). Given a triangle $\Delta pqr$ in the Delaunay Triangulation of a set of points, $\mathcal{P}$, with $L$ the line through $p, q$, then $\angle pqr \geq \angle p'qr$ for every $r' \in \mathcal{P}$ that is on the same side of $L$ as $p$, with equality only holding in the case of degeneracy.

We can now state the output guarantee.

Lemma 8.3. Suppose the Adaptive Delaunay Refinement Algorithm terminates for a given input. Let $\Delta pqr$ be a triangle in the output triangulation. Then either

(a) The angle $\angle pqr > \kappa$, or
(b) the points $p$ and $q$ are midpoints on adjoining input segments which meet at angle $\theta < \pi/3$ and

$$\angle pqr \geq \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right).$$

Consequently no angle in the output mesh is smaller than $\min \left\{ \kappa, \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right) \right\}$.

Proof. Suppose that $\angle pqr \leq \kappa$, by the definition of the Adaptive Delaunay Refinement Algorithm, it must be that $p, q$ are midpoints on an adjoining input segment, meeting at an angle, $\theta$, less than $\pi/3$. Let $x$ be the input point common to these segments. Without loss of generality, assume that $|x - p| \leq |x - q|$. The midpoint $p$ is the endpoint of two subsegments of this input segment; let the one farther from $x$ be $(p, s)$. By Claim 77, $|p - s| \leq |p - x|$. Then by Lemma 8.1, $\angle psq \geq \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right)$. Letting $L$ be the line through $p, q$, consider the location of $r$:

- Suppose $r$ is on the same side of $L$ as $x$. By Claim 8.2, $\angle pqr \geq \angle pxq = \theta > \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right)$.
- If $r$ is on the same side of $L$ as $s$, by Claim 8.2, $\angle pqr \geq \angle psq \geq \arctan \left( \frac{\sin \theta}{2 - \cos \theta} \right)$.

We note briefly that $\arctan \left[ \left( \frac{\sin \theta}{2 - \cos \theta} \right) \right] = \theta + O(\theta^2)$, which makes this lower bound much better than that of $\arcsin \left[ \left( \frac{\sin \theta}{2 - \cos \theta} \right) \right]^2 = \frac{\theta}{2\sqrt{2}} + O(\theta^2)$ achieved by Shewchuk's Terminator [3].

The following corollary gives an upper bound on output angles that depends on the output angle parameter, $\kappa$, but not on the minimum output angle. Given $\kappa = \arcsin 2^{-7/6} \approx 26.45^\circ$, it guarantees no output angle is bigger than about $\pi - 2 \arcsin \frac{\sqrt{3} - 1}{2} \approx 137.1^\circ$.

The (omitted) proof relies on the location of small output angles and uses the fact that diametral circles of subsegments are not encroached in the final mesh.

Corollary 8.4. If $\Delta pqr$ is a triangle in the output triangulation produced by the Adaptive Delaunay Refinement Algorithm, then

$$\angle pqr \leq \min \left\{ \pi - 2\kappa, \pi - 2 \arcsin \frac{\sqrt{3} - 1}{2} \right\}.$$
9. ADAPTIVE MIDPOINT SPLITTING

Our analysis so far has required that input meet Assumption 4.2. This assumption can be satisfied by first adding no more than $\frac{2}{j}$ augmenting points, effectively redefining the input. While this can be done while only suffering a constant increase in the cardinality of the final point set, this increase may be unacceptably large [8]. Ruppert’s original heuristic for dealing with midpoint-midpoint interactions can remove the additional restriction on input while still giving good point set sizes in practice.

Ruppert’s strategy of splitting on concentric circular shells [1] proceeds as follows: The first time an input segment is split, it is split by a point at its midpoint, creating two subsegments each with one input point associated. When one of these subsegments is split, it is split by a point $p$ closest to the midpoint of the subsegment such that $|p - x|$ is a power of two (in some global unit), where $x$ is the input point associated with the subsegment. All further subsegment splits are committed at midpoints.

We will refer to these first three points on any segment as “off-center” points, even though they could be at the midpoint of the involved subsegment. It is simple to show that $\text{lfs}_{\text{min}}$ is no greater than three times the length of the shortest subsegment created by an off-center split under this strategy. This follows since $\text{lfs}_{\text{min}}$ is no greater than half the length of any input segment, and the fact that the off-center split fact must occur in the middle third of the subsegment.

Then Theorem 7.1 can be reproven for the Adaptive Delaunay Refinement Algorithm with concentric shell splitting for arbitrary input satisfying Assumption 2.1. The basic strategy is that if any of the midpoints involved in the proof are actually off-center points, they can be shown to be not far away by Corollary 6.4, and then the Lipschitz property of local feature size suffices; in the end game none of the involved midpoints are off-center, and the input locally conforms to Assumption 4.2, so the previous arguments may be used.

For the analysis to be valid, it is necessary that the algorithm treat off-center points as input points, not as midpoints. This makes a difference because the adaptive variant of the Delaunay Refinement Algorithm regards triangles differently if the shortest edge has midpoints as endpoints.

In light of the discussion in Subsection 3.1, we can make the following

Claim 9.1. Suppose an input conforming to Assumption 2.1 if given to Ruppert’s Algorithm with concentric shell splitting. Then if $\kappa < 26.45^\circ \lrcorner \arctan ((\sin \theta)/\sqrt{2 - \cos \theta})$, the algorithm will terminate with no output angle smaller than $\kappa$.

10. RESULTS

Figure 5: The Baltic Sea input data. The input consists of 1401 points and 1301 line segments. There are a number of small angles and small segments present. The minimum angle, $\theta^*$ is approximately 0.052°.

The Adaptive Delaunay Refinement Algorithm with splitting on concentric shells was implemented. The code was tested on the Baltic Sea, as shown in Figure 5, with $k \approx \arcsin 2^{-7/6}$. The input has a number of small angles, the smallest being around 0.052°.

The output is shown in Figure 7, and is a mesh on 21704 vertices. The minimum and maximum angle histograms are shown in Figure 6. The minimum angle histogram shows that a small number of triangles have minimum angle less than 26.45°; these are all small input angles or “across” from small input angles, in accordance with Lemma 8.3.

References


The minimum- and maximum angle histograms are shown, respectively, in (a) and (b). In this figure triangles are counted, not angles, thus the total count is the number of triangles (in this case 43357), and not three times that number. In (a), those triangles with minimum angle smaller than $\kappa \approx 26.45^\circ$ are due to small input angles, in accordance with Lemma 8.3. The lack of large angles is guaranteed by Corollary 8.4.


Figure 7: The output mesh of the Baltic Sea input (Figure 5) with \( \hat{\kappa} \approx \arcsin 2^{-7/6} \) is shown.
A CRYSITLLINE, RED GREEN STRATEGY FOR MESHING HIGHLY DEFORMABLE OBJECTS WITH TETRAHEDRA

Neil Molino\textsuperscript{1} Robert Bridson\textsuperscript{1} Joseph Teran\textsuperscript{1} Ronald Fedkiw\textsuperscript{2}

\textsuperscript{1}Stanford University, Stanford, CA, U.S.A. \{npmolino, rbridson, jteran\}@stanford.edu
\textsuperscript{2}Stanford University, Stanford, CA, U.S.A. fedkiw@cs.stanford.edu

ABSTRACT

Motivated by Lagrangian simulation of elastic deformation, we propose a new tetrahedral mesh generation algorithm that produces both high quality elements and a mesh that is well conditioned for subsequent large deformations. We use a signed distance function defined on a Cartesian grid in order to represent the object geometry. After tiling space with a uniform lattice based on crystallography, we use the signed distance function or other user defined criteria to guide a red green mesh subdivision algorithm that results in a candidate mesh with the appropriate level of detail. Then, we carefully select the final topology so that the connectivity is suitable for large deformation and the mesh approximates the desired shape. Finally, we compress the mesh to tightly fit the object boundary using either masses and springs, the finite element method or an optimization approach to relax the positions of the nodes. The resulting mesh is well suited for simulation since it is highly structured, has robust topological connectivity in the face of large deformations, and is readily refined if deemed necessary during subsequent simulation.

Keywords: tetrahedral mesh generation, level set methods, BCC lattice, red green refinement hierarchy, large deformations, muscle simulation

1. INTRODUCTION

We are particularly interested in simulating highly deformable bodies such as the muscle and fatty tissues commonly encountered in biomechanics [1, 2], haptics [3], and virtual surgery [4, 5]. The quality of the tetrahedral mesh has a profound influence on both the accuracy and efficiency of these simulations, see e.g. [5]. Therefore, we propose a mesh generation algorithm designed specifically for such high deformation simulations.

Mesh generation is not only a broad field, but is in some sense many fields, each concerned with the creation of meshes that conform to quality measures specific to the application at hand. The requirements for fluid flow and heat transfer where the mesh is not deformed, and for small deformation solids where the mesh is barely deformed, can be quite different from those for simulating soft biological tissue that may undergo large deformations. Simple examples show that the specific requirements or measures of quality of a mesh vary depending on the problem being solved, see e.g. [6].

For example, an optimal mesh for an Eulerian fluid flow simulation should include anisotropically compressed elements in boundary layers, e.g. [7, 8, 9]. In these calculations, the solution gradient in the direction of the fluid flow is typically not as large as in the orthogonal directions. Obviously, it is desirable to have the density of the elements be larger in directions where the gradient is large and lower in directions where the gradient is small, i.e. elongated elements. In contrast, however, highly stretched cells tend to be ill-conditioned when a mesh deforms significantly as is typical for soft bodies. Either the mesh is softer in the thin direction and the cell has a tendency to invert, or
the mesh is stiffer in the thin direction and the simulation becomes very costly since the explicit time step restriction worsens with higher stiffness and smaller element cross-section. Thus, although our method has been designed to provide a high degree of adaptivity both to resolve the geometry and to guarantee quality simulation results, we neither consider nor desire anisotropically stretched elements. Also, since highly deformable bodies tend to be devoid of sharp features such as edges and corners, we do not consider boundary feature preservation.

Our main concern is to generate a mesh that will be robust when subsequently subject to large deformations. For example, although we obviously want an adaptive mesh with smaller elements in areas where more detail is desired, it is even more important to have a mesh that can be adapted during the simulation since these regions will change. Motivated by crystallography, we use a body-centered cubic (BCC) mesh (see e.g. [10]) that is highly structured and produces similar (in the precise geometric sense) tetrahedra under regular refinement. This allows us to adaptively refine both while generating the mesh and during the subsequent simulation.

We start with a uniform tiling of space and use a signed distance function representation of the geometry to guide the creation of the adaptive mesh, the deletion of elements that are not needed to represent the object of interest, and the compression of the mesh necessary to match the object boundaries [11]. This compression stage can be carried out using either a mass spring system, a finite element method or an optimization based approach. One advantage of using a physically based compression algorithm is that it gives an indication of how the mesh is likely to respond to the deformations it will experience during simulation. This is in contrast to many traditional methods that may produce an initial mesh with good quality measures, but also with possible hidden deficiencies that can be revealed during simulation leading to poor accuracy or element collapse. Moreover, our novel topological considerations (discussed below) are specifically designed to address these potential defects present in other mesh generation schemes.

2. RELATED WORK

While Delaunay techniques have been quite successful in two spatial dimensions, they have not been as successful in three spatial dimensions (see e.g. [12] for a discussion of implementation details). They admit flat sliver tetrahedra of negligible volume. Shewchuk provides a nice overview of these methods, including a discussion of why some of the theoretical results are not reassuring in practice [13]. Moreover, he discusses how the worst slivers can often be removed. Cheng et al. [14] also discuss sliver removal, but state that their theorem gives an estimate that is “miserably tiny”. Edelsbrunner and Guoy [15] showed that [14] can be used to remove most of the slivers, but is not as promising near boundaries. Another problem with Delaunay methods is that the Delaunay tetrahedralization of a set of points is convex whereas the domains of many finite element calculations are not. Thus, techniques such as the conforming Delaunay approach which inserts additional vertices into the mesh to force it to conform to the boundary of the domain must be developed. The constrained Delaunay tetrahedralization is another method used to enforce boundary recovery [16]. These approaches can be complicated and can even produce an intractably large mesh which is not polynomial in the complexity of the input domain.

Advancing front methods start with a boundary discretization and march a “front” inward forming new elements attached to the existing ones [17]. Advancing front techniques conform well to the boundary. This renders them a useful technique when the specific polygonal boundary representation of the geometry must be matched precisely, for example, when meshing a machine part. When the input geometry is not a polygonal boundary, a triangulation of this boundary must first be performed. The quality of this surface triangulation has a large impact on the three dimensional algorithm’s behavior. Poorly shaped surface triangles will engender ill-shaped tetrahedra [18]. A central decision in an advancing front algorithm is the placement of an interior point that marches the front further into the interior of the object. Local element control is possible because new nodes are created at the same time that new elements are created. The node and element creation is done as needed based on local procedures. Authors have experimented with various metrics and criteria to evaluate the placement of the new node, see e.g. [19, 20, 21]. All advancing front techniques have difficulty when fronts merge, however, which unfortunately can occur very near the important boundary in regions of high curvature [8, 9].

Radovitzky and Ortiz [22] started with a face-centered cubic (FCC) lattice defined on an octree and used an advancing front approach to march inward, constructing a mesh with the predetermined nodes of the FCC lattice. They chose FCC over BCC because it gives slightly better tetrahedra for their error bounds. However, after any significant deformation the two meshes will usually have similar character. Moreover, since we keep our BCC connectivity intact (as opposed to [22]), we retain the ability to further refine our BCC mesh during the calculation to obtain locally higher resolution for improved accuracy and robustness. On the other hand, their approach is better at resolving boundary features and is thus likely superior for problems with little to no deformation.
Fuchs [23] begins with a BCC tiling of space which is adaptively refined to obtain the desired nodal density. Vertices outside the object are simply projected to the boundary and then smoothing is applied to optimize the position of the vertices. They emphasize that the BCC connectivity is never used and instead apply Delaunay tessellation. That is, they only use the adaptive BCC lattice to obtain an initial guess for their vertex positions.

Shimada and Gossard [24] packed spheres (or ellipsoids for anisotropic mesh generation [25]) into the domain with mutual attraction and repulsion forces, and generated tetrahedra using the sphere centers as sample points via either a Delaunay or advancing front method. However, ad hoc addition and deletion of spheres is required in a search for a steady state, and both local minima and “popping” can be problematic. This led Li et al. [26] to propose the removal of the dynamics from the packing process, instead marching in from the boundary removing spherical “bites” of volume one at a time. This biting was motivated by the advancing front technique, but used here for sphere packing rather than mesh generation. The final mesh is computed with a Delaunay algorithm on the sphere centers. Later, they extended the biting idea to ellipsoids to generate anisotropic meshes [27].

Our compression phase moves the nodes on the boundary of our candidate mesh to the implicit surface, providing boundary conformity. In some sense, this wrapping of our boundary around the level set is related to snakes [28] or GDMs [29] which have been used to triangulate isosurfaces, see e.g. [30]. Neugebauer and Klein started with a marching cubes mesh and moved vertices to the centroid of their neighbors before projecting them onto the zero level set in the direction of mesh quality. When we move our mesh normal to the surface, it directly conflicts with the quality of the surface tetrahedra. In [36], de Figueiredo et al. evolved a volumetric mass spring system in order to align it with (but not compress it to) the zero isocontour, but the measure of mesh quality was still perpendicular to the evolution direction since the goal was to triangulate the zero isocontour. Later, however, Velho et al. did push in a direction conflicting with mesh quality. They deformed a uniform-resolution Freudenthal lattice to obtain tetrahedralizations using a mass spring model, but were restricted to simple geometries, mostly due to the inability to incorporate adaptivity [37].

In two spatial dimensions, Gloth and Vilsmeier also moved the mesh in a direction that opposed the element quality [38]. They started with a uniform Cartesian grid bisected into triangles, threw out elements that intersected or were outside the domain, and moved nodes to the boundary in the direction of the gradient of the level set function using traditional smoothing, edge swapping, insertion and deletion techniques on the mesh as it deformed.

### 3. THE BCC LATTICE

We turn our attention to the physical world for inspiration and start our meshing process with a body-centered cubic (BCC) tetrahedral lattice. This mesh has numerous desirable properties and is an actual crystal structure ubiquitous in nature, appearing in vastly different materials such as soft lithium and hard iron crystals, see e.g. [10]. Other spatial tilings are possible. Üngör [39] provides a number of these including tilings using acute tetrahedra.

The BCC lattice consists of nodes at every point of a Cartesian grid along with the cell centers. These node locations may be viewed as belonging to two interlaced grids. Additional edge connections are made between a node and its eight nearest neighbors in the other grid. See figure 1 where these connections are depicted in red and the two interlaced grids are depicted in blue and in green. The BCC lattice is the Delaunay complex of the interlaced grid nodes, and thus possesses all properties of a Delaunay tetrahedralization. Moreover, all the nodes are isomorphic to each other (and in particular have uniform valence), every tetrahedron is congruent to the others, and the mesh is isotropic (so the mesh itself will not erroneously induce any anisotropic bias into a subsequent calculation). The BCC lattice is structured, which may be exploited in preconditioned iterative solvers, multigrid algorithms, etc., and may allow reduced computational and memory requirements.

Although we derive motivation from this work, we note that our problem is significantly more difficult since these authors move their mesh in a direction normal to the surface, which is orthogonal to their measure of mesh quality (shapes of triangles tangent to the surface). When we move our mesh normal to the surface, it directly conflicts with the quality of the surface tetrahedra.
regular BCC tetrahedron can be refined into eight tetrahedra, shown in red in figure 2, with a one to eight (or 1:8) refinement. When the shortest of the three possible choices for the edge internal to the tetrahedron is taken, the newly formed tetrahedra are exactly the BCC tetrahedra that result from a mesh with cells one half the size. Thus, these eight new tetrahedra are geometrically similar to the tetrahedra of the parent mesh and element quality is guaranteed under this regular 1:8 refinement.

4. A RED GREEN HIERARCHY

Many applications do not require and cannot afford (due to computation time and memory restrictions) a uniformly high resolution mesh. For example, many phenomena such as contact and fracture show highly concentrated stress patterns, often near high surface curvature, outside of which larger tetrahedra are acceptable. In addition, many applications such as virtual surgery can tolerate lower accuracy in the unseen interior of a body. Thus, we require the ability to generate adaptive meshes.

As the BCC lattice is built from cubes, one natural approach to adaptivity is to build its analog based on an octree. We implemented this by adding body centers to the octree leaves, after ensuring the octree was graded with no adjacent cells differing by more than one level. The resulting BCC lattices at different scales were then patched together with special case tetrahedra. For more on octrees in mesh generation, see e.g. [40, 41, 22] (none of which use our multilevel BCC mesh).

However, we found that red green refinement is more economical, simpler to implement, and more flexible, see e.g. [42, 43, 44]. The initial BCC lattice tetrahedra are labelled red, as are any of their eight children obtained with 1:8 subdivision. Performing a red refinement on a tetrahedron creates T-junctions at the newly-created edge midpoints where neighboring tetrahedra are not refined to the same level. To eliminate these, the red tetrahedra with T-junctions are irregularly refined into fewer than eight children by introducing some of the midpoints. These children are labeled green, and are of lower quality than the red tetrahedra that are part of the BCC mesh. Moreover, since they are not BCC tetrahedra, we never refine them. When higher resolution is desired in a region occupied by a green tetrahedron, the entire family of green tetrahedra is removed from its red parent, and the red parent is refined regularly to obtain eight red children that can undergo subsequent refinement.

A red tetrahedron that needs a green refinement can have between one and five midpoints on its edges (in the case of six we do red refinement). We reduce the possibilities for green refinement to those shown in figure 2, adding extra edge midpoints if necessary. This restriction (where all triangles are either bisected or quadrisected) smooths the gradation further and guarantees higher quality green tetrahedra. While there can, of course, be a cascading effect as the extra midpoints may induce more red or green refinements, it is a small price to pay for the superior mesh quality and seems to be a minor issue in practice.

Any criteria may be used to drive refinement, and we experimented with the geometric rules described in the next section. A significant advantage of the red green framework is the possibility for refinement during simulation based on a posteriori error estimates, with superior quality guarantees based on the BCC lattice instead of an arbitrary initial mesh. Note that the lower quality green tetrahedra can be replaced by finer red tetrahedra which admit further refinement. However, one difficulty we foresee is in discarding portions of green families near the boundary (see section 6), since part of the red parent is missing. To further refine this tetrahedron, the green family has to be replaced with its red parent which can be regularly refined, then some of the red children need to be discarded and the others must be compressed to the boundary (see sections 7–8). A simpler but lower quality alternative is to arbitrarily relabel those green boundary tetrahedra that are missing siblings as red, allowing them to be directly refined. We plan to address this issue in future work.
5. LEVEL SET GEOMETRY

We represent the geometry with a signed distance function defined on either a uniform grid [45] or an octree grid [46, 47]. In the octree case, we constrain values of fine grid nodes at gradation boundaries to match the coarse grid interpolated values, see e.g. [48]. When the signed distance function has a resolution much higher than that of our desired tetrahedral mesh, we apply motion by mean curvature to smooth the high frequency features and then reinitialize to a signed distance function, see e.g. [45].

Medical data such as the National Library of Medicine’s Visible Human data set often comes in the form of volume data [49]. Thus, it is natural to devise a mesh generation technique that generates a volumetric mesh from this data. The data is first converted into a level set using straightforward and efficient algorithms such as a fast marching method [50, 51]. Level sets arise naturally in other applications as well. They are used as a design primitive in CAGD packages. They are also used as a technique to generate a surface from scattered point data [52].

At any point in space, we calculate the distance from the implicitly defined surface as \( \phi \), which is negative inside and positive outside the surface. To obtain a finer mesh near the boundary, one simply refines tetrahedra that include portions of the interface where \( \phi = 0 \). If a tetrahedron has nodes with positive values of \( \phi \) and nodes with negative values of \( \phi \), it obviously contains the interface and can be refined. Otherwise, the tetrahedron is guaranteed not to intersect the interface if the minimum value of \( |\phi| \) at a node is larger than the longest edge length (tighter estimates are available, of course). The remaining cases are checked by sampling \( \phi \) appropriately (at the level set grid size \( \Delta x \)), allowing refinement if any sample is close enough to the interface \( (|\phi| < \Delta x) \). Figure 3 shows a sphere adaptively refined near its boundary. Note how the interior mesh can still be rather coarse.

6. TOPOLOGICAL CONSIDERATIONS

To obtain the final topology of the mesh, we first cover an appropriately sized bounding box of the object with a coarse BCC mesh. Then we use a conservative discard process to remove tetrahedra that are guaranteed to lie completely outside of the zero isocontour: tetrahedra with four positive \( \phi \) values all larger than the maximum edge length are removed.

In the next step, the remaining tetrahedra are refined according to any user defined criteria, such as indicator variables or geometric properties. We have experimented with using both the magnitude of \( \phi \) and various measures of curvature as discussed in the previous section. Using simply the magnitude of \( \phi \) produces large tetrahedra deep inside the object and a uniform level of refinement around the surface, which can be useful since objects interact with each other via surface tetrahedra. A more sophisticated method uses the surface principal curvatures, better resolving complex geometry and allowing for more robust and efficient simulation when subject to large deformation.
We refine any tetrahedron near the interface if its maximum edge length is too large compared to a radius of curvature measure, $1/(|k_1| + |k_2|)$, indicating an inability to resolve the local geometry. We refine to a user-specified number of levels, resolving T-junctions in the red green framework as needed.

From the adaptively refined lattice we select a subset of tetrahedra that closely matches the object. However, there are specific topological requirements necessary to ensure a valid mesh that behaves well under deformation: the boundary must be a manifold; no tetrahedron may have all four nodes on the boundary; and no interior edge may connect two boundary nodes. Boundary forces can readily crush tetrahedra with all nodes on the boundary, or that are trapped between the boundary and an interior edge with both endpoints on the boundary. To satisfy the conditions, we select all the tetrahedra incident on a set of “enveloped” nodes sufficiently interior to the zero isocontour. This guarantees that every tetrahedron is incident on at least one interior node, and also tends to avoid the bad interior segments for reasonably convex regions, i.e. regions where the geometry is adequately resolved by the nodal samples. We specifically choose the set of nodes where $\phi < 0$ that have all their incident edges at least 25% inside the zero isocontour as determined by linear interpolation of $\phi$ along the edge.

Additional processing is used to guarantee appropriate topology even in regions where the mesh may be under-resolved. Any remaining interior edges and all edges incident on non-manifold nodes are bisected, and the red green procedure is used to remove all T-junctions. If any refinement is necessary, we recalculate the set of enveloped nodes and their incident tetrahedra as above. As an option, we may add any boundary node with surface degree three to the set of enveloped nodes (if these nodes were to remain, the final surface mesh would typically contain angles over 120°). We also add any non-manifold node that remains and the deeper of the two boundary nodes connected by a bad interior edge. We check that these additions do not create more problems, continuing to add boundary nodes to the set of enveloped nodes until we have achieved all requirements. This quickly and effectively results in a mesh that approximates the object fairly closely (from the viewpoint of an initial guess for the compression phase of the algorithm) and that has connectivity well suited for large deformation simulations.

7. PHYSICS BASED COMPRESSION

We outfit our candidate mesh with a deformable model based on either masses and springs or the finite element method, and subsequently compress the boundary nodes to conform to the zero isocontour of the signed distance function. The compression is driven using either a force or velocity boundary condition on the surface nodes. Applying forces is more robust as it allows the interior mesh to push back, resisting excessive compression while it seeks an optimal state. However, if the internal resistance of the mesh becomes larger than the boundary forces, the boundary will not be matched exactly. Thus, instead of adjusting forces, we switch from force to velocity boundary conditions after an initial stage that carries out most of the needed compression. At each boundary vertex, we choose the direction of the force or constrained velocity component as the average of the incident triangles’ normals. No force (or velocity constraint) is applied in other directions so the mesh is free to adjust itself tangentially. The magnitude of the force or velocity constraint is proportional to the signed distance from the level set boundary.

To integrate the equations of motion forward in time, we use a central difference scheme that treats the nonlinear elastic forces explicitly and the damping forces implicitly. This circumvents stringent time step restrictions based on the damping forces. Moreover, since all our damping forces are linear and symmetric negative semi-definite, we can use a conjugate gradient solver for the implicit step. We use a velocity modification procedure to artificially limit the maximum strain of a tetrahedral altitude to 50%, and to artificially limit the strain rate of a tetrahedral altitude to 10% per time step [54]. Since altitudes do not connect two mesh nodes together, all of these operations are carried out by constructing a virtual node at the intersection point between an altitude and the plane containing the base triangle. The velocity of this point is calculated using the barycentric coordinates and velocities of the triangle, and the mass is the sum of the triangle’s nodal masses. The resulting impulses on this virtual node are then redistributed to the triangle nodes, conserving momentum.

7.1 Mass Spring Models

The use of springs to aid in mesh generation dates back at least to Gheofo, who used them to move nodes for two dimensional fluid dynamics calculations [55, 56]. Löhner et al. solved the compressible Euler equations using variable spring stiffnesses to distribute the error evenly over the solution domain [57]. Later, [58] used variational principles analogous to the energy of a system of springs to achieve the same goal. Other authors also measured the error of a CFD calculation along edges of a mesh and then used a spring network to equidistribute these errors over the edges [59, 60, 61]. Bossen and Heckbert point out that inter-nodal forces that both attract and repel (like springs with nonzero rest lengths) are superior to Laplacian smoothing where the nodes only attract each other
We use the nonlinear Green strain tensor,\(^{1}\) which provides an accurate representation of deformation while constructing the mesh. Various methods to prevent element collapse during simulation, and possibly work to ensure simulation robustness while constructing the mesh.

Edge springs are not sufficient to prevent element collapse. As a tetrahedron gets flatter, the edge springs provide even less resistance to collapse. Various methods to prevent this have been introduced, e.g.\(^{63}\) proposed a pseudo-pressure term,\(^{64}\) used an elastic (only, i.e. no damping) force emanating from the barycenter of the tetrahedron.\(^{65}\) showed that these barycentric springs do not prevent collapse as effectively as altitude springs. In our model, every tetrahedron has four altitude springs each attaching a tetrahedron node to a fictitious node on the plane of its opposite face. Then, the elastic and damping forces are calculated just as for a normal spring. These forces are distributed among the three nodes on the opposite face according to the barycentric weights of the fictitious node. This model has damping forces that are linear and symmetric negative semi-definite in the nodal velocities allowing the damping terms to be integrated using a fast conjugate gradient solver for implicit integration.

When simulating a deformable object with a mass spring network, the material behavior should be independent of mesh refinement. The frequency of a spring scales as \(\sqrt{k/m}\), where our “spring constant” is \(k/l_0\), so the sound speed scales as \(l_0\sqrt{k/m}\). Requiring the sound speed to be a material property implies that \(k\) must scale as \(m/l_0\). Thus, we set the spring stiffness for an edge spring using the harmonic average of the masses of the two nodes at the ends of the spring and its restlength. Similarly, for altitude springs we use the harmonic average of the nodal mass and the triangle mass.

### 7.2 Finite Element Method

While any number of constitutive models could be used, an interesting strategy is to use the real constitutive model of the material when generating its mesh. In this sense, one might hope to predict how well the mesh will react to subsequent deformation during simulation, and possibly work to ensure simulation robustness while constructing the mesh.

We use the nonlinear Green strain tensor, \(G = \frac{1}{2}(\partial x/\partial u)^T (\partial x/\partial u) - I\), where \(x(u)\) represents a point’s position in world coordinates as a function of its coordinates in object space. Isotropic, linearly-elastic materials have a stress strain relationship of the form \(S_e = \lambda \varepsilon + 2\mu \varepsilon\) where \(\lambda\) and \(\mu\) are the Lamé coefficients. Damping stress is modeled similarly with \(S_d = \alpha \dot{\varepsilon}\) where \(\alpha = 2\mu/\partial t\) is the strain rate. The total stress tensor is then \(S = S_e + S_d\).

We use linear basis functions in each tetrahedron so that the displacement of material is a linear function of the tetrahedron’s four nodes. From the nodal locations and velocities we obtain this linear mapping and its derivative and use them to compute the strain and the strain rate, which in turn are used to compute the stress tensor. Finally, because the stress tensor encodes the force distribution inside the material, we can use it to calculate the force on the nodes.

In their finite element simulation,\(^{66}\) added a force in the same direction as our altitude springs. Since that force was the same on all nodes and based on the volume deviation from the rest state, it does not adversely penalize overly compressed directions and can even exacerbate the collapse. Instead, we artificially damp the strain and strain rate of the altitudes of the tetrahedra as discussed above.

### 8. OPTIMIZATION BASED COMPRESSION

As an alternative to physical simulation, one can directly optimize mesh quality metrics such as aspect ratios. This does not provide the same feedback on potential problems for subsequent simulation, but can give better quality measures since they are directly pursued with each movement of a node. Coupled with our robust connectivity (see section 6), this produces excellent results. Freitag and Olliver-Gooch\(^{67}\) demonstrated that optimizing node positions in a smoothing sweep, i.e. placing one node at a time at a location that maximizes the quality of incident elements, is superior to Laplacian smoothing in three spatial dimensions. We combine this optimization sweeping with boundary constraints by first moving boundary nodes in the incident triangles’ average normal direction by an amount proportional to the local signed distance value. Then the optimization is constrained to only move boundary nodes in the tangential direction.

It is important to move boundary nodes gradually over several sweeps just as with physical simulation, since otherwise the optimization gets stuck in local extrema. We also found it helpful to order the nodes in the sweep with the boundary nodes first, their interior neighbors next, and so on into the interior. Then we sweep in the reverse order and repeat. This efficiently transfers information from the boundary compression to the rest of the mesh. Typically, we do five sweeps of moving the boundary nodes 1/3 of the signed distance in the mesh normal direction, then finish off with five to ten sweeps moving boundary nodes the full signed distance to ensure a tight boundary fit. To speed up the sweeps, we do not bother moving nodes that are incident on tetrahedra of sufficiently high quality relative to the worst tetrahedron currently in the mesh. In the initial
sweeps we end up only optimizing roughly 10% of the nodes, and in the final sweeps we optimize 30%-50% of the nodes.

While more efficient gradient methods may be used for the nodal optimization, we found a simple pattern search (see e.g. [68]) to be attractive for its robustness, simplicity of implementation, and flexibility in easily accommodating any quality metric. For interior nodes we used seven well spread-out directions in the pattern search. We implemented the normal direction constraint on boundary nodes simply by choosing five equally spaced pattern directions orthogonal to the average mesh normal at the node. The initial step size of the pattern search was .05 times the minimum distance to the opposite triangle in any tetrahedron incident on the node (to avoid wasting time on steps that crush elements). After four “strikes” (searches at a given step size that yielded no improvement in quality, causing the step size to be halved) we move to the next node. For interior nodes we use as a quality metric the minimum of \( \frac{a}{L} + \frac{1}{4} \cos(\theta_M) \) over the incident tetrahedra, where \( a \) is the minimum altitude length, \( L \) is the maximum edge length, and \( \theta_M \) is the maximum angle between face normals. We found that including the extra terms beyond the tetrahedron aspect ratios helped guide the optimization out of local minima and actually resulted in better aspect ratios.

9. RESULTS

We demonstrate several examples of tetrahedral meshes that were generated with our algorithm. The results for all three compression techniques are comparable, with the FEM simulations taking slightly longer (ranging from a few minutes to a few hours on the largest meshes) than the mass spring methods, but producing a slightly higher quality mesh. For example, the maximum aspect ratio of a tetrahedron in the cranium generated with finite elements is 6.5, whereas the same mesh has a maximum aspect ratio of 6.6 when the final compression is done using a mass spring model. Mass spring networks have a long tradition in mesh generation, but a finite element approach offers greater flexibility and robustness that we anticipate will allow better three-dimensional mesh generation in the future. Currently the fastest method is the optimization based compression, roughly faster by a factor of ten.

We track a number of quality measures including the maximum aspect ratio (defined as the tetrahedron’s maximum edge length divided by its minimum altitude), minimum dihedral angle, and maximum dihedral angle during the compression phase. The maximum aspect ratios of our candidate mesh start at about 3.5 regardless of the degree of adaptivity, emphasizing the desirability of our combined red green adaptive BCC approach. This number comes from the green tetrahedra (the red tetrahedra have aspect ratios of \( \sqrt{2} \)). In the more complicated models, the worst aspect ratio in the mesh tends to increase to around 6-8 for the physics based compression methods and to around 5-6 for the optimization based compression.

For the cranium model, the physics based compression methods gave a maximum aspect ratio of 6.5 and aver-
age aspect ratio of 2.1, with dihedral angles bounded between 17° and 147°. The dragon mesh has a maximum aspect ratio of 7.6 and an average aspect ratio of 2.2, with dihedral angles bounded between 13° and 154°. The buddha model was more challenging, giving a worst aspect ratio of 8.1 and average of 2.3, and dihedral angles between 13° and 156°. Using optimization on the same examples yielded better results, listed in table 1, where we have also listed a measure of adaptivity, the ratio of the longest edge in the mesh to the shortest. The aspect ratios all drop below 6, i.e. less than twice the initial values.

Of course, these results are dependent on the types and strengths of springs, the constitutive model used in the FEM, and the quality measures used in the optimization based technique. It is easier to achieve good quality with the optimization technique since one simply optimizes based on the desired measure, as opposed to the physics based techniques where one has to choose parameters that indirectly lead to a quality mesh. However, we stress that the measure of mesh quality is the measure of the worst element at any point of dynamic simulation. It does little good to have a perfect mesh that collapses immediately when the simulation begins. For meshes that undergo little to no deformation (fluid flow, heat flow, small strain, etc.) this quality measure is either identical to or very close to that of the initial mesh. However, for large deformation problems this is not the case, and the physics based compression techniques hold promise in the sense that the resulting mesh may be better conditioned for simulation. We believe an interesting possibility for the future would be to consider hybrid approaches that use the physics based compression algorithms to guide an optimization procedure to avoid local minima.

10. EXAMPLE: MUSCLE SIMULATION

Musculoskeletal simulation is an active research area in biomechanics. We demonstrate the robustness of our meshing algorithm by simulating volumetric, deformable skeletal muscle. Our meshing algorithm allows us to create high resolution muscle, tendon and bone geometries from the Visible Human data set [49]. The data for these biological materials are originally in the form of a segmented series of consecutive images that can be used to create a level set description of each tissue geometry. This level set can then be used with either the dynamic or optimization based algorithm. Figure 8 shows an adaptive resolution biceps with tendon that was created using dynamic meshing with a finite element constitutive model.
change; the second term represents the isotropic embedding matrix; and the third term is the transversely-isotropic component that models muscle fiber contraction and is based on the standard muscle force/length curve [72]. This model can be used in both muscle and tendon, however, tendon tends to be as much as an order of magnitude stiffer and muscle has an additional contractile force added to the fiber component that depends on the muscle activation level.

In addition to activation level, muscle (and tendon) models need information about the local fiber direction. Muscle fiber arrangements vary in complexity from being relatively parallel and uniform to exhibiting several distinct regions of fiber directions. We use a B-spline solid as in [73, 74] to represent more intricate muscle fiber architectures and to assign a fiber direction to individual tetrahedra in the mesh. During both isometric and isotonic contraction, muscles are given a varying activation level throughout the simulation. The activation levels are computed from keyframes of the skeletal animation, using an established biomechanics analysis known as muscle force distribution [75] to compute activations of redundant sets of muscles.

Figure 9: Simulation of isometric contraction. A posterior (from behind) view of the upper arm shows contraction of the triceps muscle and the partially occluded biceps muscle from passive (left) to full activation (right).

Figures 9 and 10 show sample frames of our musculoskeletal simulations. Figure 9 depicts relaxed and active muscle during isometric contraction. In this simulation the activation level in the two muscles increases from 0 (fully relaxed) to 1 (fully activated) and back to 0 over the span of two seconds. The bulging in the bellies of the muscles results from larger stiffness in the tendons. Figure 10 shows several frames of musculoskeletal motion. The motion of the kinematic skeleton was key-framed (although our framework allows for motion data from other sources like motion capture). At each key-frame in the animation, an inverse dynamics analysis was computed for the biceps and triceps activation levels required to maintain the static pose. These activation levels were then interpolated in time and used for the dynamic muscle simulation.

Figure 11 shows the relative change in maximum aspect ratio observed during an isometric contraction of the biceps for meshes created using the optimization algorithm and using the dynamics algorithm. Similar results were observed for the triceps and during isotonic contraction. These results suggest that initial mesh quality may be misleading and not sufficient to guarantee performance of a mesh throughout simulation. In all of our comparisons, the optimization based meshes were of higher quality initially, but tended to undergo as much as a 70% change in maximum aspect ratio during muscle contraction, whereas the dynamics based meshes tended to degrade by only 25%. Of course, if the initial optimization mesh is of significantly higher quality then the overall maximum aspect ratio will still be lower. We are not yet claiming that a particular method is better, but simply pointing out that the initial mesh quality is not always a reliable predictor of performance during subsequent simulation.

11. CONCLUSIONS

We presented an algorithm for producing a high quality tetrahedral mesh directly from a level set. The focus of this algorithm is the generation of a tetrahe-
Research Fellowship. and J. T. was supported in part by an NSF Graduate Fellowship, ONR N00014-03-1-0071, ONR N00014-02-1-0720, NSF ITR-0121288, NSF ACI-0205671 and NSF Packard Foundation Fellowship, a Sloan Research Fellowship.

Although initial mesh quality measures are important, they do not guarantee high quality during subsequent simulation even begins (in fact this is what originally led us to the problematic connectivity). Finally, we simulated a few muscles from the NIH Visible Human data set to demonstrate the efficacy of these meshes. In particular, we illustrated that although initial mesh quality measures are important, they do not guarantee high quality during subsequent simulation, and can in fact be misleading.

12. ACKNOWLEDGEMENTS

Research supported in part by an ONR YIP award and PECASE award (ONR N00014-01-1-0620), a Packard Foundation Fellowship, a Sloan Research Fellowship, ONR N00014-03-1-0071, ONR N00014-02-1-0720, NSF ITR-0121288, NSF ACI-0205671 and NSF DMS-0106994. In addition, N. M. and R. B. were supported in part by a Stanford Graduate Fellowships, and J. T. was supported in part by an NSF Graduate Research Fellowship.

References


Invited Speaker
Abstract:

Generating Sliver-Free Well-Shaped Three Dimensional Delaunay Meshes

A mesh is cell-complex that decomposes a spatial domain for numerical simulation. Delaunay triangulations have many desirable properties for mesh generation. While there are several efficient methods for well-shaped 2D mesh generation, the generation of Delaunay meshes of well-shaped tetrahedra in 3D is considerably more difficult and has been an outstanding open problem for many years.

Most notably, slivers are notoriously common in three dimensional Delaunay meshes, where a sliver is a tetrahedron that has no short edge and whose four vertices lie closely to a great circle of its circum-sphere.

In this talk, I will survey the algorithmic and geometric techniques using weighted Delaunay triangulations and perturbations, that are recently developed for sliver removal. In particular, I will present the first Delaunay refinement algorithm, developed by Li and Teng, that always generates sliver free well-shaped unstructured meshes in three dimensions. The main ingredient of this algorithm is a novel refinement technique which systematically forbids the formation of slivers.

This talk contains collaborative works with Xiang-Yang Li, Siu-Wing Cheng, Tamal Dey, Herbert Edelsbrunner, Micheal Facello, Alper Ungor, Gary Miller, Dafna Talmor, and Noel Walkington.
Plenary 1
Session
This paper proposes a method for predicting the complexity of meshing Computer Aided Design (CAD) geometries with unstructured, hexahedral, finite elements. Meshing complexity refers to the relative level of effort required to generate a valid finite element mesh on a given CAD geometry. A function is proposed to approximate the meshing complexity for single part CAD models. The function is dependent on a user defined element size as well as on data extracted from the geometry and topology of the CAD part. Several geometry and topology measures are proposed which both characterize the shape of the CAD part and detect configurations that complicate mesh generation. Based on a test suite of CAD models the function is demonstrated to be accurate within a certain range of error. The solution proposed here is intended to provide managers and users of meshing software a method of predicting the difficulty in meshing a CAD model. This will enable them to make decisions about model simplification and analysis approaches prior to mesh generation.

Keywords: time to mesh, meshing complexity, blend detection, geometry clean-up
methods for twenty-four CAD geometries. While little effort has been directly aimed at quantifying meshing complexity in the literature, a great deal of research has been performed to identify and remove geometry and topology errors in CAD models, for the purpose of mesh generation. This process is commonly referred to as model clean-up. The obvious relationship between model clean-up and meshing complexity is that geometry and topology errors often lead to increased model complexity and time spent on the meshing process. Various approaches for model clean-up are reviewed. Additionally, approaches and techniques for generating unstructured hexahedra are also reviewed.

1.1 Model Clean-up

From a meshing perspective, the CAD model can typically have two kinds of problems: definition errors and representation problems. Both typically cause more time to be spent in the meshing process. Definition errors are those that deal with how the model is defined, both in geometry and topology. Representation problems can be more subjective and are dependent on how the model will be used.

1.1.1 CAD Model Repair

CAD Model Repair is defined as the process of fixing geometric and topological definition errors in a design model. CAD models are represented by one of two methods: boundary-representation or B-rep, and constructive solid geometry or CSG. Currently, most commercial vendors use the B-rep. B-rep models are represented by mathematical descriptions and lower-order boundary topology. For example, an edge is defined by a mathematical curve and is bounded by two vertices at the ends. The curve may be defined by a mathematical description including a simple line-segment, arc, or a more complicated B-spline. Similarly, a mathematical surface, and boundary edges define a face. Volumes are defined by a series of connected faces that wholly enclose a specific region.

CAD Model Repair typically involves fixing the mathematical curve and surface definitions. Work in this area has focused on detecting errors within the CAD model, either directly in the native format (in the software it was created) or in a third party software package. There are many different errors which can be detected. These include: inverted faces, gaps between surfaces in a volume, folded geometry, surface geometry with no bounding face, faces with no finite area, self-intersecting edges and faces, face/edge sloppiness, boundary edges that do not lie on the faces, overlapping faces, etc. [1,2,3]. There are several vendors that offer packages that detect and fix these problems. Some of these include: ACIS 3D Toolkit [4], Parasolid BodyShop [5], and CADfix [6]. Another tool called CADIQ [7], connects directly with the major vendors of CAD software and interrogates models for errors. In environments where this package/software is used, the cost savings to analysts could be large since the majority of the time spent generating a mesh is used in iterating with the design engineer on a “good” base model [2].

CAD Model Repair often appears unrelated to meshing since it is generally assumed at the meshing stage that the CAD models used for simulations are valid. Problems with the validity of the model are assumed to be taken care of upstream, or in the design stage. Unfortunately, designers have no incentive to validate their models for use in mesh generation since such models are typically created for visualization or manufacturing where model quality is less important [8]. In recent years, more meshing related software has been developed to fix these problems or build meshing algorithms that are less sensitive to geometry and topology definition errors [9,10]. In either case, it has been clearly demonstrated that CAD Model Repair is part of the overall process in going from design to analysis (D2A) and should be counted as part of any system that evaluates model complexity with respect to meshing.

1.1.2 Model Simplification

Model Simplification involves steps that are taken to detect and alter representation problems of a solid model to make mesh generation easier or possible. CAD models are typically built to accurately capture the detail of the real problem; a method which presents practical issues for discrete, numerical simulations. For example, Figure 1 shows a part that has a filleted section “colliding” with the boundary of another face, producing a tangential intersection of the boundary edges. This tangential intersection is difficult to mesh at any realistic element size; but it is especially hard to mesh with quadrilateral elements. This tangency is not a result of how the CAD model is defined, but rather how it is represented with respect to its proposed use. Model simplification is the process of detecting and removing such representation problems that make mesh generation difficult.

![Figure 1 Geometry Problem Due to Fillet](image)

New approaches for simplifying CAD models for mesh generation have recently been proposed and successfully implemented [11,12,13,14,15,16]. These methods scan the model and search for specific preprogrammed problems, such as stray vertices, sliver surfaces, or small fillets.
When the problems are found, they are either fixed in the native geometry subsystem or with a system-independent method. Sheffer et al. [11] first referred to this system-independent clean-up method as “Virtual Topology”. Virtual topology provides a way to change the topological representation of the model by layering modifications on the model without changing the model itself. Many of the approaches mentioned [12,13,14,15,16] automate the removal of preprogrammed problems; however, this process is often subjective. It is, therefore, difficult to predict which problems can be removed to make meshing easier and which problems the mesh generators must handle for the purposes of analysis. For example, a fillet in a CAD model may be included to dissipate stress concentrations at a critical region or it may be merely cosmetic.

1.2 Unstructured Hexahedral Mesh Generation

The study of meshing complexity is restricted here to generating unstructured hexahedra. Despite numerous efforts [17,18,19,20,21,22,23,24,25,26], there remains an absence of a satisfactory high quality automatic hexahedral meshing scheme. Instead, many researchers have attempted to improve the manual methods of generating hexahedral elements, namely sweeping and mapping. Some of these improvements include automating the traditional approaches by extending the sweeping and primitive algorithms and developing new automation control algorithms.

Mesh primitives are a set of pre-packaged meshes for typical or common shapes like squares, triangles, and circles in two dimensions and cubes, tetrahedrons, and spheres in three dimensions. Sweeping is essentially an extension of a cylinder primitive where the top circular surface mesh is extruded through the volume into hexahedrons. Sweeping requires that the “linking” surfaces or sidewalls of the sweep axis be meshed with a structured or regular meshing scheme like mapping [27].

In a typical manual approach, a user will decompose a part into pieces that can be meshed with either a primitive or sweeping algorithm. For instance, the model shown in Figure 2 (a) is not simply sweepable along the dominant axis of the part due to the protruding smaller cylinder on the side. In order to mesh this part, a sweep path must be cut through the larger material in order to sweep the side cylinder as shown in Figure 2 (b). Additionally, the user must be careful while creating this sweep path not to interfere with the interior of the part. The final decomposition and mesh of this model is shown in Figure 2 (c).

Primitives and sweeping often rely on user intervention to prescribe exact boundary intervals, surface meshes, and sweep directions. When meshing large assemblies of parts, managing and entering this data can become overwhelming, even for experienced users. Automation for controlling and relaxing the amount of user-supplied data has been another area of research in hexahedral meshing. Two algorithms that have substantially reduced this problem are automatic scheme selection and automatic interval assignment. Automatic scheme selection uses a sweepability proof to detect shapes that can be meshed with sweeping and other primitives [28]. The algorithm automatically assigns the proper surface schemes and determines proper sweep directions. The automatic interval assignment algorithm solves a system of linear, integer constraint equations to provide proper edge intervals for meshing [29]. The constraints are based specifically on the requirements of the meshing algorithms that are to be used.

An approach to estimating the meshing complexity of various CAD geometries is presented in this paper. The resulting information is intended to provide managers and users of meshing software a method of predicting how difficult a CAD model will be to mesh, enabling them to make decisions about model simplification and analysis approaches prior to meshing. The developments here pertain to individual parts in an assembly. Further developments would necessarily include the consideration of complexity of the entire assembly.
2 MESHING COMPLEXITY

Meshing complexity is a measure of the level of difficulty encountered in meshing a CAD geometry. Several aspects are known to make mesh generation difficult including near-tangencies, topology arrangement, etc. Meshing complexity involves the translation of these difficulties into a metric that quantifies them. For instance, there are many geometries that have extremely geometrically complex curves and surfaces but are meshed trivially. Likewise, one could easily construct a model using linear curves and planar surfaces that is nearly impossible to mesh. The problem posed by meshing complexity is finding what actually makes meshing difficult. Since many of these factors may be non-quantifiable, a reproducible and all-encompassing solution may be intractable. Instead, a solution is proposed that focuses on a subset of quantifiable issues to develop a useful measure for mesh complexity.

2.1 Variables of Meshing Complexity

Finding a single metric that accurately captures the meshing complexity of a solid is a complex problem. The problem can be viewed as determining a function that evaluates the complexity of a solid with respect to meshing. In developing such a function, the following variables may be considered: Element Type, tet vs. hex; Element Structure, structured vs. unstructured; Element Size, coarse vs. fine; Topology; Geometry; Assembly Configuration, multiple solids; Finite Element Analysis (FEA) Application, e.g. boundary layers; User Expertise; Meshing Software Maturity; and Choice of Meshing Algorithm. To reduce the scope of the problem, Meshing Software Maturity and User Expertise should naturally be removed because both are highly unpredictable.

It is assumed here that models will be meshed with unstructured hexahedra for structural mechanics applications, thus fixing the variables of Element Type, Element Structure, and Finite Element Analysis Application. Also, by choosing unstructured hexahedra and in the absence of an automatic algorithm, the Choice of Meshing Algorithm variable will be assumed to be the methods discussed in Section 1.2, namely sweeping and mapped mesh generation (including the multi-sweep and submapping algorithms). Further, assembly models are not considered. With these assumptions and restrictions, this paper seeks an approximation of the meshing complexity function with the following variables: Element Size, Topology, and Geometry.

Even when limiting the scope of the proposed mesh complexity metric to these factors, it should be noted that by its very nature, the values that go into defining the metric are somewhat heuristic. Extensive experience with meshing complex models has led to the emergence of the proposed complexity metric. Given the same set of circumstances, another individual may develop the metric in a different manner; however, the principles proposed in this work are universal.

2.2 Meshing Complexity Metric

The chief problem for unstructured hexahedral mesh generation lies in decomposing the model into suitable parts that are meshable with two available algorithms, namely: multi-sweep and submapping. It is, therefore, desirable for the approximation function to capture the shape of the part and to determine whether or not it can be swept. Variables that contribute to the shape of the model are termed base metrics. The approximation function should also consider those features in a model that make it difficult to mesh, such as topology and geometry problems like small curves, sliver surfaces, and small angles. Variables that are detrimental to meshing the part are termed negative metrics. There are various ways to empirically combine the base and negative metrics to compute the mesh complexity metric, \( C \). The following is proposed for such a combination:

\[
C = \text{LOG} \left( \frac{\sum_{i=1}^{n} w_i B_i}{1 + \sum_{j=1}^{k} N_j} \right), \quad (1)
\]

where \( C \) is the new meshing complexity approximation function; \( 0 \leq C \leq 1 \); \( \text{LOG} \) is the base 10 logarithm; \( n \) is the number of base metrics; \( B_i \) is the base metric, \( i \); \( 0 \leq B_i \leq 1 \); \( N_j \) is the negative metric, \( j \); \( k \) is the number of negative metrics; and \( w_i \) is the base weight, \( i \), with

\[
\sum_{i=1}^{n} w_i \leq 1 \quad (2)
\]

\( C = 1 \) for geometries that are trivial to mesh and vanishes for shapes that are challenging. The base 10 logarithm is used in Equation (1) because as the complexity of the part increases, the meshing complexity metric \( C \) approaches zero asymptotically. Equation (1) was determined experimentally. The scale of \( C \) was chosen to remain consistent with element quality metrics, where zero in meshing complexity is a non-trivial part and unity is a trivial part. It should be noted that Equation (1) does not assume any specific factors or number of factors for \( N_j \). As such, the complexity equation can easily be augmented to suit new negative metrics as they are determined to be useful. For the purposes of this study, and from extensive
experience, we have delineated a specific set of base and negative metrics on which we will develop our results.

The following base metrics are used: Inverse Topology Count, Sweep Detection, and Cartesian Edges. The negative metrics considered most effective are the number of: Small Curves, Small Surfaces, Close Loops, Small and Large Angles, Bad CAD Definitions, Groups of Blend Faces, and Tangential Surface Intersections. The one exception to Equation (1) is where the sweep detection metric has a value of unity. If this occurs, then C is set equal to unity, as will be explained in Section 3.1.2.

For the purpose of this research, a software program, SEER, was implemented to compute the meshing complexity metric. SEER uses the Common Geometry Module (CGM) [30] for its geometry query and model representation, and currently uses ACIS as the underlying geometric core. For input, SEER requires a solid model in the ACIS SAT, IGES or STEP formats. The user of the software must supply the solid model to be measured and the desired element size.

3 COMPUTATION OF METRIC

The meshing complexity metric is computed by examining the CAD model and measuring the base and negative metrics of the meshing complexity function.

3.1 Base Variables

In the proposed metric, three base metrics are included, namely, Inverse Topology Count, Sweep Detection, and Cartesian Edges. The purpose of base variables is to provide a starting range for the metric. Many parts that are difficult to mesh do not display any negative aspects. The base variables must, therefore, by themselves calculate the difficulty of the shape with respect to meshing. These three base variables do not capture the complexity entirely, but do offer an adequate beginning for most models.

3.1.1 Inverse Topology Count

The inverse topology count variable, I, is defined as:

$$I = \frac{1}{2} \left( \frac{6}{F} + \frac{12}{E} \right)$$

(3)

while

$$\frac{6}{F} + \frac{12}{E} \leq 2 \quad \text{otherwise} \quad I = 1$$

(4)

where F is the number of faces in the model and E is the number of edges. The scalar numerators ensure that the variable equates to unity for a cube. The restriction of limiting the metric to unity comes from shapes that have fewer faces and edges than that on a cube, like a sphere that has one face and zero edges.

The inverse topology count variable arises from the observation that as the number of topological entities increase, the difficulty of meshing the object also increases. If this were strictly the case, the inverse topology count variable would be the only variable needed. This, of course, is inaccurate since many cases have numerous faces and edges but are easy to mesh. Likewise, there are cases where there are relatively few faces and edges that are difficult to mesh. However, as a rule, the inverse topology count generally reflects the difficulty of meshing directly and as such is considered a relevant base variable.

3.1.2 Sweep Detection

Sweep detection directly addresses the major goal of the meshing complexity function: to identify models that are sweepable or to determine how easy it would be to transform the models into sweepable pieces. The sweep detection variable is based on the ideas proposed by White and Tautges [28]. This method, called auto sweep detection, connects the linking surfaces and traverses them in the opposite direction to determine sweep direction and source/target face identification. The method, however, does not differentiate between parts that are almost sweepable and parts that are not. The sweep detection variable proposed here uses the auto sweep detection method with some modifications to determine how “close” to sweepable a part would be.

Pseudo code for the algorithm to determine the sweep detection variable is given in Algorithm 1. In step 2 of Algorithm 1, the auto sweep detection algorithm is called. The auto sweep detection algorithm consists of four procedures to determine if a part is sweepable.

1. Let V be the CAD part being measured.
2. IF V is sweepable (use auto sweep) THEN
3. \hspace{1cm} LET C = 1.0;
4. ELSE
5. \hspace{1cm} IF V is a primitive shape, THEN
6. \hspace{2cm} LET C = 0.95;
7. \hspace{1cm} ELSE
8. \hspace{2.2cm} LET C = \zeta(V), where \zeta is the partial sweep detection metric.
9. RETURN C;

Algorithm 1 Sweep Detection Metric

The first procedure is to classify the Cartesian traversal types of the interior vertex angles for each surface and set the surface meshing schemes. At each vertex on every surface the interior angle is calculated by measuring the
angle between the two edges that join at that vertex. For non-linear edges, the tangent of the curve where it hits the vertex is used. The Cartesian traversal type, or vertex type, is assigned by rounding the angle to the closest Cartesian angle of 90, 180, 270 or 360 degrees. Based on the Cartesian angles the following values are assigned to the vertices for each surface (a vertex can have more than one vertex type value since it can be used differently for each surface of which it is a part): 1 for 90 degrees, 0 for 180 degrees, -1 for 270 degrees and -2 for 360 degrees. After all the vertices on a surface have been assigned, the meshing scheme of the surface is assigned based on the vertex types. If the sum of the vertex types on the surface is equal to 4, then the surface is submappable. If the sum is not equal to 4, then the meshing scheme is set to an automatic unstructured scheme, such as Paving [31] or Q-Morph [32].

The second procedure is to find chains or complete loops of submappable surfaces. Submappable surfaces have a logical 2D parameter space, i-j. The edges of the surfaces can be classified into four different boundaries of the parameter space: +i, -i, +j, and -j, where -i is opposite +i and -j is opposite +j. When submappable surfaces are connected with common edges, the parameter space can continue through to adjacent surfaces following the opposite parametric sides. Figure 3 shows an example of how the submappable surfaces are connected to form loops or chains of surfaces connected through opposite sides in the parameter space. The arrows in the figure are drawn to indicate the direction of the sweep chains. For this example four chains are visible with one additional chain hidden in the hole. The chains are considered complete if and only if they are non-self intersecting, meaning that the chain cannot begin in the “i” direction on a surface and then later cross the same surface in the “j” direction. Each edge in the direction maintained by the chain must also be connected to a submappable surface, or in other words, the chain must be complete or fully wrapping. Presence of these chains is a prerequisite for the volume to be swept [28].

In the third procedure, the edges of the volume are classified into Cartesian or edge types similar to the classification of the vertices for the faces. The surface normals on each side of every edge are used to measure the dihedral angles between the two faces connected at that edge. The dihedral angle is rounded to the nearest Cartesian angle and classified to the values of: 1 for 90 degrees, 0 for 180 degrees, -1 for 270 degrees and -2 for 360 degrees.

Finally, the fourth procedure is to traverse the chains found in the second procedure. To start, a face is chosen that is either not submappable or if all surfaces are submappable then an arbitrary face is taken and set to be the initial source face for the sweep. Each of the faces of the volume are reached by traversing from the edges of the first face, then recursively the edges of the next faces are found in a depth first search. The source and target faces are found by ensuring that between each source or target face there is a complete chain that runs in the direction opposite to the traversal, and that there are non-zero edge types between the source/target surfaces and the chains. For example, in Figure 3 let the initial face be Face 10. The edge between Face 10 and Face 9 has an edge type of 1. Face 9 is also in a complete chain and the edge between Faces 9 and 10 is not part of the chain. The algorithm would then move from Face 9 to Face 8. The common edge between Faces 8 and 9 has an edge type of -1, indicating that Face 8 should also be a source face with Face 10. The search would continue across Faces 6 and 5 until Face 11 (underneath) would be found and identified as a target face because of the edge type. This process would continue until all the faces are traversed, resulting in Faces 10, 8, 4 and 2 being selected as source faces and Face 11 as the target face.

Figure 3 Sweeping Chains

After auto sweep detection is run, as indicated in Algorithm 1, if the CAD part is not determined to be sweepable, the part is tested in step 5 to see if it is one of several primitives such as: sphere, half sphere, torus, tetrahedron, or cone.

If the part is neither a primitive nor sweepable, the function $\zeta(V)$, or the partial sweep detection metric, is used to determine the sweep detection variable. The pseudo code for $\zeta(V)$ is given in Algorithm 2. Step 6 of Algorithm 2 is picking the corners of the faces that are “forced” to be represented with the rectangular meshing primitive. Corner picking, devised by Mitchell [33], is applied to choose the most appropriate corners of the face.
1. FOR EACH Face f in V
2. LET X(f) be the mesh scheme of f
3. LET Y(f) be the number of holes in f
4. IF X(f) is unstructured AND Y(f) is > 1 THEN
5. LET X(f) = Rectangle Primitive
6. Choose the best corners for f
7. LET LC be the number of linking chains in V.
8. IF LC == 0 THEN
9. RETURN 0.0.
10. ELSE
11. return S(V)

Algorithm 2 Partial Sweep Detection

In step 7, the third procedure of the auto sweep detection algorithm is redone; namely chains of linking faces are determined. If no chains are found, the sweep detection metric is set to zero in step 9. Otherwise, in step 11, the chains are used to compute the sweep detection metric S. The sweep detection metric is computed as:

\[ S = M_{b} * M_{h} * M_{c} * \text{MAX}\left(\frac{T_{CF}}{q}, \sum_{i=1}^{m} \sum_{j=1}^{l} A_{ij} \right) \]

where \( M_{b}, M_{h}, \) and \( M_{c} \) are modifiers based, respectively, on how much the chains cover the volume, the number of cylinder holes, and the large numbers of chains present; \( \text{MAX} \) is the function to return the maximum of two scalars; \( T_{CF} \) is the number of faces in the chains; \( q \) is the number of edges in the volume; \( m \) is the number of chains; \( l \) is the number of faces in chain \( i \); \( A_{ij} \) is the area of the face \( j \) in the chain \( i \); and \( A_{i} \) is the area of the face \( k \) in the volume.

The modifier \( M_{b} \) is determined by comparing the union of all the bounding boxes of the chains and the bounding box of the volume. The comparison is made by checking the number of directions out of three (x, y, and z) in which the bounding box of the chains is equal in size to the bounding box of the volume. For example, if the bounding box for the chains is \([0,0,0], [1,1,1] \) and the bounding box of the volume is \([0,-1,0], [1,1,1] \) then the boxes will be equal in two directions. The modifier, \( M_{b} \), is equal to 0.9 if the boxes are equivalent in no directions, 1.4 if there is one direction, 2.0 if there are two directions, and 4.0 if the boxes are equivalent in all three directions. These values were determined through numerical experiment.

The modifier \( M_{b} \) is determined by finding the number of chains that are formed by simple cylinder holes with one or two surfaces constituting the entire chain. In general, these types of chains do little to determine how sweepable the part is, so this circumstance is viewed as a reducing modifier. \( M_{b} \) is computed by finding the fraction of chains that are not cylinder holes out of the total number of chains. This modifier is lower bounded to be 0.01 since the presence of holes indicates some form of a sweep direction.

The final modifier \( M_{c} \) is another reduction factor to reflect whether there are numerous chains on the volume. If there are more than ten chains, their presence can indicate that the volume has many sweep directions, meaning that meshing may be more difficult than volumes that have relatively few sweep directions. Therefore, if the bounding box of the chains is not equivalent in all three directions to the bounding box of the volume, and there are more than ten chains, \( M_{c} \) is set to be equal to the inverse of the number of chains; otherwise, \( M_{c} \) is unity.

There are, of course, many counter examples for the sweep detection metric, which is why it is only a part of the entire meshing complexity metric. However, as a general rule, the metric is shown to yield lower values for geometries that are more difficult to mesh and correctly predicts higher values for trivial geometries.

3.1.3 Cartesian Edges

The Cartesian Edges metric is intended to capture the degree to which the volume is of a “blocky” nature. In general, a blocky volume is easier to mesh since the hexahedra fit easier when the faces are aligned orthogonally. The metric is computed by first counting the number of edges connected to planar surfaces with Cartesian dihedral angles between them. For this metric, Cartesian dihedral angles are specified to be 90, 180 and 270 degrees, respectively. Dihedral angles that fall within three degrees of these values are considered Cartesian. After the number of Cartesian edges is found, the metric is calculated by dividing this number by the total number of edges in the volume. The Cartesian Edges ratio is found to be inaccurate and arbitrary when it is less than 0.4. Therefore, the metric is set to zero for volumes with ratios of less than 0.4, and set to the ratio itself for ratios greater than 0.4.

3.1.4 Base Metric Weights

The weights for the three base metrics were computed by trial and error. It was found that 0.1 would be the best weight for the Inverse Topology Count metric. For the Sweep Detection metric, the best weight was experimentally found to be 0.5. Again by numerical experiments, the Cartesian Edges metric was given a “stepping” weight based on the metric itself. The Cartesian Edges weights are defined in Algorithm 3. It is acknowledged that if the Cartesian Edges metric is less than 0.6 then the sum of the weights will be less than one, as reflected in Equation (2). It was, however, necessary to reflect the increased importance of the Cartesian Edges metric as the value of the metric increased.
1. LET CE be the computed Cartesian Edge metric value
2. LET w CE = 0.0 where w CE is the weight of the Cartesian Edge metric
3. IF CE >= 0.4 AND CE < 0.5 THEN
4. LET w CE = 0.1
5. ELSE IF CE >= 0.5 AND CE < 0.6 THEN
6. LET w CE = 0.2
7. ELSE IF CE >= 0.6 THEN
8. LET w CE = 0.6

Algorithm 3 Base Metric Weight Assignment

3.2 Negative Metrics

The negative metrics are used to cause a decrease in the value of the meshing complexity metric. The negative metrics reflect problems found in the model that will be detrimental to mesh generation. This study was unable to determine a ranking of these problems in terms of their impact on the meshing process, and as such they are weighted equally at this time. Additionally, test results revealed that all the negative metrics were rarely found in a single part. This indicates that meshing complexity can vary greatly between parts, and that the list of negative metrics mentioned here may be incomplete. New negative metrics may be added to the existing list until a general metric is found. The negative metrics identified during this study are now presented. Unlike the base metrics, these are all determined by counting the number of occurrences of a given problem. Additionally, many of these metrics are functions of element size, meaning that if the desired element size for meshing the volume changes it could impact the results of these metrics.

3.2.1 Number of Small Edges, Faces, and Close Face Loops

Small edges, faces, and close face loops all adversely affect the meshing process, making it difficult and sometimes impossible to generate a reasonable quality mesh. In fact, such artifacts often lead to robustness issues with automatic surface meshing algorithms unless mesh sizing in the region is performed carefully. Finding these entities visually is often difficult. Additionally, before the final mesh is achieved, these entities must usually be removed from the model, adding additional time to the mesh generation process.

The number of small edges is equal to the number of edges in the model with a length less than or equal to 1/5 the element size given as input. The number of small faces is equal to the number of faces with a hydraulic radius less than or equal to 1/5 the element size. The hydraulic radius [15] is computed as, $R_h = \frac{4A}{P}$, where $A$ is the area of the face and $P$ is the total length of the parameter of the face. The number of close loops metric is equal to the number of minimum distances between loops that are less than or equal to 1/5 the element size.

3.2.2 Number of Small and Large Angles

Small and large angles can also affect the quality of the mesh. When the angles are very small, the geometry must be modified in order to generate a suitable mesh. The angles for this metric are calculated at vertices on a surface and curves on a volume. For vertices the angle is calculated by measuring the angle between the tangents of the two edges at that vertex on a particular surface. For edges the angle is calculated by measuring the angle between the normal vectors of the two faces that share the common edge. The normals are measured at the mid-point of the edge and assumed to be constant throughout the length of the edge. The number of small and large angles is computed by counting all the vertex and edge angles that are obtained by measuring all the vertex and edge angles that are smaller than 20 degrees or larger than 340 degrees.

3.2.3 Number of Bad CAD Definitions

The ACIS geometry engine [4] is used in this study to represent the underlying geometric definitions of the CAD models. As mentioned in Section 1.1.1, for various reasons these definitions may be inaccurate or faulty. Many of these problems can be fixed via the healing technologies available in various software packages. In general, the presence of these problems indicates an increased time to generate a mesh. If the problems can not be fixed, meshing can be difficult since these problems can limit the use of decomposition tools at one extreme and make meshing impossible without rebuilding the geometry at the other. The ACIS geometry engine provides the ability to detect geometry definitions errors. For computing this metric, the ACIS software is queried directly to determine how many entities have problems.

3.2.4 Number of Groups of Blend Faces

Blend faces, more commonly referred to as fillets and rounds, are commonplace in CAD models. Their presence is usually directly related to increased meshing times, especially for hexahedral meshing. Cartesian edges are typically best for mesh quality in hexahedral meshing. Additionally, presence of blend faces usually does two things to a model: add extra topology that hinders mesh generation, and remove topology that is needed for sweeping. An example of this is shown in Figure 4 where a
brick of dimension 10 units has all of its curves blended with a radius of 1.0. The result is that this trivial meshing shape becomes more difficult and requires topological modifications before it can be meshed.

Figure 4 Brick with All Edges Blended

The best approach to dealing with these entities for hexahedral meshing is to remove them and return the model to Cartesian intersections. In many cases this is not allowable since, while some of the blends are present for appearance, many more are there for physical reasons. Regardless of whether or not the blends can be removed, their presence leads to mesh generation challenges and increases the time required to generate a mesh, making it important for the meshing complexity metric to capture these faces.

A blend group is a set of connected blend faces that are generally blending the same area although the edge they blend may change for various reasons. Because of these changes, the number of blend groups is found rather than the number of blend faces. While others have attempted to detect blend faces [15], a new method is presented here.

For blend face detection, the following three rules are made. First, blend faces have at least one edge where the faces attached to that edge have a dihedral angle of 180 degrees at the edge. This edge is called the parallel edge. Second, an edge opposite the parallel edge has a Cartesian angle, 90, 180, 270 degrees, between the two faces that are attached to it. The edge opposite the parallel edge is called the opposite edge. And third, the angle between the face normals on the parallel edge and the opposite edge is also Cartesian. An example of a blend face that fits these three rules is shown in Figure 5 where the parallel and opposite edges are identified.

To find blend faces, the model is searched for faces that meet these criteria. Determining the opposite edge is the most difficult part. The algorithm starts with the parallel edge, then searches the other edges on the face for the edge that is at the closest distance and most parallel to it. The search is done by traversing the edges counter-clockwise around the face. The edges are not tested until a vertex angle of less than 135 degrees is passed. The edge selected in this process is called the opposite edge. If the face only has three curves, then the face is a blend face only if all the tested curves meet the first rule, and all edges are attached to other blend faces.

After all the blend surfaces have been detected, the blend surfaces themselves are traversed going from their edges to
adjacent faces using a union-find algorithm to group the blend faces that are connected by an edge. The groups of blend faces metric is calculated by counting up the total number of blend groups found from this algorithm. Figure 6 (a), (b) and (c) are test parts with blend faces. The blend algorithm is able to find all the blend faces in these models. Several faces that would typically not be considered blend faces by visual inspection are also included as a side effect of the algorithm. In general, the algorithm is able to detect blend surfaces, and especially the hidden ones shown in Figure 6 (c) that make mesh generation difficult.

3.2.5 Number of Tangential Face Intersections

The final metric included in the negative metrics is the number of tangential face intersections. This metric seeks to find dissimilar geometrical faces that intersect tangentially and cause problems with mesh generation. Having two faces intersect tangentially is generally not a problem for mesh generation. Figure 7 (a) shows an example of two faces that intersect tangentially that offer no resistance to mesh generation. In this figure, the two faces intersect without creating any small features. Figure 7 (b), however, shows an example of the tangency causing small features. Mesh generation is difficult when tangential face intersections are "capped" by a face with small vertex angles at the intersection. These configurations are identified in this metric.

![Tangential Face Intersection](image)

**Figure 7 Tangential Face Intersections**

Tangential face intersections that cause problems are identified by the following four steps. First, the user must gather all the faces and vertices that have small vertex angles. This information can be reused from the computation of the small and large angles metric discussed in Section 3.2.2. Second, identify the face that makes a 90 degree dihedral angle with the face with the small vertex angle. This face will be attached to either of the two edges that meet at the vertex with the small angle. Third, on the face found in step two, find the edge that is also connected to the vertex with the small angle but is not attached to the face where the small angle is. And fourth, measure the dihedral angle for this edge to ensure that this edge has a tangential intersection. The angle should be 180 degrees. If all of these steps are done successfully, a tangential face intersection is recorded and counted. The metric is computed by counting all such instances.

4 RESULTS

The meshing complexity metric is best evaluated on real CAD geometries rather than contrived test parts. A set of twenty-four CAD models were obtained for this purpose. The models range from simplistic to challenging in terms of their difficulty with respect to mesh generation. This section will discuss evaluation of the twenty-four models and the resulting meshing complexity metrics.

4.1 Test Suite

To test the validity of the meshing complexity metric, CAD models were obtained and meshed with hexahedral elements. Twenty-four test models were obtained from various industrial partners. The meshing was done with the CUBIT software package [34] that contains the meshing algorithms described in Section 1.2. One of the authors performed all of the meshing so that there would be no differences in the time taken to mesh the parts based on differences in user expertise. All the time taken to mesh the models was recorded, including time spent thinking about approaches, trying different approaches, and time spent recovering from user mistakes. This is important to note since the meshing complexity metric must try to capture in some form which geometries and topologies cause the user more time. Time wasted due to bugs in the meshing software was discarded because of the assumptions stated in Section 2.1. Table 1 shows the part numbers, the element size for which they were meshed, and the total time taken to generate the mesh. The time to mesh varies from under a minute to 833 minutes for the part that took the longest. Three of the parts and resulting meshes are shown in Figure 8. Additionally, the parts shown in Figure 6 (a-c) are part numbers 1, 20 and 5, respectively.
<table>
<thead>
<tr>
<th>Part Number</th>
<th>Element Size</th>
<th>Time to Mesh (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>613</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>327</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>114</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0.17</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>369</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>102</td>
</tr>
<tr>
<td>10</td>
<td>0.005</td>
<td>2.5</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>270</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>719</td>
</tr>
<tr>
<td>15</td>
<td>0.2</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>0.5</td>
<td>0.083</td>
</tr>
<tr>
<td>17</td>
<td>23.9</td>
<td>0.083</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>146</td>
</tr>
<tr>
<td>20</td>
<td>0.09</td>
<td>425.5</td>
</tr>
<tr>
<td>21</td>
<td>0.5</td>
<td>0.017</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>0.083</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>58</td>
</tr>
<tr>
<td>24</td>
<td>5</td>
<td>13.5</td>
</tr>
</tbody>
</table>

Table 1 Mesh Test Suite with Element Size and Recorded Times to Mesh

4.2 Correlation of Data

The proposed meshing complexity metric is compared with times taken to mesh each part in the test suite. The results of this experiment are shown in Figure 9; where, time to mesh is on the x-axis while the mesh metric is plotted on the y-axis on the logarithmic scale. The logarithmic scale is used to plot this axis since the mesh metric asymptotically approaches zero as the parts become more complex. The graph shows that the meshing complexity metric predicts the time to mesh within some range of error. Specifically, it shows that as the time required to mesh the parts increases, the metric value decreases asymptotically towards zero. For cases where the metric does not correlate well, this could be due to both the limitations of the new metric and the difficulty in prediction of human computer interactions. Table 2 shows the results of the individual base metrics for the twenty-four test cases.

Table 3 shows the results of the individual negative metrics with the corresponding meshing times and part numbers. This table shows that the models that took longer to mesh contained more negative features, indicating the adverse affect these features have on the meshing process.

Figure 8 Test Parts 3(a), 13(b), and 14(c)

Figure 9 Meshing Complexity Metric vs. Time to Mesh
The parameters of Equation (1) are obtained from the first row of Table 2 and Table 3 as: \( B_1 = 0.044323607, \ w_1 = .1, B_2 = 0.0211363, \ w_2 = 0.5, B_3 = 0, N_1 = 2, N_2 = 17, N_3 = 9, N_4 = 23 \) and \( N_5 = 30 \). After inserting these values into Equation 1, the meshing complexity \( C \) for part 1 is \( 2.25 \times 10^{-6} \), as indicated in Figure 9.

Several improvements could be introduced to the meshing complexity metric. First, the metric lacks a general base metric which determines how sweepable the part is. While the sweep detection metric attempts this, it is not always accurate and can give both false-positive and positive-false results. If such a metric is found, it would also most likely lead to automatic decomposition approaches to vastly reduce the time to mesh for hexahedral elements.

Second, not all variables that affect meshing complexity are accounted for in this study. For example, in Section 2.1, it was assumed that User Expertise would not be considered a factor. In theory, this should be the case if the metric compares parts that were meshed by the same user. In practice, however, this may not be true as human interaction is difficult to predict. For example, on some of the parts in the test suite the solution to meshing the part was immediately recognized, leading to a shorter time to generate the mesh. Other problems required more thinking, and time spent reflected this trial and error. To improve the metric, the human computer interactions aspect of the problem could be incorporated. Despite these inaccuracies, the proposed metric does predict the relative difficulty between the parts in terms of the time required to generate a valid finite element mesh within a range of error.

### Table 2 Base Metric Values for Test Suite

<table>
<thead>
<tr>
<th>Number</th>
<th>Time To Mesh</th>
<th>Inverse Topology Count</th>
<th>Sweep Detection</th>
<th>Cartesian Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>813</td>
<td>0.044323607</td>
<td>0.0211363</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>0.028393818</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>579</td>
<td>0.089935065</td>
<td>0.000964</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>327</td>
<td>0.061111111</td>
<td>0.038482</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>114</td>
<td>0.1755</td>
<td>0.56875</td>
<td>0.419355</td>
</tr>
<tr>
<td>6</td>
<td>0.17</td>
<td>0.146320346</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.083</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>8</td>
<td>369</td>
<td>0.052859421</td>
<td>0.049500958</td>
<td>0.041284</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>0.017</td>
<td>0.083</td>
<td>0.059738458</td>
</tr>
<tr>
<td>10</td>
<td>102</td>
<td>0.017</td>
<td>1</td>
<td>0.344848486</td>
</tr>
<tr>
<td>11</td>
<td>2.5</td>
<td>0.017</td>
<td>0</td>
<td>0.136090226</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>0.017</td>
<td>0</td>
<td>0.059738486</td>
</tr>
<tr>
<td>13</td>
<td>719</td>
<td>0.017</td>
<td>0</td>
<td>0.019699932</td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td>0.017</td>
<td>0</td>
<td>0.070211039</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>0.017</td>
<td>0</td>
<td>0.0273</td>
</tr>
<tr>
<td>16</td>
<td>0.083</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>0.083</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>18</td>
<td>36</td>
<td>0.158004158</td>
<td>0.466486</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>146</td>
<td>0.048500958</td>
<td>0.041284</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>425.5</td>
<td>0.04931151</td>
<td>0.030319</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0.017</td>
<td>0.52941765</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0.083</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>58</td>
<td>0.055927835</td>
<td>0.031235</td>
<td>0.497908</td>
</tr>
<tr>
<td>24</td>
<td>13.5</td>
<td>0.366666667</td>
<td>0.8</td>
<td>0.571429</td>
</tr>
</tbody>
</table>

### Table 3 Negative Metric Values for Test Suite

Using the data in Table 2 and Table 3, the weights of the base metrics in Section 3.1.4, and Equation (1), the meshing complexity metric results in Figure 9 can be computed. To demonstrate this, the meshing complexity metric is calculated for part 1. The values for various parameters of Equation (1) are obtained from the first row of Table 2 and Table 3 as: \( B_1 = 0.044323607, \ w_1 = .1, B_2 = 0.0211363, \ w_2 = 0.5, B_3 = 0, N_1 = 2, N_2 = 17, N_3 = 9, N_4 = 23 \) and \( N_5 = 30 \). After inserting these values into Equation 1, the meshing complexity \( C \) for part 1 is \( 2.25 \times 10^{-6} \), as indicated in Figure 9.

Several improvements could be introduced to the meshing complexity metric. First, the metric lacks a general base metric which determines how sweepable the part is. While the sweep detection metric attempts this, it is not always accurate and can give both false-positive and positive-false results. If such a metric is found, it would also most likely lead to automatic decomposition approaches to vastly reduce the time to mesh for hexahedral elements.

Second, not all variables that affect meshing complexity are accounted for in this study. For example, in Section 2.1, it was assumed that User Expertise would not be considered a factor. In theory, this should be the case if the metric compares parts that were meshed by the same user. In practice, however, this may not be true as human interaction is difficult to predict. For example, on some of the parts in the test suite the solution to meshing the part was immediately recognized, leading to a shorter time to generate the mesh. Other problems required more thinking, and time spent reflected this trial and error. To improve the metric, the human computer interactions aspect of the problem could be incorporated. Despite these inaccuracies, the proposed metric does predict the relative difficulty between the parts in terms of the time required to generate a valid finite element mesh within a range of error.

### 5 CONCLUSION

A new metric is presented to predict the meshing complexity of CAD parts. Meshing complexity refers to the relative difficulty to generate an unstructured hexahedral mesh between various CAD parts. Two kinds of metrics have been identified; they are called base metrics and negative metrics, respectively. Base metrics analyze the shape of the parts while negative metrics detect features that traditionally cause problems with mesh generation. Three base metrics have been identified: Inverse Topology Count, Sweep Detection and Cartesian Edges. The Inverse Topology Count metric captures complexity that typically exists as the number of the topological entities the model increases. The sweep detection metric identifies the degree to which the model can be swept by inspection of both the topology and geometry. The Cartesian Edges metric measures the level to which the model is blocky. Five negative metrics are proposed: Number of Small Edges, Faces and Close Face Loops; Number of Small and Large Angles; Number of Bad CAD Definitions; Number of Groups of Blend Faces; and the Number of Tangential Face Intersections. The Number of Small Edges, Faces and Close Face Loops metric is determined by the element size provided by the user and identify small regions that are difficult to mesh. The Number of Bad CAD Definitions metric is obtained by identifying the bad geometric definitions that can impede mesh generation. The Number of Groups of Blend Faces and the Number of Tangential Face Intersections metrics identify the often harmful blend
faces and tangential intersections through sets of proposed rules. The base and negative metrics are combined to form the proposed metric.

The proposed complexity metric is compared with the timing data. For some parts the metric does not correlate accurately due to both the limitations of the metric and the difficulty in predicting human computer interactions.

The proposed metric is intended to be used for several reasons. First, by analysts and managers to aid in predicting the time it will take to generate a mesh on certain CAD models. Based on previously meshed models, and the metric values computed for them, analysts and managers can compute the complexity of new models allowing them to predict the time required for mesh generation. The proposed metric could also be used to help measure progress that is made in research in unstructured hexahedral mesh generation.

6 ACKNOWLEDGEMENTS

The authors wish to acknowledge Sandia National Laboratories for providing funding and motivation for this project. Additionally, we wish to thank ANSYS Inc. for providing many of the CAD models that were included in the test suite.

7 REFERENCES


COMPACT REPRESENTATIONS OF SIMPLICIAL MESHES IN TWO AND THREE DIMENSIONS*

Daniel K. Blandford          Guy E. Blelloch          David E. Cardoze          Clemens Kadow
dkbl@cs.cmu.edu              blelloch@cs.cmu.edu       cardoze@cs.cmu.edu        kadow@cmu.edu

Carnegie Mellon University, Pittsburgh, PA, U.S.A.

ABSTRACT

We describe data structures for representing simplicial meshes compactly while supporting online queries and updates efficiently. Our representation requires about a factor of five less memory than the most efficient standard representations of triangular or tetrahedral meshes, while efficiently supporting traversal among simplices, storing data on simplices, and insertion and deletion of simplices.

Our implementation of the data structures uses about 5 bytes/triangle in two dimensions (2D) and 7.5 bytes/tetrahedron in three dimensions (3D). We use the representations to implement 2D and 3D incremental algorithms for generating a Delaunay mesh. The 3D algorithm can generate 100 Million tetrahedrons with 1 Gbyte of memory, including the space for the coordinates and all data used by the algorithm. The runtime of the algorithm is as fast as Shewchuk’s Pyramid code, the most efficient we know of, and uses a factor of 3.5 less memory overall.

Keywords: mesh representations, computational geometry, compression

1. INTRODUCTION

The space required to represent large unstructured meshes in memory can be the limiting factor in the size of a mesh used in various applications. Standard representations of tetrahedral meshes, for example, can require 300-500 bytes per vertex. One option for using larger meshes is to maintain the mesh in external memory. To avoid thrashing, this requires designing algorithms that carefully orchestrate how they access the mesh. Although several such external memory algorithms have been designed [17, 12, 10, 27, 41, 1, 40], these algorithms can be much more complicated than their main-memory counterparts, and can be significantly slower.

Another option for using larger meshes is to try to compress the representation within main memory. There has in fact been significant interest in compressing meshes [11, 21, 39, 31, 32, 38, 24, 22, 16]. In three dimensions, for example, these methods can compress a tetrahedral mesh to less than a byte per tetrahedron [38]—about 6 bytes/vertex. These techniques, however, are designed for storing meshes on disk or for reducing transmission time, not for representing a mesh in main memory. They therefore do not support dynamic queries or updates to the mesh while in compressed form.

We are interested in compressed representations of meshes that permit dynamic queries and updates to the mesh. The goal is to solve larger problems while using standard main-memory algorithms. In this paper we present data structures for representing two and three dimensional simplicial meshes. The representations support standard operations on meshes including traversing among neighboring simplices, inserting and deleting simplices, and the ability to store data on simplices. For a class of well shaped
meshes [29] these operations all take constant time. The precise definition of our interface is described in Section 4. Although our representations are not as compact as those designed for disk storage, they still save a factor of between 5 and 10 over standard representations.

Our data structures are described in Section 5. They take advantage of the separator property of well-shaped meshes [29], and use recent results in compressing graphs [4]. In particular our technique uses separators to relabel the vertices so that vertices that share a simplex are likely to have labels that are close in value. Pointers are then difference encoded using variable length codes. We use this technique to radially store the neighboring vertices around each vertex in 2D and around a subset of the edges in 3D. A query need only decode a single vertex in 2D or vertex and edge in 3D. For applications that need to generate new vertices, e.g., Delaunay refinement, we leave extra space in the label space and assign new labels based on the labels of the neighbors.

Section 6 describes an implementation of our data structure and Section 7 presents experimental results. The implementation uses about 5 bytes per triangle in 2D and about 7.5 bytes per tetrahedron in 3D when measured over a range of mesh sizes and point distributions. We present experiments based on using our representation as part of incremental Delaunay algorithms in both 2D and 3D. We use a variant of the standard Bowyer-Watson algorithm [8, 42] and the exact arithmetic predicates of Shewchuk [37] for all geometric tests. We also present experiments based on a Delaunay refinement algorithm that removes triangles with small angles by adding new points at their circumcenters. All space is reported in terms of the total space including the space for the vertex coordinates and all other data structures required by the algorithm. The results for 1 Gbyte of memory are summarized as follows.

- We can generate a 2D Delaunay mesh with 110 million triangles (.47 Gbytes for the mesh, .44 Gbytes for the vertex coordinates, and about .1 Gbytes for auxiliary data used by the algorithm). Compared to the Triangle code [36] (the most efficient we know of) our algorithm uses a factor of 3 less memory. It is about 10% slower than Triangle’s divide-and-conquer algorithm and much faster than its incremental algorithm.

- We can generate a 3D Delaunay mesh with 100 million tetrahedrons (.75 Gbytes for the mesh, .17 Gbytes for the vertex coordinates, and .08 Gbytes for auxiliary data). Compared to the Pyramid code [35], our algorithm uses a factor of 3.5 less memory, and is about 30% faster.

- We can generate a refined 2D Delaunay mesh with 80 million triangles with no angle less than 26%. This version dynamically generates new labels, and uses an extra level of indirection in our data-structure.

Our representation can be used in conjunction with external memory algorithms. Also, although we only describe our implementation for 2D and 3D simplicial meshes, the ideas extend to higher dimensions. These topics are discussed, briefly, in Section 8.

2. STANDARD MESH REPRESENTATIONS

There have been numerous approaches for representing unstructured meshes in 2 and 3 dimensions. Some are specialized to simplicial meshes and others can be used for more general polytope meshes. For the purpose of comparing space usage, we review the most common of these representations here. A more complete comparison for 2D data structures can be found in a paper by Kettner [25].

In two dimensions most approaches are based on either triangles or edges. The simplest representation is to use one structure per triangle. Each structure has three pointers to the neighboring triangles, and three pointers to its vertices. Assuming no data needs to be stored on triangles or edges, this representation uses 6 pointers per triangle. Storing data requires extra pointers. Shewchuk’s Triangle code [36], and the CGAL 2D triangulation data structure [7] both use a triangle-based representation. To distinguish the three neighbors/vertices of a triangle, a handle to a triangle typically needs to include an index from 1 to 3. The data structure used by Triangle, for example, includes such an index in the pointer to each neighbor (in the low 2 bits) so that a neighbor query not only returns the neighbor triangle, but returns in which of three orders it is held.

There are many closely related representations based on edges, including the doubly connected edge list [30], winged-edge [3], half-edge [43], and quad-edge [20] structures. In addition to triangulated meshes, these representations can all be used for polygonal meshes. In these representations each edge maintains pointers to its two neighboring vertices and to neighboring edges cyclically around the neighboring faces and vertices. Each edge might also maintain pointers to the neighboring faces and to edge data. The most space efficient of these representations can maintain for each edge a pointer to the two neighboring vertices and to just two neighboring edges, one around each face and vertex. Assuming no data needs to be stored on a face or edge, this requires 4 pointers per edge.
which for a manifold triangulation is equivalent to the
6 pointers per triangle used by the triangle structure
(\|E\| = 3/2|T|). The half-edge data structure [43], used
by CGAL [25], LEDA [28] and HGAM [18], maintains
two structures per edge, one in each direction. These
half-edges are cross referenced, requiring an extra two
pointers per edge. The winged-edge and quad-edge
structures maintain pointers to all four neighboring
edges, requiring 6 pointers per edge (9 per triangle).

In three dimensions there are analogous representa-
tions based either on tetrahedrons or on faces and
edges. Again the simplest representation is to use a
structure per tetrahedron. Each tetrahedron has 4
pointers to adjacent tetrahedrons, and 4 to its cor-
ner vertices. Assuming no data this requires 8 point-
ers per tetrahedron. This representation is used by
Pyramid [35] and CGAL [7]. The face and edge represen-
tations are often called boundary representations
(b-reps). Such boundary representations are more
general than the tetrahedron representations, allow-
ing the representation of polytopes meshes, but tend
to take significantly more space. Dobkin and Las-
zlo [13] suggest a data structure based on edge-face
pairs, which in general requires 6 pointers per edge-
face. For tetrahedral meshes this representation can
be optimized to 9 pointers per face (6 to the adja-
cent faces rotating around its 3 edges, and 3 to the
corner vertices). This corresponds to 18 pointers per
tetrahedron. Weiler’s radial-edge representation [44],
Erisson’s cell-tuple representation [9], and Linehard’s
G-map representation [26] all take more space.

In summary, the most efficient standard representa-
tions of simplicial meshes use 6 pointers per triangle
in 2D and 8 pointers per tetrahedron in 3D. At least
one extra pointer is required to store data on triangles
in 2D or tetrahedrons in 3D.

3. PRELIMINARIES

In this section we review some basic notions of combi-
natorial topology used in this paper. For a more de-
tailed discussion the reader can refer to [14] and [33]
among others.

An (abstract) simplicial complex K is a non-empty col-
clection of finite sets which is closed under taking non-
empty subsets. The elements of K are called simplicies.
The underlying set \text{UK} of K is called the vertex set and its
elements are called vertices. The dimension of a sim-
plex with d vertices is \(d-1\). The dimension of K is the
maximum dimension among its simplicies. A simplicie \(\tau\)
is a face of a simplicie \(\gamma\) iff \(\tau \subseteq \gamma\), and if \(\tau \neq \gamma\) we say
that \(\tau\) is a proper face of \(\gamma\). We say that K is pure if
every simplicie is a face of a simplicie of highest dimen-
sion. Let \(S\) be a subset of \(K\). We call the collection of
all simplicies in \(S\) together with all their faces, \(\text{CL}(S)\),
the closure of \(S\). The star of a simplicie is the union of
its superfaces, \(\text{St}(\sigma) = \{ \gamma : \sigma \subseteq \gamma \}\). The link of a sim-
plicie \(\sigma\) is the set of simplicies in the closure of its star
that do not intersect it, \(\text{Lk}(\sigma) = \text{CL}(\text{St}(\sigma)) - \text{St}(\sigma)\).

Let \(E\) be a mapping from the vertices of \(K\) to \(\mathbb{R}^m\). We
let \(|\sigma|\) denote the convex hull of the images of its
vertices of \(\sigma\) under \(E\), and let \([K] = \cup_{\sigma \in K}|\sigma|\).
We say that \([K]\) is an embedding of \(K\) iff for all simplicies
\(\sigma\) and \(\tau\) it holds that \(|\sigma| \cap |\tau| = |\gamma|\) where \(\gamma\) is their
maximum common face (which may be empty). We
say that \(K\) is a \(d\)-manifold (with boundary) iff \([K]\) is
a \(d\)-manifold (with boundary). If \(K\) is a manifold of
dimension \(d\) then the link of every \((d-2)\)-simplex
is a cycle of edges and vertices (i.e., a 1-manifold). If
\(K\) is a manifold with boundary, then the link of every
\((d-2)\)-simplex is either a cycle or a path, i.e., a 1-
manifold with or without boundary (see Figure 1 (a)).

We will make use of this fact in our representation
described in section 5.

An ordering, \(s^d\), of a \(d\)-simplex, \(s^d\), is a total ordering
of its vertices. An orientation, \(\overrightarrow{s^d}\), of a simplex, \(s^d\), is a
maximal set of orderings which are even permutations
of each other\(^1\). Every ordering \(\overrightarrow{s^d}\) on a simplex implies
an orientation \(\overrightarrow{s^1}\) on the simplex, and every \(d\)-simplex,
\(d \geq 0\), has two possible orientations. The orientation
\(\overrightarrow{s^d}\) of a simplex induces an orientation \(\overrightarrow{s^{d-1}}\) on every
d \((d-1)\)-subsimplex—i.e., for all \(\overrightarrow{s^{d-1}} \in \overrightarrow{s^{d-1}}\) there exists
\(\overrightarrow{s^d} \in \overrightarrow{s^d}\) such that \(\overrightarrow{s^{d-1}}\) is a prefix of \(\overrightarrow{s^d}\).

For our purposes, a \(d\)-pseudomanifold is a pure \(d\-
complex where every \((d-1)\)-simplex is contained in
at most two \(d\)-simplicies and where the dual graph is
connected. The vertices of the dual graph are the \(d\-
simplicies and the edges are the \((d-1)\)-simplicies. A
\(d\)-pseudomanifold is orientable if its \(d\)-simplicies can be
given orientations in such a way that when they meet
at a \((d-1)\)-simplex \(s\), they induce opposite orien-
tations on \(s\). Every orientable \(d\)-pseudomanifold has
two possible orientations, which can be specified by
the orientation of one of its \(d\)-simplicies. If \(K\) is a \(d-
pseudomanifold then the link of every \((d-2)\)-simplex
is a collection of disjoint cycles and/or paths (see Fig-
ure 1 (b)).

In this paper we will use the term simplicial mesh to
refer to a pseudomanifold abstract simplicial complex
with a given orientation.

4. INTERFACE

In this section we present the interface for simplicial
meshes that our representation implements. It is
a simplified version of an interface described in [5].
The interface supports standard operations on meshes
\(^1\)An even permutation is a permutation reached by an
even number of swaps.
including, a mechanism to systematically traverse a mesh (e.g., reflect across a face, or rotate around a vertex), and for updating the mesh, including inserting and deleting simplices and associating data with simplices. The interface consists of three operations on ordered simplices (empty, up, and down), and three operations on simplicial meshes (add, delete, and findUp).

Let \( s^k = (v_1, \ldots, v_{k+1}) \) be an ordered simplex. The empty operation creates an empty simplex \( \text{empty}() \rightarrow () \). The up operation adds a vertex \( v \) to \( s \): \( \text{up}(s^k, v) \rightarrow (v_1, \ldots, v_{k+1}, v) \). The down operation extracts the last vertex from \( s^k \): \( \text{down}(s^k) \rightarrow (v_1, \ldots, v_k) : v_{k+1} \).

Let \( M \) be a \( d \)-dimensional simplicial mesh. The add operation takes \( M \) and a highest dimension ordered simplex \( s^d \) and returns a new mesh \( M' \) that results from adding \( s^d \) to \( M \). We require that \( s^d \) has consistent orientation with \( M \). Note that when we add \( s^d \), we don’t have to store all even permutations of \( s^d \)—just storing \( s^d \) is enough to determine the orientation of \( s^d \). The delete operation takes \( M \) and a highest dimension ordered simplex \( s^d \) and returns the mesh \( M' \) that results from removing \( s^d \) from \( M \). The findUp takes an ordered simplex \( s^k \), \( 0 \leq k \leq d \), and \( M \). If \( s^k \) is not in \( M \) it returns null. Otherwise it returns an ordered simplex \( s^d \) such that \( s^d \in M \) and \( s^k \) is a prefix of \( s^d \). In the special case where \( k = d - 1 \), then there is at most one \( s^d \) that can be returned.

In addition to the core interface, we also provide two operations to associate and retrieve data from simplices in a complex. The addData operation takes \( M \), an ordered simplex \( s^k \), \( 0 \leq k \leq d \) and some user supplied data \( u \). It associates \( u \) with \( s^k \) in \( M \). The operation findData takes \( M \) and an ordered simplex \( s^k \) and returns the user data \( ud \) associated with \( s^k \) in \( M \).

If there is no associated data then null is returned.

5. REPRESENTATION

Here we describe our 2D and 3D representations for simplicial meshes (simplicial orientable pseudo manifolds). We first describe uncompressed versions of the representations and then describe how to compress them. Our representations are based on storing the link for a set of \((d - 2)\)-simplices. In 2D this is similar to the half-edge structure [43], and in 3D it is similar to the Dobkin and Laszlo [13] structure. We note, however, that all references are to vertex labels instead of pointers to other higher-dimensional simplex structures, allowing us to compress based on vertex labels. Our representations have the property that if the degree of all vertices is bounded all queries take constant time. We first describe a version for manifold complexes.

Our 2D representation maps each vertex to its link, represented as a cycle of the labels of its neighboring vertices. The cycle is ordered radially around the vertex in the orientation of the complex, e.g., clockwise. A findUp query on the ordered edge \((v_1, v_2)\) can be answered by looking up the link for \( v_1 \), finding \( v_2 \) in the link, and returning the next vertex in the link. A findUp on a vertex can be answered by selecting the first two vertices off of its link.

The link can be stored as a list of labels starting at an arbitrary point on the cycle. If the vertex has bounded degree, the lookup takes constant time. To analyze the space note that each edge appears in two cycles, and each appearance requires two pointers, one to the vertex label and one to the next element in the list. The total space is therefore 4 pointers/edge + 1 pointer/vertex. This is identical in space usage to the triangle-based structure, assuming that it also maintains a pointer from each vertex to one of its incident triangles. Our representation is similar to the half-edge structure since there are effectively two structures per edge, one pointing in each direction. It differs, however, in that there are no direct cross pointers between the matching half edges.

In 3D the representation maps a subset of all ordered edges to their link, represented as a cycle of vertex labels. The cycle is maintained in a consistent orientation, e.g., obeying a right-hand rule with respect to the order (direction) of the edge. The representative subset \( E' \) is selected to include only the edges \( \{v_1, v_2\} \) for which either the labels of \( v_1 \) and \( v_2 \) are both odd, or they are both even. Furthermore an edge is only stored in one of its two orders, chosen using a fixed rule, e.g., lower labeled vertex first. Since for any triangle (2-simplex) at least two labels have to be either odd or even, this sampling of the edges guarantees that
every triangle has at least one representative ordered edge in $E'$. The representation also needs to supply a way to access the link given the vertex labels of any edge in $E'$. This can be implemented using an adjacency list for each $v \in V$ of all outgoing representative edges $(v, v') \in E'$. Each element of the list stores $v'$ and a pointer to the link of $(v, v')$.

A `findUp` on an ordered triangle $(v_1, v_2, v_3)$ works as follows. It first finds a representative ordered edge $(v_a, v_b)$ from the triangle. Let’s call the third vertex on the triangle $v_c$. It looks up the link of $(v_a, v_b)$ in the adjacency list for $v_c$, and searches for $v_c$ in the link. If $(v_1, v_2, v_3)$ and $(v_a, v_b, v_c)$ have the same orientation (are an even permutation of each other) `findUp` returns the next vertex in the link, otherwise it returns the previous vertex in the link. A `findUp` on a vertex can be implemented by selecting any of its outgoing edges, and selecting the first two vertices of the edge’s link. A vertex, however, might have no outgoing edges in $E'$. For such a vertex $v$ the representation can store $(v_1, v_2)$ for any triangle $(v, v_1, v_2)$. The triangle can be used to find the tetrahedron. To support `findUp` on edges requires storing all edges (in one direction), but not necessarily their links. For edges not in $E'$ (i.e., odd-even edges), the representation needs only store a single vertex in their link.

To analyze the space for this representation we assume that the links of representative edges are stored as lists of vertices. Each list element has two pointers: one to the vertex and one to the next element in the list. For an edge $e \in E'$ there is a one-to-one correspondence between the triangles for which $e$ is a face and list elements in the link of $e$. Since a triangle has 3 edges, and on average half the edges will appear in $E'^2$, every triangle will contribute and average of $3 \times 0.5 \times 1.5$ list elements to the overall data structure. Since there are twice as many triangles as tetrahedrons, each tetrahedron will contribute an average of 3 list elements, which corresponds to 6 pointers. We also need to store the vertex adjacency lists for out-edges in $E'$. Each edge $(v_1, v_2) \in E'$ will appear as an element in one list $(v_1)$, and will require three pointers: one to $v_2$, one to the link of $(v_1, v_2)$, and one to the next element in the list. Additionally a pointer from each vertex to its list is required. The total space to support `findUp` on triangles is therefore $6 |T| + 3/2 |E| + |V|$. For a typical reasonably shaped mesh $|E| \approx 7/6 |T|$ and $|V| \approx 1/6 |T|$, giving approximately 8 pointers/tetrahedron. This is the same as the representation based on tetrahedrons.

The additional space to support `findUp` on vertices is trivial since most vertices already have an outgoing edge. To support `findUp` on edges, we need to separately store the excluded edges $(v_1, v_2)$ that are not in $E'$ in either direction. These can be stored off of $v_1$ using a linked list with 3 pointers per edge—one for $v_2$, one for some $v$ in the link of $(v_1, v_2)$, and one for the next pointer. This comes to about $3 \times 1/2 \times |E| = 7/6 \times 3/2 |T| = 7/4 |T|$. Many applications will not need `findUp` on edges, so in these cases this extra data need not be stored.

For manifolds with boundaries, the link might be a path of vertices instead of a cycle. We can simply keep the path starting at the first element. For pseudo-manifolds the link of singular vertices (2D) or edges (3D) can consist of a set of cycles and/or paths. We call this set the link set and it can be represented as multiple lists.

A $d$-simplex $s$ can be deleted by finding the representative $(d - 2)$-simplices that faces of $s$, and splitting a cycle or path of each of their links. For example, in Figure 1 when the triangle is deleted from (a) going to (b), the path for the link of vertex $v$ is split into two paths. Similarly the cycles for the other two vertices on the triangle are each split into a path. If splitting a link leaves the link set empty, then the $(d - 2)$-simplex is deleted. A $d$-simplex $s$ can be added by finding the representative $(d - 2)$-simplices of $s$, and extending each of their link sets. This extension might add a new path to the set (if neither of the two new vertices are in the set), it might extend an existing path (if one vertex is in the set), it might join two existing paths (if the two vertices are in separate paths), or it might joint a path into a cycle (if the two vertices are the ends of the same path). If the representation is restricted to manifolds with boundary, then the single path must either be extended by one (on either side), or jointed into a cycle.

Data can be added to the $d$ simplices (or $d - 1$ simplices) by adding a data field to each element of the link. Since a $d$ simplex will appear in multiple links, the data only needs to be stored on one of them (chosen in a fixed manner to make lookup easy). We make use of this in the compressed representation.

**Compressed Representation**: We first discuss how to compress the representation in 2D. Compression in 3D is similar. We make use of difference coding, in which each element in a vertex’s link is represented by its difference from the original vertex. If these differences are small, then a variable-length prefix code (such as the Gamma code of Elias [15]) can represent them efficiently. An additional sign bit can be added to allow for negative differences. To ensure that the differences are small, our algorithm relabels the vertices in a preprocessing phase which we will discuss later.
Once the vertices are relabeled, the link of a vertex can be represented by concatenating the code for its degree to the codes for the differences of its neighbors. (See Figure 2 for an example.) If a vertex has a link consisting of multiple cycles or paths (as can occur in a pseudomanifold), this link set can be represented by putting the cycles/paths one after the other with a count before each. If data is associated with some of the simplices, this can be interlaced with the codes for the neighbors. The resulting vertex encodings are stored in fixed-length blocks; if an encoding is larger than will fit in one block, multiple blocks may be formed into a linked list to hold the encoding. Our representation makes use of a hashing technique to minimize the size of the pointers used in these linked lists.

When the representation is queried, the code for the corresponding vertex is decompressed. When an update is made, the code for the corresponding vertices is decompressed, modified, and then compressed again.

Compression of a 3D representation is similar except that the representation stores the link around representative edges rather than around vertices. For each vertex the representation stores a list of that vertex’s representative out-edges, with pointers to the links of those out-edges. These pointers are compressed using the same hashing technique as above.

Generating Labels. If all the vertices are known before the algorithm begins, our algorithm can relabel them using a technique based on x-y cuts. Given a set of points, the technique first finds which of the x and y axes has the greatest diameter. It finds the approximate median in that coordinate and partitions the points on either side of that median. The points on one side are labeled first, then the points on the other side. This is done recursively to produce a labeling in which points that are near each other have similar labels. This is similar to a separator-based technique for graph compression through relabeling [4] except that it occurs before any edges have been added to the mesh.

If not all vertices are known before the algorithm begins, our algorithm can assign a sparse labeling to the initial vertices. When a new vertex is added, it is assigned a label that is close to the labels of its neighbors. It would be inefficient to allocate storage for every possible label; instead, our algorithm uses an extra level of indirection to map vertex labels to memory blocks.

6. IMPLEMENTATION

2D Triangulation. Our 2-dimensional compressed data structure is implemented as follows.

For difference encoding our structure uses the nibble code, a code of our own devising that stores integers using 4-bit "nibbles". Each nibble contains three bits of data and one "continue" bit. The continue bit is set to 0 if the nibble is the last one in the representation of an integer, and 1 otherwise. We find that this code is much faster than the gamma code while being almost as space-efficient.

It is sometimes necessary to store an extra bit b with a value v. This is accomplished with a shift operation: \( v' \leftarrow 2v + b \). In particular, if any value might be negative, our difference coder stores its absolute value plus a sign bit: \( v' \leftarrow 2|v| + \text{sign}(v) \).

A vertex is represented with a nibble code for the degree of the vertex, followed by nibble codes for the differences to each of the vertex’s neighbors. Our representation stores two additional “special-case” bits with each neighbor to provide information about the triangle that precedes it in the link. One bit is set to indicate a gap in the link set: it indicates that there is no triangle preceding that neighbor in the mesh. The other bit is set when data is associated with the triangle preceding that neighbor. In this case, the code for that neighbor is followed with a nibble code representation of the data.

As an optimization, note that for many vertices none of the special-case bits will be set. Our representation stores a bit with the degree of each vertex to indicate if none of its special-case bits are set; if this is so, those bits are omitted in the encoding of that vertex.

Our representation stores the nibble codes for each vertex in an array containing one seven-byte block per vertex. If a block overflows (that is, if the storage needed is greater than seven bytes), additional space
is allocated from a separate pool of seven-byte blocks. The last byte of the block stores a pointer to the next block in the sequence. Our representation uses a hashing technique to ensure that the pointer never needs to be larger than one byte. This requires a hash function that maps (address, i) pairs to addresses in the spare memory pool. Our representation tests values of i in the range 0 to 127 until the result of the hash is an unused block. It then uses that value of i as the pointer to the block. Under certain assumptions about the hash function, if the memory pool is at most 75% full, then the probability that this technique will fail is at most \( \frac{75^{128}}{127^{128}} \approx 10^{-16} \).

If the vertices are labeled sparsely (so that new labels can be generated dynamically), our representation also makes use of a hash mapping between labels and vertex data blocks. One byte of memory is allocated per label, if the label is in use, this byte contains a hash pointer to the first data block for that vertex.

One bit is stored with each block to indicate whether the current block is the last in the sequence. For the first block this bit is stored with the degree of the vertex; for subsequent blocks it is stored as the eighth bit of the one-byte pointer to that block.

There is a tradeoff in the sizes of the blocks used. Large blocks are inefficient since they contain unused space; small blocks are inefficient since they require space for pointers to other blocks. In addition, there is a cost associated with computing hash pointers by searching for unused blocks in the memory pool. Figure 3 shows the tradeoff between these factors for our Delaunay triangulation algorithm run on \( 2^{20} \) uniformly distributed points in the unit square. We chose a block size of 7 since it gives the most efficient use of space.

To improve the efficiency of lookups our representation use a caching system. When a query or update is made, the blocks associated with the appropriate vertex are decoded. The information is represented in uncompressed form as a list with one vertex in the link per element of the list. The lists are kept in a FIFO cache with a maximum capacity of 2000 nodes. Update operations may affect the lists while they are in the cache. The lists are encoded back into blocks when they are flushed from the cache.

### 3D Triangulation

Our 3-dimensional structure is implemented as a slight generalization of our 2-dimensional structure. Recall that our 3D representation keeps a map from each vertex \( v \) to all of its representative out-edges. This is stored as a difference coded list of the corresponding neighbors. The code for each neighbor \( v' \) is followed by a code for the number of nibbles in the encoding of the representative edge \((v, v')\), and a pointer to the first block containing the data for that edge. (The pointer is stored using the same hash trick as above to keep pointer sizes small.) Every representative edge has its own block allocated from the memory pool, with the capability to allocate additional blocks if needed.

When an edge is queried, our representation loads only the list for one vertex and for the edge itself into the cache. It does not need to decompress the other edges adjoining that vertex.

Since the number of nibbles needed per representative edge is quite variable, our data structure allocates from pools of 2, 4, 6, 8, or 10-byte blocks to reduce wasted space. The number of blocks in each pool was determined experimentally and is shown in Figure 4. The data structure ensures that each pool always has at least 10% free space; if a block cannot be allocated from a given pool, the data structure looks for a larger one. The initial block for each vertex comes from a separate array containing blocks of size 7.

### Dynamic point generation

To support dynamic point generation we use an expanded label space. If a total of \( n \) vertices are to be generated, we allow for \( 2^n \) possible labels. Each label receives a one-byte hash pointer which, if the label is in use, points to the initial

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Blocks Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.55n</td>
</tr>
<tr>
<td>4</td>
<td>1.3n</td>
</tr>
<tr>
<td>6</td>
<td>1.55n</td>
</tr>
<tr>
<td>8</td>
<td>1.3n</td>
</tr>
<tr>
<td>10</td>
<td>1.8n</td>
</tr>
</tbody>
</table>

**Figure 4:** The number of blocks of each size that are allocated for an \( n \)-vertex 3D mesh, and the percentage of blocks that were used for \( n = 2^{10}, 2^{15}, \) and \( 2^{20} \).
data block for the corresponding vertex. The initial vertices are spread evenly across the label space.

**Incremental Delaunay:** We implemented a Delaunay triangulation algorithm in two and three dimensions using our compressed data structure. We employ the well-known Bowyer-Watson kernel [8, 42] to incrementally generate the mesh. During the course of the algorithm a Delaunay triangulation of the current pointset is maintained. An incremental step inserts a new vertex into the mesh by determining the elements that violate the Delaunay condition. Those elements form the Delaunay cavity. The faces that bound the cavity are called the horizon. The mesh is modified by removing the elements in the cavity and connecting the new vertex to the horizon.

Each element (simplex of highest dimension) in the mesh has an associated list of all points in its interior that have not yet been added to the mesh as described in [19]. At each incremental step all points on cavity elements have to be reassocitated with new elements using lineside tests in 2D and planeside tests in 3D, which accounts for the dominant cost of the algorithm. We have carefully implemented the bulldozing idea described in [5] and extended it to three dimensions.

Our implementation does not require extra memory for the lists of points since at any time a point is either a vertex in the mesh or in one such list. The memory that will be used to store the vertex in the mesh can first be used as a list node.

The algorithm maintains a work queue of elements whose interiors contain points. When no elements contain points (i.e., all have been added to the mesh), the algorithm terminates.

In this scenario all points are known at the beginning. We relabel the input points using cuts along coordinate directions as described above. The runtimes reported in the next section include this preprocessing step.

**Delaunay Refinement:** To test our implementation's performance for the case when new points are dynamically generated at runtime, we implemented a 2D Delaunay refinement code in the style of Ruppert [34]. We augment a Delaunay triangulation by adding circumcenters of badly shaped triangles while maintaining the Delaunay property. When the initial triangulation is built we walk through the mesh once and check the quality of each element, queuing the ones not satisfying a preset minimum angle bound. The same work queue used in the triangulation phase of the algorithm is used to store the list of triangles to be split.

<table>
<thead>
<tr>
<th>Distribution</th>
<th># Pts</th>
<th># Extra Blocks</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>$2^3$</td>
<td>70823</td>
<td>3.16</td>
</tr>
<tr>
<td>normal</td>
<td>$2^4$</td>
<td>72239</td>
<td>3.52</td>
</tr>
<tr>
<td>kuzmin</td>
<td>$2^5$</td>
<td>72917</td>
<td>4.36</td>
</tr>
<tr>
<td>line</td>
<td>$2^6$</td>
<td>66297</td>
<td>3.64</td>
</tr>
<tr>
<td>uniform</td>
<td>$2^{20}$</td>
<td>288255</td>
<td>13.25</td>
</tr>
<tr>
<td>normal</td>
<td>$2^{20}$</td>
<td>292580</td>
<td>14.41</td>
</tr>
<tr>
<td>kuzmin</td>
<td>$2^{20}$</td>
<td>292709</td>
<td>21.34</td>
</tr>
<tr>
<td>line</td>
<td>$2^{20}$</td>
<td>276124</td>
<td>15.86</td>
</tr>
</tbody>
</table>

**Figure 5:** The number of extra 7-byte blocks needed to store triangular Delaunay meshes for various point distributions using our structure and the runtime of our 2D implementation.

Whenever a new point $p$ is generated the algorithm assigns as new label by considering the horizon vertices $H$ of the cavity created by $p$ and calculating the value $v$ that minimizes the sum of the log norms to $H$. It then finds the closest label to $v$ that is not yet used.

In the pure triangulation code, all vertices are known at the beginning, so we can store the point coordinates and the first level vertex array densely. In the refinement code we can only fill these arrays up to about 85% before the open address hashing takes prohibitively long. We also require extra memory for the additional map form the label space to the vertices.

7. **EXPERIMENTS**

We report experiments on a Pentium 4, 2.4 GHz system, running Red Hat Linux Kernel 2.4.18, GNU C/C+ compiler version 3.0.1. For all geometric operations (lineside, planeside, incircle, and insphere tests) we use Shewchuk's adaptive precision geometric predicates [37]. We use single-precision floating-point numbers to represent the coordinates.

**2D Delaunay:** We tested our 2D implementation on data drawn from several distributions to assess its memory needs for non-uniform data sets. We ran tests on the following distributions: Uniformly random, normal, kuzmin, and a line singularity. Details on these distributions can be found in [6]. In Figure 5 we report the number of extra (overflow) 7-byte blocks used to store Delaunay meshes of various point distributions and the runtime of our implementation. It can be seen that the runtime varies by about 40% while the number of extra blocks varies by about 10%. Furthermore the number of extra blocks used comes to only about 28% of the number of default blocks needed, which is one per vertex. In our experiments we set the number of extra blocks available to 35% of the number of default blocks. The extra blocks therefore fill
Figure 9: The number of bytes needed for occupied blocks to store tetrahedral Delaunay meshes for various point distributions and the runtime of our 3D implementation.

Figure 6: Runtime in 2D, uniformly random points

Figure 7: Memory use in 2D, uniformly random points

Figure 8: Breakdown of memory use in 2D, uniformly random points

to about 80% of capacity. Given this setting, the total space we require for the mesh is 1.35 x 7 bytes/vertex, which is 4.725 bytes/triangle.

Next, we compare runtime and memory usage of our implementation to Shewchuk's Triangle [36] code which is the most efficient code reported by Boissonnat et. al. [7]. In Figure 6 we report the runtime of our (incremental) code vs. Triangle's divide-and-conquer and its incremental implementation. We report the total memory use of both codes in Figure 7 and break down our memory use for the simplicial mesh, point coordinates and the work queue in Figure 8. While using just about a third of the memory our code runs about 10% slower than Triangle's divide-and-conquer implementation and is about an order of magnitude faster than Triangle's incremental implementation. In our code 50% of the memory is used to represent the mesh, 40% to store the coordinates, and 10% for the work queue.

3D Delaunay: As in 2D we tested our 3D implementation on the same four point distributions. In our 3D structure we allocate memory blocks of different size. To compare the memory needs for various point distribution, we report the number of bytes used to store occupied blocks in Figure 9. As in 2D the runtimes differ, but the memory needed is nearly independent of the distribution.

We compare our 3D implementation with uniform random data to Shewchuk's Pyramid code [35][36]. Figures 10 and 11 show the runtime and the memory usage. Figure 12 breaks down the memory usage of our code.

In comparison our implementation runs slightly faster and uses only about one third of the memory. In 3D the representation of the mesh uses about 75% of the

3We note that the version of Pyramid we are using is a Beta release.
total memory; point coordinates and work queue account for 18% and 7%, respectively.

**2D Delaunay refinement:** We tested our 2D Delaunay refinement code and compare runtime and memory use to our pure 2D Delaunay code, see Figures 13 and 14. The Figures show problem size in terms of the final number of elements in the mesh. In the pure Delaunay code, all \( n \) points are known initially, whereas in the refinement code only \( n/2 \) points are known initially, the other \( n/2 \) are generated and labeled on the fly as described in Section 6. We refine the mesh up to a minimum angle of 26.85°.

The runtimes for the two versions are almost identical. We need about 30% more memory in the refinement
code. Additional memory is needed for the map from labels to vertices and for slack in the point coordinate array and the first level vertex array needed for our hashing technique.

8. DISCUSSION

The representation we described can be used as an alternative to external memory (out-of-core) representations, when the mesh is within a factor of five or so of fitting in memory relative to a standard representation. Our representation has the advantage that it allows random access to the mesh without significant penalty; and can therefore be used as part of standard in-memory algorithms (or even code) by just exchanging the mesh interface.

In conjunction with external memory. For very large problems our representation can be used in conjunction with external-memory techniques. Since in our representation the ordering of the vertices is designed to be local (it is based on the quad/oct tree decomposition), and the blocks of memory for vertices are laid out in this ordering nearby vertices in the mesh will most likely appear on the same page. One problem is that if the data for a vertex overlaps we now assign the overlap data to the extra blocks using a hash, which has no locality. To make sure that the overlap data has some spatial locality one could be more careful about assigning the extra blocks (e.g., preferentially within the same page as the original block). Based on this representation, algorithms that have a strong bias to accessing the mesh locally (e.g., see the recent work of Amenta, Choi and Rote [1]) will tend to have good spatial locality and work well with virtual memory when it does not fit into physical memory.

Generalizations to d-dimensions. The idea of storing the link of every $d - 2$ dimensional simplex generalizes to arbitrary dimension. The compression technique also generalizes to arbitrary dimension, but is likely to be ineffective for large dimensions. This is because the size of the difference codes depends on the separator sizes [4], which in turn depends on the dimension. Choosing an effective way to select the representative subset of the $d - 2$ dimensional simplices will depend on the dimension and would need to be considered to use our representation on dimensions greater than three. We have not done any experimentation to analyze the effectiveness of our techniques on dimensions greater than three, or to compare our representations to other representations.

ACKNOWLEDGEMENTS

We are grateful to Jonathan Shewchuk for commenting on the paper and letting us use a pre-release version of Pyramid. This work was done as part of the Sangria [23] project, and several project members have contributed ideas.

References


A MESH WARPING ALGORITHM BASED ON WEIGHTED LAPLACIAN SMOOTHING

Suzanne M. Shontz† Stephen A. Vavasis‡

†Center for Applied Mathematics
Cornell University, Ithaca, NY 14853
shontz@cam.cornell.edu.

‡Department of Computer Science
Cornell University, Ithaca, NY 14853
vavasis@cs.cornell.edu.

ABSTRACT

We present a new mesh warping algorithm for tetrahedral meshes based upon weighted laplacian smoothing. We start with a 3D domain which is bounded by a triangulated surface mesh and has a tetrahedral volume mesh as its interior. We then suppose that a movement of the surface mesh is prescribed and use our mesh warping algorithm to update the nodes of the volume mesh. Our method determines a set of local weights for each interior node which describe the relative distances of the node to its neighbors. After a boundary transformation is applied, the method solves a system of linear equations based upon the weights to determine the final position of the interior nodes. We study mesh invertibility and prove a theorem which gives sufficient conditions for a mesh to resist inversion by a transformation. We prove that our algorithm yields exact results for affine mappings and state a conjecture for more general mappings. In addition, we prove that our algorithm converges to the same point as both the local weighted laplacian smoothing algorithm and the Gauss-Seidel algorithm for linear systems. We test our algorithm’s robustness and present some numerical results. Finally, we use our algorithm to study the movement of the canine heart.

Keywords: moving meshes, optimization-based mesh smoothing, unstructured mesh generation, tetrahedral meshes, cardiology

1. INTRODUCTION

Moving meshes arise in cardiology, computer graphics, animation, and crash simulation, among other applications in science and engineering. With moving meshes, the mesh is updated at each step in time due to a moving domain boundary, thus resulting in potentially drastically varying mesh quality from step to step. One problem that can occur at each timestep is element inversion. We focus on maintaining good-quality tetrahedral meshes throughout the mesh warping process in this paper.

It is well-known that poor quality elements affect the stability, convergence, and accuracy of finite element and other solvers because they result in poorly conditioned stiffness matrices [1]. If well-shaped elements are not the result of updating the mesh boundary, the mesh quality must be improved by topological or geometrical means after each time step.

Research has shown that mesh smoothing (or r-
reﬁnement) methods can be applied to improve the quality of a mesh. These methods adjust the positions of the vertices in the mesh while preserving its topology.

Laplacian smoothing is the most popular method for node-based mesh smoothing. In an iterative manner, it repositions the vertices of the mesh by moving each interior node to the geometric center of its neighbors. It is often used because it is computationally inexpensive and is very easy to implement. However, the method has several undesirable properties. One of them is that the method is not guaranteed to work, i.e., sometimes it inverts mesh elements. A second drawback is that the resulting mesh depends upon the order the nodes are smoothed.

A related type of smoothing, namely Winslow smoothing, is more resistant to mesh folding due to the requirement that the logical variables be harmonic functions. See [2] for more details on Winslow smoothing.

Other, more accurate methods for r-refi nement are possible. Most of these methods are based upon optimization. Optimization-based methods are used with the goal of guaranteeing an improvement in the mesh quality by minimizing a particular mesh quality metric. Their main drawback, however, is their computational expense. Examples of optimization-based methods for r-refi nement can be found in the following papers: [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], and [14]. For a theory of algebraic mesh quality metrics see [15].

To address the above issues, Baker [16] developed a three-step method for the metamorphosis of tetrahedral meshes. Each cycle of the method involves a combination of r-refi nement, mesh coarsening, and mesh enrichment to adapt the mesh. The ﬁrst step in the cycle is to move the interior nodes as far as possible using r-refi nement while avoiding element inversion. The second step is to remove the poorly shaped elements in the mesh using mesh coarsening. The ﬁnal step is the addition of elements to improve the mesh quality by mesh reﬁnement. This dynamic procedure was shown to be more cost-effective than just r-refi nement. One disadvantage of this technique is that it comes with no theoretical guarantees as it is diﬃcult to analyze because each cycle is a combination of three very diﬀerent techniques.

We study a diﬀerent mesh warping problem where the connectivity of the mesh is not allowed to change, which is important for some applications. Seeking simplicity, preservation of the mesh’s combinatorial structure, theoretical guarantees, and low computational expense, we developed a linear weighted laplacian smoothing (LWLS) method for mesh warping in two and three dimensions. A full description of the algorithm is given in Section 2. In Section 3, we prove a theorem that gives suﬃcient conditions for when a mesh is able to resist inversion by a transformation. We also study mesh inversion within the context of our algorithm. In particular, we prove that our algorithm yields exact results for affine boundary transformations, and we state a conjecture for more general mappings. In Section 4, we show that our algorithm converges to the same point as both the local version of weighted laplacian smoothing and Gauss-Seidel. In Section 5, we test our algorithm on several types of mesh deformations. In Section 6, we apply our algorithm to study the motion of the beating canine heart under normal conditions. In Section 7, we summarize our work and identify directions for future research.

2. LINEAR WEIGHTED LAPLACIAN SMOOTHING

The problem that we address is as follows: Given a 3D domain, bounded by a triangulated surface mesh, and given an interior volume mesh composed of unstructured tetrahedra, suppose the triangulated surface mesh is displaced. Is there an algorithm to move the nodes of the volume mesh so that it continues to conform to the surface mesh and to be a good quality mesh?

Our mesh warping algorithm to address this question is based upon weighted laplacian smoothing. The ﬁrst step in the algorithm is to generate a set of local weights for each interior node that represent the relative distances of the node to each of its neighbors. We use an interior point method from nonlinear programming in order to generate these weights. Next we apply a transformation to the boundary nodes. Using these new positions for the boundary nodes and the sets of weights from the original mesh, we solve a system of linear equations to determine new positions for the interior nodes. We now give a more detailed description of this algorithm.

Here we describe a nonlinear programming method for computing the weights for 2D meshes. Note that our method can be extended to 3D in the straightforward manner; however, we have not done extensive testing of our 3D implementation. In addition, this paper analyzes primarily the 2D version.

Let \((x_i, y_i)\) denote the \(x\) and \(y\)-coordinates respectively of the \(i\)th interior node in the initial mesh. In addition, let the \(x\) and \(y\)-coordinates of its adjacent vertices be given by \(\{(x_j, y_j) : j \in N_i\}\), where \(N_i\) denotes the set of neighbors of node \(i\).

In order to ﬁnd the set of weights \(w_{ij}\), where \(w_{ij}\) is the weight of node \(j\) on interior node \(i\), we use the log barrier function from linear programming to formulate the following optimization problem for each \(i\):
The use of convex optimization in mesh smoothing is not a new idea. Amenta et al. developed a framework for formulating mesh smoothing problems as quasi-convex programs so that they can be optimized as generalized linear programming problems [9]. In addition, Freitag et al. developed methods for local mesh smoothing and untangling using mesh quality metrics with convex function levels sets in [7] and [8].

By starting with an initial feasible point in the interior of the space of all possible weights, we are able to solve (1)-(5) using the Projected Newton Method. This method can be applied to solve optimization problems of the following form:

\begin{align*}
\max_{w_{ij}, j \in N_i} & \quad \sum_{j \in N_i} \log(w_{ij}) \\
\text{subject to} & \quad w_{ij} > 0, \\
& \quad \sum_{j \in N_i} w_{ij} = 1, \\
& \quad x_i = \sum_{j \in N_i} w_{ij} x_j, \\
& \quad y_i = \sum_{j \in N_i} w_{ij} y_j
\end{align*}

We note that the objective function together with the constraints form a strictly convex optimization problem for which there is a unique optimum. The optimum can thus be found by an interior point method.

The optimality conditions for the above problem are:

\begin{align*}
\nabla f(x) - A^T \lambda &= 0 \\
b - Ax &= 0.
\end{align*}

Here \( \lambda \) denotes the vector of Lagrange multipliers. Note that this is a system of nonlinear equations in \( x \) and \( \lambda \); thus Newton’s method can be used to solve for \( x \) and \( \lambda \). However, if the initial point is feasible (i.e., \( b - Ax = 0 \)) then \( p_k \) must satisfy \( Ap_k = 0 \) where \( x_{k+1} = x_k + p_k \), i.e., \( p_k \) is the Newton direction at iteration \( k \). This implies that \( p_k = Zv_k \) for an \((n - m)\)-dimensional vector \( v_k \), where \( Z \) is a basis for the null space of \( A \). It is easy to see that the resulting equation for \( p_k \) is given by

\[ p_k = -\left( H^{-1} - H^{-1} A^T \left[ AH^{-1} A^T \right]^{-1} AH^{-1} \right) \nabla f(x_k), \]

where \( H = \nabla^2 f(x_k) \) [17].

In order to compute an initial feasible point in the interior of the solution space, we choose three of the interior node’s adjacent vertices and write \((x, y)\) as a convex combination of the positions of the three nodes. This yields three positive weights; call them \( w_1, w_2, \) and \( w_3 \). In order to find a set of \( n \) positive weights, the rest of the \( w_i \) are initialized to \( \epsilon \), a small positive constant. Then once \( \epsilon \) has been specified, linear equations (3) through (5) can be solved for new values of \( w_1, w_2, \) and \( w_3 \). For small values of \( \epsilon \), all of the constraint equations above will be satisfied (using the new values for the weights). An inexact linesearch is used to make \( \epsilon \) as large as possible so that \( w_1, w_2, \) and \( w_3 \) are still reasonably sized, i.e., each of them should be greater than some small tolerance.

Once the weights have been generated, the input surface deformation is applied to move the boundary nodes to new locations. The final step is to use the positions of the boundary nodes and the sets of weights to simultaneously determine the final positions for the interior nodes by solving a linear system of equations. The linear equations that must be solved are the following:

\begin{align*}
\sum_{j \in N_i} w_{ij} x_j &= x_i, \quad (9) \\
\sum_{j \in N_i} w_{ij} y_j &= y_i. \quad (10)
\end{align*}

The resulting linear systems from (9) and (10) are both \( m \times m \), where \( m \) is the number of interior nodes.

We first prove that the resulting linear system has a unique solution.

**Theorem 1** The linear system (11), which expresses the position of each interior node as a convex combination of its neighbors, has a unique solution.

**Proof.**

We first establish some necessary notation and terminology. Let \( b \) and \( m \) represent the numbers of boundary and interior nodes, respectively. Next, define \( x_B \) and \( y_B \) to be vectors of length \( b \) that contain the initial \( x \)- and \( y \)-coordinates of the boundary nodes. Similarly, let \( x_I \) and \( y_I \) be vectors of length \( m \) that contain the initial \( x \) and \( y \)-coordinates of the interior nodes. Then \([x_B, y_B]\) and \([x_I, y_I]\) contain the original positions of the boundary and interior nodes respectively. Next, define the weighted Laplacian matrix, \( L \), for a weighted graph \( G(V; E; w) \) as follows:

\[ L(i, j) = \begin{cases} 
-w_{ij} & \text{if } i \neq j, \\
\sum_{k \in V} w_{ik} & \text{if } i = j,
\end{cases} \]

where \( w_{ij} = 0 \) if \( (i, j) \notin E \). The boundary nodes are assumed to be numbered last.
Denote by $A = [A_I, A_B]$ the matrix that is derived from the weighted Laplacian matrix in LWLS by deleting its last $b$ rows. Note that $A_I$ contains all of the weights corresponding to the interior neighbors and is $m \times m$. In addition, $A_B$ contains all of the weights corresponding to the boundary neighbors and is $m \times b$. Linear systems (9)–(10) then can be expressed

$$A_I[x_I, y_I] = -A_B[x_B, y_B].$$

(11)

To argue that we have uniqueness when LWLS is used to generate the weights, we classify $A_I$ as an $M$-matrix which is defined as follows:

Definition: An $n \times n$ real matrix $A$ is said to be an $M$-matrix if $a_{ij} \leq 0$ for all $i \neq j$ and $A^{-1} \geq 0$.

To see that $A_I$ is an $M$-matrix, we make use of the following characterization theorem:

**Theorem 2** [18] If $A \in \mathbb{R}^{n \times n}$ satisfies $a_{ii} > 0$, $a_{ij} \leq 0$ for all $i \neq j$, is weakly row diagonally dominant (i.e., $Ae \geq 0, e \neq 0$, where $e$ is the vector of all 1’s), and is irreducible, then $A$ is an $M$-matrix.

Because the mesh is connected and a positive weight is associated with each edge, we see that $A_I$ is irreducible. Because the diagonal entries of $A_I$ are 1, and the off-diagonal are negative and sum to a number in $[-1, 0]$, we see that $A_I$ is diagonally dominant. Therefore, $A_I$ satisfies the definition of an $M$-matrix by the above argument [19]. Thus, $A_I$ is invertible, and (11) has a unique solution.

Because the above system has a unique solution, it is next solved via Gaussian elimination. Because the matrix is weakly diagonally dominant, partial pivoting is not necessary. This is a major advantage, since sparse matrix algorithms are much more efficient without pivoting. The sparsity structure is apparent, since, on average, an interior node has 6 neighbors in 2D, whereas a typical 2D mesh may have hundreds, thousands, or even millions of nodes. In addition, because the nonzero pattern is symmetric, we can first apply the symmetric minimum degree ordering to the appropriate matrix in an attempt to improve the speed of the computations. GMRES could also be used to solve the linear system instead of Gaussian elimination in an attempt to further improve the algorithm’s speed although we have not tried this.

Another big advantage of our method is that if a continuous deformation of the boundary is given, then LWLS is a valid algorithm for computing the resulting trajectory that specifies the movement of the interior nodes. In addition, these trajectories will be continuous. This is vital for some applications where continuity of motion is required.

Consider the following application of the LWLS Method. Figure 1 shows the three steps of the mesh warping process. Note that the original mesh is shown in the top graph. A rather large deformation is then applied to the boundary, and the result is shown in the middle graph. Finally, the interior nodes are moved to new positions as is shown in the bottom graph. The final mesh is a valid mesh, that is, it has no inverted elements.

![Figure 1: The three stages of the LWLS algorithm [20.]](image)

These figures show that the method demonstrates promise for use as both a mesh untangler and a smoother in the context of the mesh warping problem. However, the method will not always be successful. If the boundary itself becomes tangled (self-intersecting) under the deformation, then there is no possibility that LWLS could recover an untangled mesh. But even if the boundary is not tangled, LWLS may still fail as indicated by our computational experiments below.

Because LWLS is not guaranteed to work for all types of boundary transformations, we seek to determine exactly what conditions will guarantee that LWLS will
be successful. In order to meet this goal, we first prove a theorem about mesh invertibility and then give a conjecture (with supporting theory) on sufficient conditions for LWLS to be successful on the mesh warping problem.

3. THEORY OF MESH INVERTIBILITY

In this section we consider sufficient conditions to guarantee that elements are not inverted in the mapped mesh. The material in this section focuses on the 2D case, but we also indicate results that extend to 3D. First, we prove a theorem about mesh invertibility in the case when a transformation \( f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) is applied to all the nodes. Then we conjecture that a similar theorem holds in the case when \( f \) is applied only to the boundary and the LWLS algorithm is used to position interior nodes (i.e., the situation under consideration). We provide some partial analysis in support of the conjecture.

In order prove Theorem 3 below, we first prove the following useful lemma, which has appeared in other forms in the previous literature (e.g., [21]). This lemma extends to 3D as well if “angle” in 3D means solid angle.

Lemma 1 Let \( T \) be a triangle with vertices \( v_1, v_2, v_3 \in \mathbb{R}^2 \). Let \( A(T) = [v_2 - v_1; v_3 - v_1] \), and define the aspect ratio of \( T \), \( a(T) \), by \( \frac{a}{b} \), where \( a \) is the minimum angle given in radians. Then, \( a(T) \leq \kappa(A(T)) \leq 16a(T) \).

Sketch of Proof: First, we argue that without loss of generality, we can assume that the smallest angle of \( T \) is at \( v_1 \) and that \( \frac{v_2 v_1}{\cos \theta} \) is longer than \( \frac{v_2 v_3}{\cos \phi} \). The second step in the proof is to compute \( \|A(T)\| \) and \( \|A(T)^{-1}\| \) and show that the quantities are related to the area and edge lengths of \( T \). Throughout the proof we use \( \| \cdot \|_\infty \) for our choice of norm. The result is that \( \|A(T)^{-1}\| \leq \frac{\sqrt{2}}{\min \text{area} v_1 v_2 v_3} \max \text{length} \), where \( \max \text{length} \) is the length of the longer of the edges \( v_1 v_2 \) and \( v_2 v_3 \). Similarly, it can be shown that \( \|A(T)\| \leq 2\sqrt{2} \max \text{length} \). Putting this together, we obtain that \( \kappa(A(T)) \) is bounded above by \( \frac{2\sqrt{2} \max \text{length}}{\min \text{area} v_1 v_2 v_3} \). Similarly, it can be shown that \( \kappa(A(T)) \) can be bounded below by \( \frac{1}{\sqrt{2} \min \text{area} v_1 v_2 v_3} \). Recall the product of an edge with its altitude is always twice the area of \( T \). Using the relationships between side-lengths, we can rewrite our bounds as \( \frac{1}{\sqrt{2} \min \text{area} v_1 v_2 v_3} \leq \kappa(A(T)) \leq \frac{1}{\min \text{area} v_1 v_2 v_3} \). Now define \( \theta \) to be the angle at \( v_1 \). Because we assumed that the smallest angle is at \( v_1 \), the longest edge is \( v_2 v_3 \). Therefore, \( \frac{1}{\min \text{area} v_1 v_2 v_3} \leq \kappa(A(T)) \leq \frac{1}{\min \text{area} v_1 v_2 v_3} \). Note that \( 0 < \theta \leq \frac{\pi}{2} \) because \( \theta \) is the smallest angle. Using basic calculus, we see that for \( \theta \in (0, \frac{\pi}{2}] \), \( \frac{1}{\sin^2 \theta} \leq \frac{1}{\sin^2 \theta} \leq \frac{1}{\min \text{area} v_1 v_2 v_3} \). Therefore, \( a(T) \leq \frac{1}{\sin^2 \theta} \leq \kappa(A(T)) \leq \frac{1}{\min \text{area} v_1 v_2 v_3} \leq \frac{1}{\min \text{area} v_1 v_2 v_3} = 16a(T) \), which proves the lemma.

We are now ready to prove a theorem about mesh resistance to invertibility in the case that a transformation is applied to all nodes. This theorem is stated for 2D but extends to 3D.

Theorem 3 Suppose that \( f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) is bijective and differentiable on the entire mesh, with \( f \) nonsingular. Suppose \( f \) is applied to all nodes of the mesh. Then the triangles in the mesh are not flipped by \( f \) if inequalities (14) - (16) below hold.

Proof.

For simplicity, assume for the rest of this section that \( \det(f') > 0 \) on the whole mesh since the case \( \det(f') < 0 \) is handled symmetrically. We give sufficient conditions for the noninvertibility of the mesh elements. Let \( t \) be a triangle in the triangulation \( T \) with vertices \( v_1, v_2, \) and \( v_3 \).

Note that \( t \) is flipped if \( f \) implies

\[
\begin{vmatrix}
(v_2 - v_1)_x & (v_3 - v_1)_x & (v_3 - v_2)_x \\
(v_2 - v_1)_y & (v_3 - v_1)_y & (v_3 - v_2)_y \\
(f(v_2) - f(v_1))_x & (f(v_3) - f(v_1))_x & (f(v_3) - f(v_2))_x
\end{vmatrix} < 0. \tag{12}
\]

Using Taylor’s Theorem, we obtain:

\[
f(v_2) = f(v_1) + J_{v_1}(v_2 - v_1) + \frac{1}{2} v \left( f_1^{\perp \perp} (\epsilon, \mu) \right) + \frac{1}{2} v \left( f_2^{\perp \perp} (\epsilon, \mu) \right),
\]

where \( v \) is the line connecting \( v_1 \) and \( v_2 \).

Substituting the above into (12) we obtain an inequality of the following form giving necessary and sufficient conditions for invertibility:

\[
\begin{vmatrix}
1 & a_{11} & a_{12} \\
a_{21} & a_{11} + \gamma_1 & a_{12} + \gamma_2 \\
a_{22} & a_{21} + \gamma_3 & a_{22} + \gamma_4
\end{vmatrix} < 0. \tag{13}
\]

So, Lemma 2.7.1 in [22] applies to (13) once we have sufficient conditions for the hypotheses to hold. Note that the portion of the lemma we used states that if \( A \) is invertible, if \( M = A + \delta A \) satisfies \( \|A\| \leq \epsilon \|A\| \), and if \( \kappa(A) < 1 \), then \( M \) is nonsingular. Here \( \kappa(A) \) denotes the condition number of \( A \).

Using algebra and Lemma 1 above, we find that the following conditions ensure the hypotheses of the above lemma are satisfied:
\[
\frac{1}{J_{v_1}} \frac{\partial^2 f_1}{\partial x^2}(e) \leq \frac{2}{51} \frac{h_{\text{min}}}{a(T) h_{\text{max}}^2} \tag{14}
\]
\[
\frac{1}{J_{v_1}} \frac{\partial^2 f_1}{\partial x \partial y}(e) \leq \frac{1}{51} \frac{h_{\text{min}}}{a(T) h_{\text{max}}^2} \tag{15}
\]
\[
\frac{1}{J_{v_1}} \frac{\partial^2 f_1}{\partial y^2}(e) \leq \frac{2}{51} \frac{h_{\text{min}}}{a(T) h_{\text{max}}^2} \tag{16}
\]

where \( h_{\text{min}} \) and \( h_{\text{max}} \) are the smaller and larger altitudes (respectively) from \( v_2 \) and \( v_3 \). Similar conditions are obtained for \( f_2 \).

Note that this proof generalizes to higher dimensions and is thus applicable to unstructured mesh generation in 3D.

Next, we state our conjecture for the LWLS method, which is formally reminiscent of Theorem 3. After stating the conjecture, we provide some supporting theory.

**Conjecture 1** Suppose that \( f : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) is bijective and differentiable on the entire mesh, with \( f \) non-singular. Consider transforming the mesh by applying \( f \) only to the boundary nodes of the triangulation, and then using LWLS to position the interior nodes. Then no triangles are flipped provided that an upper bounding of the first derivative and mesh aspect ratio is satisfied.

The condition in the conjecture is analogous to the hypotheses in Theorem 3, namely, a condition for invertibility that relates the first and second derivatives of \( f \). There is a qualitative difference, however, between the theorem and the conjecture: The theorem states that as long as \( f \) is bijective, then for a sufficiently refined mesh (with bounded aspect ratio), there will be no inversions. On the other hand, the conjecture and supporting theory here apparently indicate that the condition for occurrence of inversions in the LWLS algorithm depends on \( f \) and the mesh aspect ratio but not so much on the level of refinement.

In considering how to prove the conjecture, let us fix a particular triangle \( T \) in the original mesh’s interior, centered at \((x, y)\), with vertices \( v_1, v_2, v_3 \). As noted above, the positions of the nodes in the updated mesh are obtained from solving \( A_T \tilde{x} = A_B \tilde{y} \), for \([\tilde{x}], [\tilde{y}]\) given that \([x_B], [y_B]\) are obtained from the original \([x_B], [y_B]\). In fact, each two-element row of \([x_B], [y_B]\) is \( f \) applied to the corresponding row of \([x_B], [y_B]\). Let \( W_T \) be the \( 2 \times m \) matrix with 1’s in the \((1, v_1)\) positions and -1’s in the \((1, v_2)\) and \((2, v_3)\) positions. Then \( W_T \) has \( 2 \times 2 \) matrix, call it \( A(T) \), whose determinant must be positive to avoid inversion. Observe that

\[
A(T) = -W_T A_T^{-1} A_B [\tilde{x}_B, \tilde{y}_B], \tag{17}
\]

Next, expand \( f \) as a Taylor series about \((x, y)\), and for simplicity assume \( f \) is a quadratic function. The Taylor expansion therefore has the form

\[
f(x, y) = f(x, y) + D_x f(x, y) (x - x) + D_y f(x, y) (y - y) + \frac{1}{2} D_{xx} f(x, y) (x - x)^2 + D_{xy} f(x, y) (x - x)(y - y) + \frac{1}{2} D_{yy} f(x, y) (y - y)^2.
\]

We can write the first three terms of this expansion as \( v^T + [x, y]L \), where \( L \) is a \( 2 \times 2 \) matrix and \( v^T \) is 2-vector, i.e., in the general form for an affine mapping, where \( L \) and \( v^T \) depend on the choice of triangle \( T \). (The dependence on \( T \) is not explicitly denoted; the superscript \( T \) on \( v \) indicates “transpose.”) Let us write the remaining terms as \( Q((x, y) - (x, y)) \), where \( Q \) is a pure quadratic function also depending on \( T \). Thus,

\[
f(x, y) = v^T + [x, y]L + Q((x, y) - (x, y))
\]

We substitute this into (17) to obtain

\[
A(T) = -W_T A_T^{-1} A_B [\tilde{x}_B, \tilde{y}_B] = ev^T + [x_B, y_B]L + \left[ Q((x_B, y_B) - (x, y)) \right]. \tag{18}
\]

where \((x_B, y_B), \ldots, (x_B, y_B)\) is an explicit enumeration of the nodes of \([x_B], [y_B]\). We can analyze the sum of the first two terms on the right-hand side explicitly using the following lemma.

**Lemma 2** Suppose the boundary transformation applied to the initial mesh is an affine mapping. Then if LWLS is used to reposition the interior nodes, the resulting mesh is the same as if the affine mapping were applied to all nodes in the initial mesh.

This lemma extends to 3D as well.

**Proof.**

Continuing to use the above notation, we note that the position of the interior nodes in the deformed mesh is given by

\[
[x], [y] = -A_T^{-1} A_B [x_B, y_B][L^T + ev^T]. \tag{19}
\]

In order to show that affine mappings yield exact results with LWLS, we want to show that (19) is the same as:

\[
[x], [y] = [x], [y][L^T + ev^T]. \tag{20}
\]

Observe that if (20) is the same as (19), then it will follow that

\[
A_T[x], [y][L^T + ev^T] = -A_B [x_B, y_B][L^T + ev^T]. \tag{21}
\]
Thus, it remains to check that (21) holds.

Because the weights for each interior node sum to 1, $Ae = 0$; i.e., $A_I e + A_B e = 0$. Hence $(A_I e + A_B e)v^T = 0$. Also, because $[x_I, y_I]$ and $[x_B, y_B]$ denote the original positions of the nodes, we know that $A_I[x_I, y_I] + A_B[x_B, y_B] = 0$. So, $(A_I[x_I, y_I] + A_B[x_B, y_B])L^T = 0$.

Putting these together, we see that

\[(A_I[x_I, y_I] + A_B[x_B, y_B])L^T + (A_I e + A_B e)v^T = 0. \tag{22}\]

Therefore, (21) holds, and the lemma is proven. $\blacksquare$

For an example of using LWLS in conjunction with an affine boundary transformation, see Figure 2.

![Figure 2: Exact results from LWLS for affine mapping [20].](image)

Substituting the result of this lemma into (18) and using the fact that $W_T e = 0$ (by definition of $W_T$) yields

\[A(T) = W_T[x_I, y_I]L - W_TA_I^{-1}A_B \begin{bmatrix} Q((x_B, y_B) - (x_I, y_I)) \\ \vdots \\ Q((x_B, y_B) - (x_I, y_I)) \end{bmatrix}. \tag{23}\]

Observe that this equation is in a form analogous to the proof of Theorem 3: the matrix whose determinant we must take is written as the product of a determinant in the original mesh (namely $W_T[x_I, y_I]$) by the first derivative of $f$ locally (namely, $L$) plus a quadratic term. We now wish to argue that the linear term dominates the quadratic term under suitable assumptions. Let us write the quadratic term as $q_I + \cdots + q_k$, where $q_i$ denotes the contribution from the $i$th boundary node. In other words, $q_i = -W_TA_I^{-1}A_B(Q((x_B, y_B) - (x_I, y_I))e_i)$ where $e_i$ denotes the $i$th column of the identity matrix. Fix $l$ and let $r = -A_I^{-1}A_B e_l$, so that $A_I r + A_B e_l = 0$ and

\[|q| \leq ||W_T r|| \cdot |Q((x_B, y_B) - (x_I, y_I))|. \tag{24}\]

If all the weights were equal and the mesh were uniform, the equation $A_T r + A_B e_l = 0$ would be a discretization of Laplace’s equation on the mesh domain, in which the boundary condition is a Dirichlet condition of 1 at a single boundary node (node $l$) and 0 at all other boundary nodes. The quantity under consideration is $W_T r$, which (by definition of $W_T$) is a 2-vector each of whose entries is a finite difference of two entries of $r$ at neighboring mesh nodes of triangle $T$ centered at $x_I, y_I$. Therefore, we analyze this quantity by considering the model problem of solving Laplace’s equation on the unit disk in which a Dirichlet boundary condition of 1 is imposed on boundary segment $S$ of length $h$ and a Dirichlet condition of zero is imposed on the rest of the boundary. Here $h$ is the typical mesh cell size for the mesh under consideration. Let $u$ be the Laplace solution for this boundary condition. The Poisson integral formula may be used to show that for any point $t$ inside the unit disk, $||\nabla u(t)|| \leq \text{const} \cdot h^2/\text{dist}(t, S)$. Therefore, if $t_1, t_2$ are two points in the unit disk satisfying $||t_1 - t_2|| \leq h$, then

\[|u(t_1) - u(t_2)| \leq \text{const} \cdot h^2/\min(\text{dist}(t_1, S), \text{dist}(t_2, S))^2. \]

If we hypothesize that a bound of the same form holds for the LWLS algorithm, we conclude that

\[||W_T r|| \leq \text{const} \cdot h^2/\text{dist}((x_I, y_I), (x_B, y_B))^2. \]

This takes care of the first factor in (24). Because the second factor is quadratic, it is bounded above by $\text{const} \cdot \text{dist}((x_I, y_I), (x_B, y_B))^2$. Combining shows that $|q| \leq \text{const} \cdot h^2$ since the $dist^2$ terms cancel each other.

Since there are $b = O(1/h)$ terms in the sum $q_1 + \cdots + q_k$, we conclude that the overall contribution from the quadratic term of (23) is $O(h)$ and that the constant factor in this $O(h)$ bound depends on the second derivative of $f$. On the other hand, the first term in (23) is also $O(h)$ (since $W_T$ takes a finite difference of locations of mesh neighbors), but here the constant depends on the first derivative. This concludes our justification of the conjecture.

We have less insight into how the mesh aspect ratio affects (23), but we can show that no weight is too close to zero as long as the mesh has bounded aspect ratio.

**Theorem 4** There is a positive-valued function $q(x)$ such that $w_i > q(x)$, where $w_i$ is any weight computed by the LWLS algorithm, and $a$ is the worst (largest) aspect ratio in the mesh.

**Proof.**

First, for this proof we need the following claim.

Claim. Let $p = (1, 0)$ be an interior node in a bounded-aspect ratio triangulation, and suppose that all of its neighboring nodes have nonnegative $x$-coordinates. Let $x_{max}$ be the maximum $x$-coordinate among neighbors of $p$. Then $x_{max} \leq q_0(a)$, where $q_0$ is a fixed function and $a$ is the mesh aspect ratio.

The proof of claim is omitted, but we give a sketch of its main idea here. For more details, see [24]. Consider the triangle $t_i$ containing $p$ and also the line segment starting at $p$ and proceeding in the negative $x$
direction. Since no neighbor of \( p \) has a negative \( x \)-coordinate, we conclude that this triangle \( t_1 \) has an altitude at most \( \lambda \). Therefore, its maximum side-length is at most \( \lambda \), using the following definition of the aspect ratio: \( AR = \frac{4\max}{\theta_{\text{min}}} \), where \( \max \) and \( \theta_{\text{min}} \) denote the maximum side-length and minimum altitude of the triangle. The triangle immediately adjacent to \( p \), possessing one side of length at most \( \lambda \), therefore has maximum side-length at most \( \lambda \), and so on. The number of triangles adjacent to \( p \) is at most \( \lambda \), using the following definition of the aspect ratio, \( AR = 1/\theta_{\text{min}} \), where \( \theta_{\text{min}} \) is the minimum angle in the triangle. (Note that the two definitions of aspect ratio are equivalent up to a constant.) Therefore, the longest side-length among neighbors of \( p \) is \( \lambda \), which is the function \( \lambda(p) \). A similar argument appears in [25].

Once the claim is established, we can return to the proof of the theorem, which is as follows. Fix an interior point \( p \) in the initial mesh with \( n \) neighbors, and let the optimal weights computed by the LWLS algorithm be \( w_1, \ldots, w_n \). (Omit the second subscript since \( p \) is fixed throughout the proof.) Let the neighbors of \( p \) be \( (x_1, y_1), \ldots, (x_n, y_n) \), and let \( p \) itself be located at \( (x^p, y^p) \). Since the LWLS problem is a strictly convex optimization problem on a bounded convex set, the Lagrange multiplier conditions are necessary and sufficient for optimality [26]. These conditions are as follows. There exist three multipliers \( \lambda, \mu, \nu \) such that:

\[
\begin{align*}
    w_{i1}x_1 + \cdots + w_{in}x_n &= x^p, \\
    w_{i1}y_1 + \cdots + w_{in}y_n &= y^p, \\
    w_1 + \cdots + w_n &= 1, \\
    \lambda x_1 + \mu y_1 + \nu &= 1/w_1, \\
    \vdots \\
    \lambda x_n + \mu y_n + \nu &= 1/w_n.
\end{align*}
\]

First, let us consider a very simple case that \( \lambda = \mu = 0 \). In this case, each \( w_i \) is equal to \( 1/\nu \). Since the sum of the \( w_i \)'s is 1, this means in fact that \( 1/\nu = 1/n \) and hence \( w_i \geq 1/n \) for all \( i \). Furthermore, \( 1/n > (1/(\alpha a)) \) using the argument about the number of neighbors from the proof of the claim. This shows that \( w_i > q(a) \) in this special case.

The generic case is that at least one of \( \lambda, \mu \) is nonzero. Without loss of generality, we can assume \( \lambda = 1, \mu = \nu = 0 \). The reason this assumption is WLOG is as follows. We can apply an arbitrary transformation of the form:

\[
(x, y) \mapsto (x, y) \left( \begin{array}{cc} u & v \\ -v & u \end{array} \right) + (k_1, k_2)
\]

(with at least one of \( u, v \) nonzero) uniformly to the original data \( (x_1, y_1), \ldots, (x_n, y_n), (x^p, y^p) \); applying such a transformation has the effect of transforming \( \lambda, \mu, \nu \) arbitrarily as long as not both \( \lambda, \mu \) are zero but does not change any of the aspect ratios in the original triangulation (since the above transformation is a combination of a rigid motion, a translation, and a uniform scaling).

After this transformation is applied, we can further apply a uniform translation to all the \( y \)-coordinates in the original data to force \( y^p = 0 \). Since \( \lambda = 1 \) and \( \mu = \nu = 0 \), we conclude from the KKT system that \( x_i = 1/w_i \) so that \( w_i x_i = 1 \). This means that \( w_1 x_1 + \cdots + w_n x_n = n \), hence \( x^p = n \) so \( p = (n, 0) \). Also, the relationship \( x_i = 1/w_i \) means that all the \( x_i \)'s are positive. Finally, the relationship \( x_i = 1/w_i \) means that \( \lambda x_i + \mu y_i + \nu = 1/w_i \). In particular, in order to have a very small \( w_i \) means that there must be a very large \( x_i \). It follows from the claim that \( \max x_i \leq n q_0(\alpha) \), hence \( \min x_i \geq 1/(n q_0(\alpha)) \). Therefore, there exists a positive-valued function \( q(x) \) such that all the weights are at least \( q(\alpha) \). ■

### 4. CONVERGENCE THEORY

Next, we study the convergence of LWLS and compare it with the convergence of the local weighted Laplacian smoothing and Gauss-Seidel algorithms.

**Theorem 5** The LWLS Algorithm converges to the same point as the local weighted Laplacian smoothing method and the Gauss-Seidel method taken one pass at a time.

**Proof.**

First, we note that as proved in Theorem 1, unique positions result when using LWLS to solve for positions of the interior nodes. Throughout this discussion we assume that the input transformation has already been applied to the boundary.

Now, we discuss the convergence of the Gauss-Seidel algorithm when applied to \( A_I^0[x_I, y_I] = -A_I^0[\bar{x}_B, \bar{y}_B] \). It is well-known in the numerical analysis community that this algorithm is guaranteed to converge for an \( M \)-matrix. (See [27] for the details.) Thus, according to Theorem 2, it will converge when applied to this system.

Next, we observe that it is easy to show the iterates produced by the local version of weighted Laplacian smoothing to the above linear system are the same. (The details are omitted due to space constraints.)

Thus, we can compute the iterates for the local version of weighted Laplacian smoothing as follows:

\[
[x_I, y_I]^{(k+1)} = C[x_I, y_I]^{(k)} + F[\bar{x}_B, \bar{y}_B],
\]

where \( C = (M_2^{-1} N_G), F = (-M_2^{-1} A_B), M_G = D + L, D = \text{diag}(A_I), L = \text{tril}(A_I), N_G = -\text{triu}(A_I), \) and \( [x_I, y_I]^{(0)} = [x_I, y_I] \).

Finally, we show that the local version of weighted Laplacian smoothing converges to the same point as LWLS. To this end, we consider the convergence of the local version of the weighted Laplacian smoothing method. In order to study its convergence, we need to concern ourselves with
Putting everything together, we see that (26) is equal to
\[
\lim_{k \to \infty} \left( \begin{array}{ccc} C & F \\ 0 & I \end{array} \right)^k.
\]  
(25)

Note that (25) is equal to
\[
\left( \lim_{k \to \infty} C^k \right) \left( \sum_{j=0}^{\infty} C^j \right) = (I - C)^{-1} F.
\]
(26)

Now, observe that \( (I - C)^{-1} F = (I - C)^{-1} F. \) In addition, observe that \( A_I \) and \(-A_B\) can be obtained from \( I - C \) and \( F \) via elementary row operations since \( I - C = I - M_B^{-1} N_B \) and \( F = -M_B^{-1} A_B \). Thus, the desired matrices are obtained by left multiplying by \( M_B \). This proves that \( (I - C)^{-1} F = -A_I^{-1} A_B \).

The final observation to make is that the eigenvalues of \( C \) satisfy \( \rho(\lambda) < 1 \), which can be seen as follows. First, we note that all of the entries of \( C \) are nonnegative by definition of \( C \). Second, we note that each row sum is less than or equal to 1 since each row is a convex combination of rows with nonnegative entries and row sums that are less than or equal to 1. In addition, there is at least one row with row sum strictly less than 1 since there is at least one interior node with a boundary neighbor. Thus, the eigenvalues of \( C \) are less than 1 in absolute value, i.e., \( \lim_{k \to \infty} C^k = 0 \).

Putting everything together, we see that (26) is equal to
\[
\begin{pmatrix} 0 & -A_I^{-1} A_B \\ 0 & I \end{pmatrix}.
\]
(27)

This proves the theorem, since the solution to the second linear system, \( A_I [\hat{x}_I, \hat{y}_I] = -A_B [\hat{x}_B, \hat{y}_B] \), can be written as follows:
\[
\begin{pmatrix} \hat{x}_I \\ \hat{y}_I \end{pmatrix} = \begin{pmatrix} 0 & -A_I^{-1} A_B \\ 0 & I \end{pmatrix} \begin{pmatrix} x_I \\ y_I \end{pmatrix}.
\]

\[\Box\]

5. NUMERICAL TESTS

In order to test the robustness of LWLS, we designed three numerical tests each based upon a series of mesh deformations. These tests have been designed to mimic the basic elements of a beating heart’s motion, namely translation and rotation. We study the heart’s motion in Section 6. All of the tests involve the deformation of an annulus on each timestep. The annulus is composed of four, equally-spaced concentric rings of triangles. Its inner radius is 1, and its outer radius is 10. The initial annulus mesh is shown in the upper, left-hand corner of Figure 3.

In the first test, the inner circle was deformed via an outward translation on each iteration. In particular, its radius was increased by 0.5 each timestep. The test showed that when the inner radius of the annulus became 6.5, too much deformation had occurred, since the mesh had inverted, and the minimum mean ratio of a triangle in the mesh was less than 1. (See [3] for the mean ratio definition.) However, as this test demonstrates, LWLS tolerates a large amount of deformation is tolerated before inversion occurs.

Rotation of the inner circle occurs in the second deformation series. On each iteration, the inner circle was rotated counterclockwise 10 degrees. While performing this test, we observed that as the inner circle rotates counterclockwise, the quality of the triangles in the inner ring decreases. The mesh remained untangled until it had been rotated 70 degrees.

The goal of the third test was to combine translation and rotation of the inner circle. At each timestep, the motions of the first two tests were combined, i.e., the radius of the inner circle was increased by 0.5, and it was rotated by 10 degrees. Figure 3 shows the results. The new radius for the inner circle, its amount of rotation, and the minimum mean ratio of the triangles are given. Observe that the mesh remains untangled until the radius of the inner circle is 3.5 and the circles has been rotated by 50 degrees.

From these three tests, we conclude that LWLS can withstand relatively large amounts of deformation of various kinds while resisting inversion. However, whenever the deformations either tangle the boundary or are too large, LWLS fails because inversion of elements occurs.
6. APPLICATION TO CARDIOLOGY

We now use LWLS in order to study the movement of the beating canine heart under normal conditions. To do this, we obtained data from the Laboratory of Cardiac Energetics at the National Institutes of Health (NIH) [28]. We were given \((x, y, z, t)\) data for 192 points on the inner surface of the left and right ventricles of the beating canine heart from a physiological pacing experiment. The data frames are 14.6 milliseconds apart with the first frame occurring 12 milliseconds before the pacing spike.

The first step in the simulation of the ventricular movement was to generate a mesh for the initial position of the ventricles. In order to do this, we first noted that the 192 points we were given were arranged in 8 slices with 24 points each. Thus, in order to generate the initial mesh, we decided to create a mesh for the top slice and then use LWLS to do the mesh warping necessary to create meshes for the remaining slices. Note that this uses LWLS in one and two dimensions as we describe in detail below. Once we have the meshes for all of the levels, we connect the triangular meshes into a tetrahedral mesh for the ventricles.

The procedure to do this is also described below.

We now give a more detailed description of the method we used to create the initial mesh of the canine ventricles. We first used the two-dimensional quality mesh generation package called Triangle [29] to generate an initial mesh of the top slice (after projecting it into the \(x, y\) plane). Note that this yielded a good-quality mesh in the \(x, y\) plane with several additional nodes. Second, we computed the \(z\)-coordinates for the new points on the boundary of the top slice using 1D LWLS. Third, we used the weight-finding portion of our LWLS algorithm to compute the weights for the appropriate 2D linear system obtained from the \(x\) - and \(y\)-coordinates. Fourth, we determined the \(z\)-coordinates for the mesh of the top slice by forcing the \(z\)-coordinates to satisfy the appropriate 3D linear system using the 2D weights. At this point, we had the mesh for the top slice.

Our second task was to generate meshes for each of the remaining 7 slices. This was done using our LWLS algorithm to warp the mesh for the top slice into meshes for each of the remaining slices. In order to accomplish this, the first step was to determine the coordinates of the additional boundary nodes for the mesh of the appropriate slice. This was done using 1D LWLS. Then, the \((x, y)\) coordinates of the interior nodes of that slice were determined using 2D LWLS. The \(z\)-coordinates for the interior nodes were found by forcing them to satisfy the appropriate 3D linear system using these weights.

The third step was to connect the triangular meshes for each of the 8 slices into one tetrahedral mesh for the canine ventricles. To do this, the corresponding triangles between two slices were connected to form a triangular prism. After a temporary mesh of triangular prisms was created, the triangular prisms were subdivided into tetrahedra using the method outlined in [30].

After the initial tetrahedral mesh was created, we checked its quality using the mean ratio mesh quality metric. Using this test, it was determined that the initial mesh was of poor quality. This was because the 8 slices were equally-spaced even though the curvature of the ventricles changes much more rapidly near the bottom of the heart. Thus, we decided to use linear interpolation in conjunction with LWLS in the obvious way in order to add two additional slices of nodes near the bottom of the heart. Because the curvature of the ventricles changes more rapidly near the bottom of the heart, the first additional slice was placed halfway between levels 7 and 8, and the second additional slice was placed halfway between the first new level and level 8. The resulting initial mesh is shown in Figure 4.

![Figure 4: Initial canine ventricular mesh.](image-url)
each timestep than the circular rings in the test cases. However, the motion of the heart is anisotropic which makes it difficult to predict in advance how it will tolerate deformations. Interestingly enough, the values of the minimum and average mean ratios were relatively constant across all timesteps. Only the value of the maximum mean ratio changed a significant amount. The mean ratio computations are also a good indication that the heart meshes are of sufficiently good quality for use with a numerical PDE solver that requires moving meshes.

In order to further test our mesh warping algorithm, the motion of the ventricles was exaggerated by a factor of 3. In this case, the motion was large enough to detect in separate figures and is shown in Figure 5. We note that the LWLS algorithm also performed successfully in this case, which is encouraging given the much larger deformations.

7. CONCLUSIONS

In summary, we developed a new mesh warping algorithm for tetrahedral meshes based upon weighted laplacian smoothing.

Our method determines a set of local weights for each interior node which describe the relative distances of the node to its neighbors. A deformation is then applied to the boundary, and the method solves a system of linear equations based upon the weights to determine the final positions of the interior nodes.

Furthermore, LWLS is simple, preserves the mesh’s combinatorial structure, is computationally inexpensive, and provides theoretical guarantees for use on the mesh warping problem.

We proved a theorem regarding mesh invertibility and a lemma showing that LWLS yields exact results for affine boundary transformations. In addition, we proved a convergence result comparing LWLS to the local weighted laplacian smoothing and Gauss-Seidel algorithms.

We also used LWLS to study the motion of the canine ventricles under normal conditions.

We now describe some possibilities for future research. First, we would like to extend the above theory to prove Conjecture 1. Second, we would like to use our canine ventricle meshes to study the bioelectricity of the heart. To do this, we will couple the heart’s electrical activity with its mechanical motion (from the LWLS meshes). We will then use the finite element method to simulate the electricity on the beating heart. This will yield new knowledge of the heart’s activity since most of the current models have not coupled the heart’s electricity with its mechanical motion.

8. ACKNOWLEDGEMENTS

The authors wish to thank O. Faris and E. McVeigh from the Laboratory of Cardiac Energetics at the
References


A NEW ALGORITHM FOR GENERATING QUADRILATERAL MESHES AND ITS APPLICATION TO FE-BASED IMAGE REGISTRATION

S. Ramaswami\textsuperscript{1} M. Siqueira\textsuperscript{2,3} T. Sundaram\textsuperscript{2} J. Gallier\textsuperscript{2} J. Gee\textsuperscript{2}

\textsuperscript{1}Rutgers University, Camden, NJ 08102, USA, rsuneeta@camden.rutgers.edu
\textsuperscript{2}University of Pennsylvania, Philadelphia, PA 19104, USA, \{marcelos.tessa\} @seas.upenn.edu
\textsuperscript{3}Universidade Federal de Mato Grosso do Sul, Campo Grande, MS 79070-900, Brazil.

ABSTRACT

The use of finite element (FE) analysis in the simulation of physical phenomena over the human body has necessitated the construction of meshes from images. Despite the availability of several tools for generating meshes for FE-based applications, most cannot deal directly with the raw pixel-wise representation of image data. Additionally, some are optimized for the construction of much simpler shapes than those encountered within the human body. In this work, we introduce a new algorithm to obtain strictly convex quadrilateral meshes of bounded size from triangulations of polygonal regions with or without polygonal holes. We present an approach to construct quadrilateral meshes from segmented images using the aforementioned algorithm, and a quantitative analysis of the quality of the meshes generated by our algorithm with respect to the performance of a FE-based image registration method that takes image meshes as input.

Keywords: Quadrilateral mesh, mesh generation, finite element method, image registration

1. INTRODUCTION

Finite element analysis (FEA) is a powerful tool for numerically solving differential equations of variational problems that arise during structural modeling in engineering and the applied sciences. An essential prerequisite for the use of FEA is the availability of a mesh over the problem domain. If the problem domain is a subset of the Euclidean plane, triangular or quadrilateral meshes are typically employed. The accuracy of a problem’s solution and the efficiency with which it is obtained using a particular FE implementation are highly dependent on a variety of mesh parameters, including element number and shape, and mesh regularity, directionality and grading, [1].

Triangular meshes have been extensively investigated by the meshing community, and their theoretical properties are now well understood. Algorithms for generating provably good triangular meshes of polygonal domains have been proposed, [2]. On the other hand, the generation of good quadrilateral meshes is not as well understood. A few algorithms exist to generate quadrilateral meshes of bounded size, [3, 4, 5, 6], bounded maximum angle, [4], and controlled density and directionality, [7, 8], for polygonal domains. However, there are no known algorithms to generate quadrilateral meshes of arbitrarily complex polygonal domains that are provably guaranteed to simultaneously meet several quality criteria. Yet good quality quadrilateral meshes may be more desirable for certain FE-based applications, such as planar stress-strain analysis, [9].

The use of FE analysis in biomedical research has led researchers to explore the construction of meshes from images, [10]. In two dimensions, image data is represented as a discrete collection of pixels. Although several algorithms have been proposed for generating meshes of arbitrary geometric domains, most of them have been developed and optimized with structural engineering applications in mind. As a result, they are
not directly applicable to discrete data, and they may not perform very well when presented with geometrically complex shapes such as the ones encountered in biology.

In this paper we are primarily concerned with the generation of strictly convex quadrilateral meshes from images, and the influence of their quality on the accuracy and performance of a particular implementation of a FE-based image registration method. Strictly convex quadrilateral meshes are meshes in which each of the four angles of every quadrilateral is strictly less than 180°, and they are the only desirable quadrilateral meshes for FE-based applications.

The main contributions of our work are two-fold. First, we describe a new algorithm for generating strictly convex quadrilateral meshes of provably small size for polygonal domains with or without polygonal holes. In particular, we show that the interior of a polygonal region with holes triangulated with \( t \) triangles can be converted into a quadrilateral mesh with at most \( \frac{2t}{3} + 2 \) strictly convex quadrilaterals. We also use this algorithm to show that an arbitrary constrained triangulation (i.e., a triangulation in which some edges, called constraint edges, must be included) of \( t \) triangles can be converted into a constrained quadrilateral mesh with at most \( \frac{2t}{3} + 3h \) quadrilaterals, where \( h \) is the number of connected components in the dual graph of the triangulation. These results improve on previously known bounds on mesh size. Our algorithm runs in \( O(t) \) time and space. Second, we provide a quantitative comparison of the accuracy and performance of FE-based image registration when presented with distinct types of meshes from magnetic resonance (MR) images of the human brain (including the ones generated by our aforementioned algorithm).

The remainder of this paper is organized as follows. In Section 2 we introduce some basic concepts related to our work and review related work. In Section 3 we describe the details of our new algorithm for generating quadrilateral meshes of polygonal domains. In Section 4 we discuss an approach to generate meshes from images using algorithms for meshing polygonal domains. In Section 5 we define image registration, briefly describe the FE-based image registration method used here, and present the aforementioned comparative and quantitative analysis. In Section 6 we summarize our results and discuss future work.

2. BACKGROUND AND RELATED WORK

The problem of generating a quadrilateral mesh of a polygonal region \( \mathcal{R} \) is more complex than that of producing a triangular mesh. For one thing, if we require the set of vertices of the mesh to be the set of vertices of \( \mathcal{R} \), a quadrilateral mesh may not even exist, and the problem of deciding whether or not it exists has been shown to be NP-complete for \( \mathcal{R} \) with one or more holes, [11]. In addition, the theoretical properties to generate good quality quadrilateral meshes are not as well understood as the ones for producing good quality triangular meshes. These facts have led several researchers to adopt an indirect approach to obtaining quadrilateral meshes: the polygonal domain is first triangulated and then the triangulation is converted into a quadrilateral mesh, [3, 12, 5, 7, 13, 8]. This approach relies on the premise that a good quality quadrilateral mesh could be more easily generated from an existing triangulation of the problem domain.

Let \( \mathcal{R} \) be a polygonal region with \( n \) vertices and \( k \) polygonal holes, and let \( \mathcal{T} \) be any triangular mesh of \( \mathcal{R} \) such that the set of vertices \( \mathcal{V}_\mathcal{R} \) of \( \mathcal{R} \) is contained in the set of vertices \( \mathcal{V}_\mathcal{T} \) of \( \mathcal{T} \), \( \mathcal{V}_\mathcal{R} \subseteq \mathcal{V}_\mathcal{T} \). From Euler’s relation, we know that \( \mathcal{T} \) has \( t = 2m + 2k - 2 - m_0 \) triangles, where \( m \) is the number of vertices of \( \mathcal{T} \) and \( m_0 \) is the number of vertices of \( \mathcal{T} \) on its boundary, with \( n \leq m_0 \leq m \). A very simple algorithm for converting such a triangulation into a strictly convex quadrilateral mesh was proposed by de Berg, [3]. His algorithm runs in \( O(t) \) time, produces \( 3t \) quadrilaterals, and inserts exactly \( 3m + 5k - 5 - 2m_0 \) extra vertices inside \( \mathcal{R} \). It is simple and fast, but the size of the output quadrilateral mesh may prevent its practical use in the presence of large input triangular meshes. Everett et al., [3], introduced another linear time algorithm to convert triangular meshes into strictly convex quadrilateral ones that generates at most \( \frac{2n}{3} \) quadrilaterals. However, the size of the output quadrilateral mesh may still be prohibitive in practice. An interesting feature of this algorithm, which is also present in de Berg’s algorithm, is the preservation of the input mesh grading. Johnston et al. proposed another indirect approach-based algorithm that uses several heuristics to obtain a strictly convex quadrilateral mesh from a triangulation, [12]. Their algorithm runs in \( O(t^2) \) time, and selectively combines adjacent triangles to obtain quadrilaterals. However, it is not clear from the description in [12] that the heuristic procedures are always successful in producing an all-quadrilateral mesh.

Shimada et al., [7], proposed an algorithm for generating quadrilateral meshes that takes into account mesh regularity, directionality and grading as well as element shape. Their algorithm employs a physically-based relaxation process, called cell packing, that fills in the problem domain with squares, whose size and direction are controlled by user-defined, scalar density and vector functions. Mesh vertices are placed at the center of every square and then connected to generate a triangulation of the entire domain. Finally, the triangulation is converted into a strictly convex quadrilateral mesh. Later, Viswanath et al., [8],
modified this algorithm by using rectangular cells instead of square cells, which enabled them to generate anisotropic quadrilateral meshes. Both algorithms produce nearly regular quadrilateral meshes with well-shaped elements and precise control over their direction and size distribution. However, if precise control of directionality is not critical and the problem domain has complex geometry, neither algorithm may be very attractive due to the expense of the cell packing approach. Furthermore, conversion of a triangular mesh into a quadrilateral one may leave isolated triangles, which may cause the algorithm to undergo an extra subdivision step to obtain an all-quadrilateral mesh.

Owen et al., [13], presented another quadrilateral meshing algorithm that takes into account directionality and element shape. It converts a triangular mesh into a strictly convex quadrilateral one using advancing fronts initially defined by the boundary edges of the input mesh. Quadrilaterals are generated by combining and transforming triangles as the fronts move from the boundary to the interior of the input mesh. Local smoothing and topological clean-up, commonly performed as post-processing steps, are part of the conversion process. One limitation of this algorithm is that directionality cannot be arbitrarily specified as in [7, 8]. Although the algorithm in [13] and the ones in [7, 8] do not provide any theoretical bounds on mesh size nor element shape, they can generate very good quality quadrilateral meshes in practical applications.

The algorithms proposed in [3, 12, 13] take a triangular mesh as input, while the ones in [7, 8] take a polygonal region as input and build a triangulation for it. Most of the algorithms for generating triangular meshes, as well as the algorithms in [7, 8], cannot deal directly with image data represented by a discrete collection of pixels. However, using image processing techniques, it is possible to identify several distinct structures within an image and then build polygonal approximations for their boundaries, [14]. By using polygonal approximations rather than pixel-wise representations, we are able to employ any algorithm for generating triangular and quadrilateral meshes of polygonal domains to separately mesh the polygonal approximation of each individual structure on the image.

3. THE ALGORITHM

In this section, we present a new algorithm to convert an arbitrary triangulation, possibly with constrained edges, into a quadrilateral mesh of bounded size. The conversion allows unconstrained edges to be deleted, but does not allow deletion of input points (vertices). New points, called Steiner points, may be inserted along with new edges between Steiner points and/or input points, in order to construct the quadrangulation. The mesh produced by our algorithm consists of strictly convex quadrilaterals. First, we show that the interior of a triangulated polygonal region with holes can be quadrangulated with at most \( \frac{2t}{3} + 2 \) quadrilaterals by inserting at most \( t + 2 \) Steiner points, where \( t \) is the number of triangles. All Steiner points, except possibly one, lie in the interior of the polygonal region. Next, we show how to extend this algorithm to convert a constrained triangulation into a strictly convex quadrangulation satisfying the given constraints. The resulting number of quadrilaterals is at most \( \frac{2t}{3} + 3h \), obtained by using at most \( t + 2h \) Steiner points, where \( h \) is the number of connected components in the dual graph of the triangulation.

3.1 Polygonal Regions with Holes

The idea behind our algorithm is to quadrangulate a small group of triangles at a time until the input triangulation is converted into a quadrilateral mesh. The group of triangles represents the triangulation of small, simple polygonal regions (the ones with a small constant, \( \leq 7 \), number of vertices). This triangulation is converted into a partial or complete quadrangulation of the polygonal region. By using a spanning tree of the dual graph of the triangulation and processing it in a bottom-up fashion, our algorithm systematically groups triangles together so that no isolated triangles remain in the resulting decomposition.

Let \( \mathcal{R} \) and \( \mathcal{T} \) be as defined in Section 2. Let \( m \geq n \) be the number of vertices of \( \mathcal{T} \) and \( t \) the number of triangles of \( \mathcal{T} \). The first step of our algorithm is to build a rooted spanning tree \( T \) of the dual graph \( G \) of \( \mathcal{T} \). The dual graph of \( \mathcal{T} \) is the graph that contains a node for every triangle of \( \mathcal{T} \) and an edge between two nodes if the corresponding triangles share an (unconstrained) edge. \( T \) is built as a breadth-first search (BFS) tree. The root of \( T \) is any node corresponding to a triangle containing a boundary edge of \( \mathcal{T} \). Note that \( T \) is a binary tree. After constructing \( T \), the algorithm computes the set \( V_i \) of all nodes of \( \mathcal{T} \) at level \( i \), for every \( i \in \{0, 1, \ldots, d\} \), where \( d \) is the depth of \( \mathcal{T} \) and the root node is the singleton node at level 0. Next, the algorithm processes the nodes of \( T \) one level at a time in decreasing order of depth, i.e., \( V_0, V_{d-1}, \ldots, V_0 \). Let \( \text{par}(v) \) denote the parent of \( v \in \mathcal{V} \), \( \text{sib}(v) \) the sibling of \( v \), and \( \text{ele}(v) \) the triangle of \( \mathcal{T} \) corresponding to \( v \). Note that \( \text{ele}(v) \) and \( \text{ele}(\text{par}(v)) \) necessarily share an edge of \( \mathcal{T} \). For each vertex \( v \in V_k \) (1 \( \leq k \leq d \)) we consider the subtree rooted at \( \text{par}(v) \) or at \( \text{par}(\text{par}(v)) \). We denote this subtree by \( T_v \) and its root by \( v_0 \). Let \( G_v \) denote the subgraph of \( G \) induced by \( T_v \). We show that in the original triangulation \( \mathcal{T} \), the subgraph \( G_v \)
corresponds to a triangulated polygonal region $T_v$ consisting of 4, 5, 6, or 7 vertices. This triangulation is then converted into a partial or complete quadrangulation by adding Steiner points within the boundary of $T_v$. If the result is a complete quadrangulation of the domain of $T_v$, the subtree $T_v$ is eliminated from $T$. If the quadrangulation is not complete, there will be only one leftover triangle within the boundary of $T_v$. The root node $r_v$ now represents this triangle and the remaining nodes of $T_v$ are eliminated from $T$. The sets $V_k, V_{k-1}$, and $V_{k-2}$ are updated accordingly.\(^2\)

The algorithm runs in phases. Each phase of the algorithm examines nodes from the three deepest levels and eliminates some of them from $T$ and from the appropriate $V_i$. We show that for every two nodes eliminated from $T$ during a given phase, at most three quadrilaterals are created by using at most two Steiner points. In other words, we allow the creation of “one and a half” quadrilaterals per triangle in the pruning process. Furthermore, at the end of each phase, the depth of $T$ decreases by at least one. The tree $T$ is thus pruned until all nodes are eliminated and the underlying triangulation $T$ is converted into a strictly convex quadrangulation.

Before describing the details of our algorithm, however, we discuss two special situations. First, when $T_v$ is a subtree of three nodes containing $v, r_v = \text{par}(v)$ and $\text{sib}(v)$, and there is a cross-edge between $v$ and $\text{sib}(v)$ in $G_v$, the triangulated polygonal region $T_v$ has a point in its interior, as shown in Figure 1. Second, when processing $T_v$, our algorithm may place a Steiner point $s$ on the edge $e$ between $\text{ele}(r_v)$ and $\text{ele}(\text{par}(r_v))$, as shown in Figure 2.

Observation 3.1.1. Since $T_v$ gets eliminated from $T$ when the Steiner point is placed on edge $e$, degenerate pentagons are leaves of $T$, and degenerate quadrilaterals are either leaves or internal nodes of degree 2. Furthermore, if $v$ is at level $k$, the nodes corresponding to these degenerate elements are at level $k - 2$ or level $k - 3$ (refer to cases 3d, and 4c(ii) in the algorithm description).

Observation 3.1.2. In all cases when such degenerate elements are created, the number of nodes in $T_v$ is odd. Since we are allowed “one and a half” quadrilaterals per node but only create whole numbers of quadrilaterals, this implies that we have a “credit” of at least “half” a quadrilateral for every Steiner point $s$ on the boundary of a degenerate element.

Observation 3.1.3. Since all quadrilaterals in the quadrangulation of the domain of $T_v$ are strictly convex, there must be an edge of the quadrangulation incident on $s$ and lying outside $\text{ele}(\text{par}(r_v))$, as shown in Figure 2. We can slightly perturb $s$ along this edge without destroying the strict convexity of the quadrilaterals.

We now describe the steps involved in each phase of the algorithm. During the course of the description, we refer to various lemmas pertaining to quadrangulations of small polygonal regions, which are stated formally later in Section 3.3. Let $1 \leq l \leq d$ be the current deepest level of $T$. We first eliminate all leaves $v$ of $T$ such that $\text{ele}(v)$ is a degenerate quadrilateral, degenerate pentagon, or a non-empty triangle. Note that the first two types of leaves will be at levels $l, l - 1$, or $l - 2$, and the third type will all be at level $l$.

Step 1. Eliminate all $v \in V_l \cup V_{l-1} \cup V_{l-2}$ such that $v$ is a leaf and $\text{ele}(v)$ is a degenerate quadrilateral. Let $s$ be the Steiner point of $\text{ele}(v)$, and let $e_s$ be the edge of the quadrangulation (constructed thus far) incident on $s$. We convert $\text{ele}(v)$ into a strictly convex quadrilateral by perturbing $s$ along the edge $e_s$, as shown in Figure 2a, and then we remove $v$ from $T$.

Step 2. Eliminate all $v \in V_l \cup V_{l-1} \cup V_{l-2}$ such that $\text{ele}(v)$ is a degenerate pentagon. Let $s_1$ and $s_2$ be the

---

\(^2\)This general idea of pruning the dual tree was also used in [5] to convert triangulations into quadrangulations consisting of quads that are not necessarily convex.
two Steiner points of \( e(v) \) and let \( e \) be the shared edge of \( e(v) \) and \( e(\text{par}(v)) \). It is straightforward to convert \( e(v) \) into a strictly convex quadrilateral and a leftover triangle \( \Delta \) as shown in Figure 2b. Now \( v \) represents the leftover triangle, i.e., \( e(v) = \Delta \). Note that we have created one convex quadrilateral, but have not eliminated any nodes from \( T \). However, from Observation 3.1.2, we know that each of \( s_1 \) and \( s_2 \) has a credit of half a quadrilateral. Thus, the number of quads produced in this case remains within the stated bounds.

**Step 3.** Eliminate all \( v \in V_i \) such that \( e(v) \) is a non-empty triangle. Let \( T_v \) be the subtree of \( T \) rooted at \( \text{par}(v) \). We consider the following cases (refer to the illustrations in Figure 3 and Figure 4):

**3a.** \( \text{par}(v) \) is a node of degree 2, and \( e(\text{par}(v)) \) is a degenerate quadrilateral. Let \( s \) be the Steiner point of \( e(\text{par}(v)) \). By connecting \( s \) to the vertex of \( e(\text{par}(v)) \) that is not adjacent to \( s \), we decompose \( T_v \) into a triangle \( \Delta \) and a quadrilateral with a point in its interior (see Figure 3a). The latter can be quadrangulated into five convex quads with three Steiner points in its interior by Lemma 3.2.2. We then remove \( v \) from \( T \). A total of three nodes (the nodes that gave rise to \( v \) have been eliminated from \( T \). Once again, since \( s \) has a credit of half a quadrilateral, the five convex quads created in this step keep us within the stated bounds. The node \( \text{par}(v) \) now corresponds to the triangle \( \Delta \).

**3b.** \( \text{par}(v) \) is a node of degree 2, and \( e(\text{par}(v)) \) is a triangle. In this case, the domain of \( T_v \) is a quadrilateral and this quadrilateral contains a vertex of \( T_v \) in its interior (see Figure 3b). Again, by Lemma 3.2.2, the region can be quadrangulated into five convex quads with three Steiner points in its interior. We then remove \( v \) and \( \text{par}(v) \) from \( T \). Four nodes have been eliminated from \( T \).

**Figure 3: Cases 3a and 3b of Step 3.**

**3c.** \( \text{par}(v) \) is a node of degree 3, and \( e(\text{par}(v)) \) is a triangle. If \( G_v = T_v \), then the domain of \( T_v \) is a pentagon and this pentagon contains a vertex of \( T_v \) in its interior (see Figure 4a). Then by Lemma 3.2.4, this region can be decomposed into at most six convex quads and one triangle \( \Delta \) by using at most four Steiner points in the interior. Remove \( v \) and \( \text{par}(v) \) from \( T \), and let \( r_v \) now represent \( \Delta \). If \( G_v \) contains a cross-edge between \( v \) and \( \text{par}(v) \), then \( e(v) \) and \( e(\text{par}(v)) \) form a quadrilateral with a point inside (see Figure 4a), which can be decomposed into five convex quadrilaterals by using three Steiner points. Eliminate \( v \) and \( \text{par}(v) \) from \( T \). Four nodes have been eliminated from \( T \) in either case.

**Figure 4: Cases 3c and 3d of Step 3.**

**3d.** \( \text{par}(v) \) is a node of degree 3, and \( e(\text{par}(v)) \) is a non-empty triangle. If \( G_v = T_v \), then the domain of \( T_v \) is a pentagon and this pentagon contains two vertices of \( T_v \) in its interior. If \( G_v \) contains a cross-edge between \( v \) and \( \text{par}(v) \), then the domain of \( T_v \) is a triangle and this triangle contains three vertices of \( T_v \) in its interior. In either case, \( T_v \) can be decomposed into two quadrilaterals, each with a point in its interior as follows: add a Steiner point on the edge shared by \( e(\text{par}(v)) \) and \( e(\text{par}(v)) \) and connect it to the vertex of \( e(\text{par}(v)) \) that is not adjacent to it (see Figure 4b). By Lemma 3.2.2, each quadrilateral can be decomposed into five convex quads using three Steiner points. Remove all nodes of \( T_v \) from \( T \). Hence a total of seven nodes were eliminated and ten convex quadrilaterals were created using seven Steiner points. In the next phase of the algorithm, \( e(\text{par}(v)) \) will be a degenerate quadrilateral or pentagon.

**Observation 3.1.4.** After steps 1-3 are carried out, for every node \( v \in V_i \cup V_{i-1} \), \( e(v) \) is either a triangle or a degenerate quadrilateral. In the latter case, \( v \) is a node of degree 2.

**Step 4.** The last step eliminates all remaining \( v \in V_i \). From Observation 3.1.4, and the fact that \( T \) is a BFS tree, it follows that the only possible configurations for \( T_v \) are those described in the sub-steps below.
4a. Eliminate all \( v \in V_i \) such that \( \text{ele}(\text{par}(v)) \) is a degenerate quadrilateral. Let \( T_v \) be the subtree of \( T \) rooted at \( r_v = \text{par}(v) \). Perturb the Steiner point \( s \) of \( \text{ele}(r_v) \) along the quadrangulation edge incident to it. A Steiner point \( s' \) placed in \( \text{ele}(r_v) \) decomposes the region \( \mathcal{T}_v \) into two convex quads and a triangle \( \Delta \) adjacent to the shared edge of \( \text{ele}(r_v) \) and \( \text{ele}(\text{par}(r_v)) \) (see Figure 5a). Eliminate \( v \) from \( T \) and let \( r_v \) now represent the triangle \( \Delta \). The credit of half a quadrilateral on \( s \) keeps the number of convex quads within the stated bounds.

```
(a) \( \text{ele}(\text{par}(v)) \)
(b) \( \text{ele}(v) \)
```

**Figure 5: Cases 4a and 4b of Step 4.**

4b. Eliminate all \( v \in V_i \) such that \( \text{par}(v) \) is a node of degree 3. Again, let \( T_v \) be the subtree of \( T \) rooted at \( r_v = \text{par}(v) \) and refer to Figure 5b. If \( G_v \) contains an edge between the nodes \( v \) and \( \text{sib}(v) \), then we remove \( v \) and \( \text{sib}(v) \) from \( T_v \). The node \( \text{par}(v) \) now is the non-empty triangle corresponding to the boundary of \( \mathcal{T}_v \), with the fourth vertex of \( \mathcal{T}_v \) in the interior of its domain (see Figure 1). If there is no cross-edge between \( v \) and \( \text{sib}(v) \), then by Lemma 3.2.3, the domain of \( \mathcal{T}_v \) can be subdivided into two convex quadrilaterals and one triangle \( \Delta \) (adjacent to the edge shared by \( \text{ele}(r_v) \) and \( \text{ele}(\text{par}(r_v)) \)) by adding one Steiner point. Remove \( v \) and \( \text{sib}(v) \) from \( T \). The node \( r_v \) now represents \( \Delta \).

4c. Finally, eliminate all \( v \in V_i \) such that \( \text{par}(v) \) is a node of degree 2. Let \( T_v \) be the subtree of \( T \) rooted at \( r_v = \text{par}(v) \). If the domain of the triangulated quadrilateral \( \mathcal{T}_v \) formed by \( \text{ele}(v) \) and \( \text{ele}(\text{par}(v)) \) is strictly convex, then we remove \( v \) and \( \text{par}(v) \) from \( T \). Otherwise, let \( T_v \) be the subtree of \( T \) rooted at \( r_v = \text{par}(\text{par}(v)) \) and consider the following sub-cases (refer to Figure 6, Figure 7, and Figure 8):

i. \( \text{ele}(r_v) \) is a degenerate quadrilateral. Note that \( G_v = T_v \) for this case because \( T \) is a BFS tree. Perturb the Steiner point of \( \text{ele}(r_v) \) along the quadrangulation edge incident to it so that the domain of \( \mathcal{T}_v \) is now a hexagon. By Lemma 3.2.1, this region can be subdivided into at most four convex quadrilaterals by using at most three Steiner points in its interior. Eliminate all the nodes of \( T_v \) from \( T \).

```
  \text{ele}(\text{par}(v))
  \text{par}(v)
  \text{ele}(v)
  v
  \text{ele}(\text{par}(v))
```

**Figure 6: Cases i and ii of Step 4c.**

ii. \( r_v \) is a node of degree 2 and \( \text{ele}(r_v) \) is a triangle. Once again, \( G_v = T_v \) for this case. Therefore, the domain of \( \mathcal{T}_v \) is a pentagon. We apply Lemma 3.2.3, where the shared edge of \( \text{ele}(r_v) \) and \( \text{ele}(\text{par}(r_v)) \) is designated as the “outgoing” edge. Then, we have two situations. First, the domain of \( \mathcal{T}_v \) is subdivided into three convex quads and one triangle \( \Delta \) adjacent to the outgoing edge by using two Steiner points. Second, the domain of \( \mathcal{T}_v \) is subdivided into four convex quads by using three Steiner points, one of which lies on the outgoing edge. In the first situation, we remove \( v \) and \( \text{par}(v) \) from \( T \), and let \( r_v \) now represent \( \Delta \). In the second situation, we remove all nodes of \( T_v \) from \( T \), and \( \text{ele}(\text{par}(r_v)) \) becomes a degenerate quad or a pentagon in the next phase of the algorithm.

```
  \text{ele}(\text{par}(r_v))
  \text{ele}(r_v)
  \text{par}(v)
  \text{ele}(v)
  v
  \text{ele}(\text{par}(r_v))
```

**Figure 7: Case iii of Step 4c (dashed edges are cross-edges).**

iii. \( r_v \) is a node of degree 3, and \( \text{sib}(\text{par}(v)) \) is a leaf. If \( G_v = T_v \), the domain of \( \mathcal{T}_v \) is a hexagon, to which we apply Lemma 3.2.1. If \( G_v \neq T_v \), then the domain of \( \mathcal{T}_v \) is a quadrilateral that contains a vertex of \( \mathcal{T}_v \) in its interior, to which we apply Lemma 3.2.2. In either case, at most five convex quadrilaterals are created by using at most three
Steiner points in the interior of $T_c$. Remove all nodes of $T_v$ from $T$.

iv. $v_i$ is a node of degree 3, and sib(par($v_i$)) is a node of degree 2. The different possibilities for the graph $G_v$ are derived from the fact that $T$ is a BFS tree. All cases are illustrated in Figure 8. Cases (a)–(e) correspond to a pentagon with a point in its interior, to which we apply Lemma 3.2.4. In cases (d)–(e), the non-root nodes of $T_v$ ($v, v_1, v_2$ and $v_3$ in the figure) correspond to a quadrilateral with a point inside, to which we apply Lemma 3.2.2. Finally, case (f) corresponds to a septagon, to which we apply Lemma 3.2.5. In all cases, we remove all four non-root nodes of $T_v$ and at most six convex quadrilaterals are created by adding at most four Steiner points.

After all $d - 2$ phases have been carried out, the sets $V_d, V_{d-1}, \ldots, V_2$ are all empty, and the sets $V_0$ and $V_1$ are possibly non-empty. If $V_0$ and $V_1$ are non-empty, apply Steps 1-3 and Step 4a to $V_1$ and then $V_0$. It is easy to show that now the only remaining nodes in $T$ are either the singleton root node, or the root node and one child, or the root node and two children. The domain of $T$ is thus either a triangle, a quadrilateral, a pentagon, or a triangle with an interior point. In the case of the quadrilateral, four internal Steiner points decompose it into five convex quads. In the other cases, we add one Steiner point outside the boundary of $R$ (this is unavoidable because the boundary of $R$ has odd parity) to obtain one, three, or five convex quads using zero, one, or three internal Steiner points, respectively.

![Figure 8: $G_v$ for case iv of Step 4c (dashed edges are cross-edges).](image)

**Theorem 3.1.1.** Given a polygonal region $R$, possibly with one or more polygonal holes, and a triangulation $T$ of $R$ with $t$ triangles, the algorithm described above converts $T$ into a strictly convex quadrangulation of $R$ with at most $\left\lfloor \frac{t^2}{4} \right\rfloor + 2$ quadrilaterals by using at most $t + 2$ Steiner points, all except one of which lie within the boundary of $R$. The algorithm runs in $O(t)$ time and space.

**Proof.** In each step of phase $l$ of the algorithm described above, at most $k$ Steiner points are added for every $k$ nodes eliminated from $T$. At the very last step, either one more Steiner point is added just outside the boundary of $R$, or two more are added within $R$. Therefore, the total number of Steiner points added is at most $t + 2$. Furthermore, for every two nodes eliminated from $T$, at most three strictly convex quadrilaterals are constructed. At the very last step, two additional quads may be constructed. This gives us the upper bound of $\left\lfloor \frac{t^2}{4} \right\rfloor + 2$ for the number of strictly convex quadrilaterals in the quadrangulation. At the end of phase $l$ of the algorithm, all nodes of the set $V_l$ have been eliminated, and hence the depth of the tree decreases by at least one. Furthermore, phase $l$ of the algorithm examines only the nodes at levels $l, l - 1$, and $l - 2$. In other words, each node in $T$ gets examined only a constant number of times. Therefore, the algorithm runs in $O(t)$ time. The space requirements are clearly $O(t)$ as well.

**3.2. Small Polygonal Regions**

In this section, we list several lemmas pertaining to strictly convex quadrangulations of small and simple polygonal regions, i.e., regions consisting of 4, 5, 6, or 7 boundary edges and no holes. As seen in the previous section, these facts are necessary to prove the correctness of our algorithm. We merely state the lemmas here. Proofs are omitted due to lack of space (see details in [6] and [19]).

**Lemma 3.2.1.** [6] A hexagon can be decomposed into at most four convex quadrilaterals by using at most three Steiner points in its interior.

**Lemma 3.2.2.** [6] A quadrilateral with a point in its interior can be decomposed into at most five convex quadrilaterals by using at most three Steiner points in its interior.

For polygonal regions bounded by an odd number of edges, one of the boundary edges is designated as an outgoing edge. (The outgoing edge is simply the triangulation edge between the root of subtree $T_v$ and its parent in the algorithm described in the previous section.) When quadrangulating this region, all Steiner points except one are placed in the interior of the polygon, and one Steiner point may be placed on the outgoing edge. The resulting quadrangulation consists of strictly convex quadrilaterals, possibly with one leftover triangle adjacent to the outgoing edge. The following lemmas state the relevant facts formally:

**Definition 3.2.1.** Let $P$ be a pentagon and let $e$ be an edge of $P$. Given a triangulation $T$ of $P$ such that $V_T = V_P$, $T$ necessarily consists of three triangles, two of which are ears of $T$. Each of these ears shares two vertices and a distinct diagonal of $T$ with the third triangle, denoted by center triangle. Furthermore, the
edge \( e \) is said to be of type 1 with respect to \( T \) if it is the edge of \( P \) shared with the center triangle of \( T \). If \( e \) is not of type 1 and \( e \) is adjacent to the type 1 edge, it is said to be of type 2 with respect to \( T \). If \( e \) is neither of type 1 nor of type 2, then \( e \) is incident to the common vertex of all triangles of \( T \) and is said to be of type 3.

**Lemma 3.2.3.** [15] Let \( P \) be a pentagon and let \( e \) be the outgoing edge of \( P \). Then, given any triangulation \( T \) of \( P \) such that \( V_T = V_{P} \), we have the following:
1. If \( e \) is of type 1 with respect to \( T \), the pentagon \( P \) can be decomposed into two convex quadrilaterals and one triangle adjacent to \( e \) by adding one Steiner point inside \( P \).
2. If \( e \) is of type 2 with respect to \( T \), then \( P \) can be decomposed into three convex quadrilaterals and one triangle adjacent to \( e \) by adding two Steiner points inside \( P \).
3. If \( e \) is of type 3 with respect to \( T \) then, \( P \) can be decomposed into four convex quadrilaterals by adding two Steiner points inside \( P \) and one more on the edge \( e \).

**Lemma 3.2.4.** [15] Let \( P \) be a pentagon with a point in its interior and let \( e \) be the outgoing edge of \( P \). Then, the pentagon \( P \) can be decomposed into at most six convex quadrilaterals and one triangle adjacent to \( e \) by adding at most four Steiner points inside \( P \).

In our algorithm, a polygonal region \( S \) bounded by seven edges is obtained when the subtree \( T_v \) is a path of five nodes, with the middle node as the root of \( T_v \). This is the only case that results in a septagon, and the outgoing edge is always the edge of \( S \) belonging to the element corresponding to the middle node of \( T_v \).

**Lemma 3.2.5.** [15] Let \( S \) be a septagon such that \( S \) admits a triangulation \( T \), with \( V_T = V_S \), whose dual graph is a path. Let the edge of \( S \) contained in the middle triangle of \( T \) be the outgoing edge \( e \). Then, \( S \) can be decomposed into six convex quadrilaterals and one triangle adjacent to \( e \) by adding at four Steiner points inside \( S \).

### 3.3 Constrained Quadrilateral Meshes

Our algorithm for generating quadrilateral meshes of polygonal regions with holes can be extended in a straightforward manner to work with arbitrary constrained triangulations. The ability to handle such input is critical for our image registration application described in Section 5, as we need to construct quadrilateral meshes of polygonal approximations of brain structures that are modeled as nested polygonal regions (see Figure 10). Both the interior and the exterior of each polygonal region (except the exterior of the outermost one) is to be meshed and it is important to respect the boundaries of the polygonal regions in the quadrangulation.

Let \( T \) be a given constrained triangulation with \( t \) triangles. Let \( G \) be the dual graph of \( T \) (\( G \) does not include a dual edge when the corresponding triangulation edge is a constraint), and let \( h \) be the number of connected components of \( G \). In order to construct a convex quadrangulation that satisfies the given constraints, we build the spanning forest \( T = \{ T_i, T_2, \ldots, T_h \} \) of \( G \) and run our algorithm on each \( T_i \). The root node of each \( T_i \) represents a triangle adjacent to a boundary (constraint) edge \( e_i \) of the underlying triangulation \( T_i \). Let \( t_i \) denote the number of nodes in \( T_i \). From the algorithm and Theorem 3.1.1, it follows that \( T_i \) can be quadrangulated with at most \( \frac{3t_i}{2} + 2 \) quadrilaterals using at most \( t_i + 2 \) Steiner points. Note that if \( t_i \) is odd, one Steiner point is placed on the constraint \( e_i \) and the adjacent triangles are modified accordingly, so that the number of nodes for every \( T_i \) becomes even. In either case, we show that the total number of strictly convex quads in the constrained quadrangulation is at most \( \frac{3t}{2} + 3h \), obtained by using at most \( t + 2h \) Steiner points.

### 3.4 Implementation and Results

We implemented our algorithm using C++ and the open source Computational Geometry and Algorithms Library (CGAL) class library (http://www.cgal.org). Figure 9a shows a triangular mesh with 3346 triangles generated by Triangle, which is a constrained Delaunay triangular mesh generator with quality constraints, [16]. Figure 9b shows a quadrilateral mesh with 1762 quads obtained from the mesh in Figure 9a using our algorithm. This reduction in mesh size of about 60% has been observed in almost all our test cases. Figure 9b highlights the input triangular mesh grading preservation that is also present in [3, 13]. Our algorithm does not provide any theoretical guarantee on mesh element shape, and it can indeed generate meshes with a few poorly-shaped quadrilaterals. We can further improve mesh quality by using post-processing methods at the expense of runtime and mesh size. Figure 9c illustrates the result of post-processing the mesh in Figure 9b using angle-based smoothing [17], and topological clean-up, [18].

### 4. MESHES FROM IMAGES

Polygonal approximations of structures in a two-dimensional image can be obtained by performing two operations on the input image: image segmentation and boundary approximation. Image segmentation is the process of subdividing an image into its constituent parts, [14]. This operation makes it possible to extract the collection of all pixels corresponding to a particular structure of an image. The boundary approximation operation takes as input the collection of pixels of a particular segmented structure, searches for the pixels
on the structure boundary, extracts the closed polygonal curves (polygons) defined by the exterior vertices and edges of the boundary pixels, and then simplifies these polygonal curves. The simplification can be carried out by using a polygonal curve simplification algorithm, [19].

Figure 9: (a) Triangular mesh of “Lake Superior”. (b) Quadrilateral mesh of Lake Superior. (c) Mesh in (a) after post-processing.

The result of applying segmentation and boundary approximation operations to an image is a hierarchy of (nested) polygons, or contours, such that the contours of any two consecutive hierarchy levels define one or more polygonal regions. Each polygonal region is an approximation of a distinct segmented structure of the image, so any meshing algorithm for polygonal regions can be used to generate a mesh of the entire image from its set of contours. Figure 10 shows the set of contours obtained from a two-dimensional slice of a segmented human brain image volume.

Figure 10: Contours of a human brain image.

5. AN APPLICATION

Image registration is the process of finding a spatial alignment between two images so that corresponding features can easily be related, [20]. This spatial alignment is usually achieved by a combination of rigid and non-rigid image transformations. The former is used to globally align two images and it accounts for translational and rotational differences, while the latter is used to maximize the regional similarity between corresponding structures. Image registration has become a very important tool for image analysis, understanding and visualization in medical applications.

Broit, [21], developed a method for non-rigid image registration in which one image, modeled as an elastic continuum, is warped to match the appearance of another. Later, Gee and Bajcsy, [22], proposed a variational and probabilistic framework to numerically compute a finite element-based solution for the registration problem using Broit’s method. An implementation of Gee and Bajcsy’s framework is available in the open source Insight Segmentation and Registration Toolkit (ITK), http://www.itk.org, sponsored by the National Library of Medicine (NLM). This implementation takes as input a pair \((A, B)\) of images and a mesh for image \(B\), and outputs an image \(A'\) resulting from warping image \(A\) to match the appearance of image \(B\). Figure 11 illustrates the FE-based image registration method implemented in ITK. Here, we adopt the ITK implementation of Gee and Bajcsy’s FE-based image registration framework to evaluate the quality of quadrilateral meshes produced by our algorithm in Section 3, as well as the quality of their triangular counterparts and regular rectangular grids automatically generated by the registration software.

We set up an experiment for registering a pair \((A, B)\) of 2D images of the human brain in which \(A\) was a coronal MR image with dimensions equal to \(256 \times 256\) pixels and \(B\) was the result of applying a cubic polyno-
mial warp to $A$. Next, we used the approach in Section 4 to compute a polygonal approximation for the structures of interest in image $B$, and generated several triangular and quadrilateral meshes for this polygonal approximation. We used Triangle, [16], to produce four triangular meshes whose triangles have minimum angles of $20^\circ$, $25^\circ$, $30^\circ$, and $33^\circ$. Then we produced quadrilateral meshes from these triangular meshes using the algorithm in Section 3. Next, we obtained four more quadrilateral meshes by smoothing and improving the topology of the previous four meshes. We also used an internal procedure within ITK to generate four regular grids of $8 \times 8$, $4 \times 4$, $2 \times 2$, and $1 \times 1$-pixel elements. Table 1 shows the number of elements, vertices, and edges of all meshes used in our experiment. Meshes 1–4 are triangular meshes with minimum angle $20^\circ$, $25^\circ$, $30^\circ$, and $33^\circ$, respectively. Meshes 5–8 are quadrilateral meshes generated from meshes 1–4. Meshes 9–12 are the result of post-processing meshes 5–8. Meshes 13–16 are rectangular grids of $8 \times 8$, $4 \times 4$, $2 \times 2$, and $1 \times 1$-pixel elements, respectively.

![Figure 11: (a) Source image. (b) Target image and its associated mesh. (c) Image resulting from warping image in (a). (d) Subtraction of (c) from image in (b).](image-url)

Table 1: Size of the meshes 1–16.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>#Elements</th>
<th>#Vertices</th>
<th>#Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2921</td>
<td>1472</td>
<td>4392</td>
</tr>
<tr>
<td>2</td>
<td>3549</td>
<td>1790</td>
<td>5338</td>
</tr>
<tr>
<td>3</td>
<td>4914</td>
<td>2481</td>
<td>7594</td>
</tr>
<tr>
<td>4</td>
<td>8254</td>
<td>4113</td>
<td>12426</td>
</tr>
<tr>
<td>5</td>
<td>1645</td>
<td>1617</td>
<td>3301</td>
</tr>
<tr>
<td>6</td>
<td>1941</td>
<td>1917</td>
<td>3897</td>
</tr>
<tr>
<td>7</td>
<td>2581</td>
<td>2605</td>
<td>5185</td>
</tr>
<tr>
<td>8</td>
<td>4318</td>
<td>4364</td>
<td>8681</td>
</tr>
<tr>
<td>9</td>
<td>1773</td>
<td>1785</td>
<td>3557</td>
</tr>
<tr>
<td>10</td>
<td>2073</td>
<td>2089</td>
<td>4161</td>
</tr>
<tr>
<td>11</td>
<td>2747</td>
<td>2771</td>
<td>5517</td>
</tr>
<tr>
<td>12</td>
<td>4499</td>
<td>4545</td>
<td>9043</td>
</tr>
<tr>
<td>13</td>
<td>1024</td>
<td>1089</td>
<td>2112</td>
</tr>
<tr>
<td>14</td>
<td>4096</td>
<td>4225</td>
<td>8320</td>
</tr>
<tr>
<td>15</td>
<td>16384</td>
<td>16641</td>
<td>33024</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>66049</td>
<td>131584</td>
</tr>
</tbody>
</table>

Examination of the results in Tables 1 and 2 demonstrates that the larger the size of a particular type of mesh, the smaller the associated RMS. In addition, quadrilateral meshes 5–8 have less than 60% of the number of elements of their triangular counterparts, meshes 1–4. The use of post-processing techniques to obtain meshes 9–12 from meshes 5–8 produced meshes with approximately 10% more quadrilaterals. Even so, the number of quadrilaterals in meshes 9–12 is still less than 61% of the number of triangles in corresponding meshes 1–4. Despite this reduction in mesh size, the RMS associated with the quadrilateral meshes and their counterparts are comparable. The RMS associated with the quadrilateral meshes is bounded from above and from below by the RMS associated with their triangular counterparts sampled with 1 and 3 integration points, respectively. Note that post-processing techniques in general improved the quadrilateral meshes. The runtime associated with the quadrilateral meshes, however, is higher than that of the triangular counterparts. This is due to the fact that the procedures to compute the finite element solution using 4-noded quadrilaterals are more expensive.
than the simple ones associated with 3-noded triangles.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Int. Pts.</th>
<th>Runtime (s)</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
<td>17.95</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>17</td>
<td>17.25</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>12</td>
<td>17.35</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>21</td>
<td>16.98</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>17</td>
<td>17.20</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>30</td>
<td>16.81</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>32</td>
<td>16.78</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>52</td>
<td>16.62</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>21</td>
<td>17.67</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>24</td>
<td>17.12</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>33</td>
<td>16.93</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>59</td>
<td>16.68</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>23</td>
<td>17.44</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>27</td>
<td>17.47</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>35</td>
<td>16.92</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>62</td>
<td>16.66</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>12</td>
<td>18.56</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>46</td>
<td>16.99</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>205</td>
<td>16.11</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>1001</td>
<td>15.93</td>
</tr>
</tbody>
</table>

Table 2: Summary of the registration results.

Quadrilateral meshes 7 and 11 have about 75% as many quadrilaterals as the regular rectangular grid 14, yet both meshes have a smaller RMS. This is an example of a case in which mesh regularity and well-shaped elements alone were not enough to provide a better result. The registration method is very sensitive along the boundary of distinct structures (see Figure 11d), and appropriately graded meshes can provide a better result. Figure 12 shows triangular mesh 3. Figures 13a and 13b show the quadrilateral mesh 7 produced from mesh 3 by the algorithm in Section 3, and quadrilateral mesh 11 that is the smoothest and topologically improved version of mesh 7, respectively.

6. CONCLUSIONS

We presented an algorithm to convert triangulations of polygonal regions with or without polygonal holes into strictly convex quadrangulations. Our algorithm has a runtime linear in the number of triangles of the input triangulation, offers better bounds than similar algorithms ([3]) that also produce strictly convex quadrilateral meshes of bounded size, and is simpler and faster than algorithms that produce better quality meshes (in terms of element shape, regularity and directionality control) at the expense of runtime, [7, 13, 8]. We also evaluated the quality of the meshes generated by our algorithm, their triangular counterparts and regular rectangular grids with respect to the performance of a FE-based image registration method. Our evaluation demonstrated that our quadrilateral meshes lead to slightly more accurate registrations when compared with those obtained using rectangular grids of similar size, and also lead to solutions comparable with the ones obtained using their denser triangular counterparts.

Future work will focus on the investigation and formalization of the relationship between the quality of the input triangulation and the quality of the corresponding output quadrangulation obtained by our algorithm with respect to quality measures such as angle bounds, etc. We also intend to extend the experiment in Section 5 to include an algorithm that is known to generate well-shaped quadrilateral meshes, such as the ones in [7, 13, 8], as well as to investigate an extension of our meshing algorithm to produce hexahedral meshes between planar cross-sections of image volumes, a problem directly motivated by applications in medical imaging.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their comments. The first author’s research is partially supported by NSF Grant No. CCR-0204293, and the second author is partially supported by CNPq, Brazil.

References


Figure 12: Triangular mesh with minimum angle 30°.

Figure 13: Mesh 7.

Figure 14: Mesh 11.
Invited Speaker
Abstract:

Surface Approximation and Remeshing in Computer Graphics

Efficiently managing complex surface geometry is of fundamental importance in almost all graphics applications dealing with real-world data. At the heart of many of the problems encountered in such applications is the need to automatically adapt surface triangulations produced by laser scanners, isosurfacing of volume data, and similar systems. This talk will explore some of the primary techniques developed in this area, particularly surface simplification and remeshing methods. We will focus on the close connections between simplification/remeshing, parameterization, and graph partitioning.
Session 2A
Surface Meshing
ABSTRACT

The visualization of 3D vector and tensor fields in a 2D image is challenging because the large amount of information will either be mixed during projection to 2D or lead to severe occlusion problems.

In this work we segment from the symmetric 3D tensor field regions dominated by stream tubes and regions dominated by diffusion surfaces. The diffusion surfaces are integrated with a higher order Runge–Kutta scheme and approximated with a triangle mesh.

Our main contribution is to steer the integration with a face-based coding scheme, that allows direct compression of the integrated diffusion surfaces and ensures that diffusion surfaces of any topology can be created.

Finally we sample the stream tubes and diffusion surfaces with points. The points from different entities are colored with different colors. We lit the points during rendering with a lighting model adapted to the tensor field. The resulting visualization of symmetric 3D tensor fields is sparse because of the sampling on points and allows for a deeper view inside the volumetric tensor field but also allows the simultaneous visualization of a dense set of tubes and surfaces.

Keywords: Tensor Field, Surface Integration, Surface Meshing, Visualization, Point Rendering, Diffusion Surfaces

1. INTRODUCTION

The visualization of 3D vector and tensor fields in a 2D image is challenging because the large amount of information will either be mixed during projection to 2D or lead to severe occlusion problems.

A lot of work has been done to visualize vector fields. Stream lines and stream surfaces are popular visualization techniques. A stream line is a curve where for every point on the curve the associated vector is tangent to the curve. One can imagine a stream line as the path that a particle takes through the vector field. Stream lines do not intersect each other except for points where the vector field vanishes or is undefined.

A stream surface is the path that a curve takes through the vector field and can be thought of as the dense collection of stream lines, all starting at a given curve.

The situation changes slightly if we look at symmetric 3D tensor fields. Throughout this paper we use the term tensor for symmetric 3D tensors. Symmetric 3D tensors play a great role in physics or medicine as for example diffusion tensors are symmetric 3D tensors. At every point a tensor field contains a (symmetric) tensor, i.e. a symmetric $3 \times 3$-matrix, instead of a single vector.

Eigenvector decomposition of the tensor is a popular approach to analyze a tensor. A symmetric tensor can always be decomposed into a diagonal matrix $\Lambda$ with the three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ on the diagonal and an orthonormal rotation matrix $V$ with $VV^t = 1$, with the unit matrix $1$:

\[
\forall T \in \mathbb{R}^{3 \times 3} \quad \text{with} \quad T = T^t : \\
\exists V, \Lambda \in \mathbb{R}^{3 \times 3} \quad \text{with} \quad V V^t = 1, \lambda_{ij} = 0 \Leftrightarrow i \neq j : \\
T = \Lambda V V^t.
\]

The columns $v_i \triangleq V_i$ of $V$ form an orthonormal basis of $\mathbb{R}^3$ and are called the eigenvectors. The combination of eigenvectors and eigenvalues $(V, \Lambda)$ is called the eigensystem of the tensor. If a unit sphere is scaled in the direction of the eigenvectors with the eigenvalues we obtain an ellipsoid that can be used to visualize the tensor. In the case of diffusion tensors the ellipsoids describe for any possible direction...
the rate of diffusion. Particles would have to be traced in all possible directions with speeds given by the ellipsoid. A diffusion tensor can be imagined as a description of how a spherical water drop is diffused into an ellipsoid. The terms stream line and stream surface can therefore not so easily applied to tensor fields.

Diffusion tensors often degenerate over large regions, such that their ellipsoids look like cigars or like pancakes. In the case of a cigar we speak of linear anisotropy, when one eigenvalue dominates the other two. In the other case we speak of planar anisotropy and two eigenvalues are much larger than the last one. If all eigenvalues are of similar size the corresponding region of the tensor field is called isotropic.

The eigenvector with the largest eigenvalue is called the major eigenvector, the eigenvector with the smallest eigenvalue is called the minor eigenvector, and the eigenvector with the medium eigenvalue is the medium eigenvector.

Given the anisotropy of tensors, the domain of a tensor field can be partitioned into linear, planar, and isotropic regions. Every region only contains tensors of the one specific type of anisotropy.

Within linear regions, a tensor field can be interpreted as a vector field formed by the major eigenvectors. Thus, we can define stream tubes as tubes whose middle axis is a stream line. Every point on the stream line is tangential to the major eigenvector of the tensor at this point. The cross section of the tube is defined by the medium and minor eigenvectors which are perpendicular to the major eigenvector.

Within planar regions, a diffusion surface can be defined as surface whose tangential plane for every point is the plane defined by the major and medium eigenvectors, i.e. it is normal to the vector field of the minor eigenvectors. We use the term diffusion surface here because the term stream surface may be misleading to what a stream surface is for a vector field.

We segment from the symmetric 3D tensor field regions dominated by stream tubes and regions dominated by diffusion surfaces. We reconstruct a dense set of stream tubes and diffusion surfaces and point sample them in way that the distance between two points is inversely proportional to the diffusion rate. Thus the points are closely spaced along a stream tube and sparsely orthogonal to it. On a diffusion surface the points are closely spaced over the surface but the surfaces are further apart from each other. The human eye of the observer will automatically merge close points to tubes and surfaces. The points from different entities are distinguished by their color. We lit the points during rendering with a lighting model adapted to the tensor field. The resulting visualization of symmetric 3D tensor fields is sparse because of the sampling on points and allows for a deeper view inside the volumetric tensor field but allows on the other hand the simultaneous visualization of a dense set of entities.

2. RELATED WORK

**Isosurfaces** The marching cubes algorithm [1] can extract an isosurface from a scalar field. An isosurface is thereby defined as the collection of points with equal scalar values. The marching cubes algorithm creates a triangular mesh that approximates the isosurface. For every vertex of the triangular mesh, a normal is calculated to perform lighting during rendering.

The basic algorithm loops over all (small) cubes whose corner points are in the center of eight voxels of the discrete scalar field. For every such cube the isosurface is located within the cube. There are $2^8 = 256$ ways how the surface may intersect with the cube. Every case is triangulated. A surface intersects an edge of the cube when one vertex is outside and the other vertex is inside the surface.

**Stream Surfaces** Stream surfaces from vector fields can be represented as parametric surfaces. A simple approach to construct such parametric stream surfaces is to place a number of seed points onto the original curve, to trace these points along their stream lines, and to connect the resulting stream lines. Remember that stream lines of vector fields can never cross each other.

This approach has many drawbacks. For example, consider converging or diverging stream lines. Connecting diverging stream lines may result in slivers or even don’t approximate the surface correctly because of the large area that is approximated linearly by the triangle.

Hultquist [2] improves this parametric stream surface approach. An initial set of stream lines is traced and as the tracing proceeds, additional stream lines are introduced in the case of diverging stream lines, or stream lines are removed on cases of converging stream lines.

Scheuermann et al. [3] presented an algorithm that is related to Hultquist. The stream surfaces are calculated for tetrahedral grids. A surface is propagated through a tetrahedra, calculating the intersections between the surface and the tetrahedra. The surface within a tetrahedra is a ruled surface, which means that the surface is generated by two stream lines that are blended by line segments. All calculations are done in barycentric coordinates of the tetrahedra which simplifies the formulas.

Van Wijk [4] models stream surfaces as implicit functions instead of parametric surfaces. A stream surface is given by the implicit function $f(x) = C$ where $C$ is a (scalar) constant. The difficulty is to find the function $f$. Once $f$ is found, a family of stream surfaces can be generated efficiently by varying $C$.

Van Wijk calculates $f$ from the convection equation for incompressible flow with $f$ as the transported quantity and $\vec{v}$ as the velocity:

$$\frac{\partial f}{\partial t} = -\nabla f \cdot \vec{v}$$
A range of values is placed along the boundary as initial (boundary) values. The convection equation is then solved over time using some numerical integrator until a steady state is reached. One can imagine this process as inserting varying concentrations of ink along the boundary of water. After some time, this ink is distributed (not necessary evenly!) within the water and the stream surfaces are the areas of equal concentration. Once the implicit function $f$ is calculated, the isosurface for a particular value $C$ is extracted and rendered with the marching–cubes algorithm.

In the case of diffusion surfaces the implicit approach is not possible. In the implicit approach the surface normal corresponds to the gradient $\nabla f$ of the implicit function. The diffusion surfaces are defined to be orthogonal to the vector field of the minor eigenvectors of a tensor field. Therefore, we are given $\nabla f$ and need to find $f$. From the Helmholtz–Hodge decomposition follows that this is possible only if the rotation of the vector field of the minor eigenvectors vanishes. As the rotation does not vanish for arbitrary symmetric tensor fields, the implicit approach is not possible. Figure 7 shows the diffusion surface of a tensor field with a planar region where the minor eigenvector field has quite a lot rotation.

**Diffusion Surfaces** There hasn’t been much work done yet to extract diffusion surfaces from tensor fields. Zhang et al. [5] presented a technique to extract stream tubes and diffusion surfaces from volumetric diffusion tensor MR images. Stream tubes are extracted in linear regions and diffusion surfaces in planar regions. So stream tubes represent structures with primarily linear diffusion while diffusion surfaces represent structures with primarily planar diffusion. Additional information is encoded in the color and cross section of the stream tubes.

Our approach of extracting diffusion surfaces is similar to [5]. That’s why this approach is discussed in more detail. Zhang et al. generate a dense set of stream tubes and diffusion surfaces and call them later using a set of metrics.

Linear regions are interpreted as vector fields formed by the major eigenvectors of the tensors. Thus the trajectory of a stream tube is a stream line through this vector field. The MR images are interpolated using tricubic B-Splines to get tensors not only at the sample points of the MR images. Zhang et al. generate seed points for every sample point within a linear region and jitter them within the voxel. The stream tube starts at a seed point and follows the major eigenvector field both forward and backward. An second-order Runge-Kutta integrator is used to track the stream line.

Diffusion surfaces are extracted from planar regions. The major and medium eigenvectors of a tensor at a point in space define the tangential plane of the surface at this point. Again, the seed points are placed into the voxels by jittering the sample points.

Starting from a seed point $v$, six initial search directions are distributed evenly around $v$. Every search direction is tracked and thus follows the shape of the surface. A triangle is created for every pair of neighboring edges. This first step creates a triangle fan consisting of six triangles.

From every vertex $u$ new search directions are created by projecting the triangles that are adjacent to $u$ onto the tangential plane $P(u)$ and the initial directions in $P(u)$ that are not covered by triangles. This is repeated for every newly generated vertex.

The new search directions of a vertex $u$ are traced through a vector field which is defined as the linear combination of the normalized major and medium eigenvectors which lies on a Plane $P_t$ that is both perpendicular to the tangential plane $P(u)$ at $u$ and contains the search direction.

The extension of the diffusion surface stops if it gets out of the data boundary, hits a low planar region, enters a region of low signal-to-noise ratio, or incurs a high curvature term.

While rendering, color is mapped onto the surface to represent the planar anisotropy.

### 3. POINT-BASED TENSOR FIELD VISUALIZATION

#### 3.1 Volume Segmentation

As already mentioned in the introduction, the tensor field domain is partitioned into linear, planar, and isotropic regions. We use three quantities of a (diffusion) tensor to define this partition as suggested by [6]:

\[
\begin{align*}
  c_l &= \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3} \\
  c_p &= \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3} \\
  c_s &= \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}
\end{align*}
\]

where $c_l$ measures linear anisotropy, $c_p$ planar anisotropy, and $c_s$ isotropy. Note that $c_l + c_p + c_s = 1$. $\lambda_i$ are the eigenvalues of the tensor with $\lambda_1 \geq \lambda_2 \geq \lambda_3$. The greater $c_l$ is the more the ellipsoid looks like a cigar (and the smaller $c_p$ and $c_s$ are). Similarly, the greater $c_p$ is the more the ellipsoid looks like a pancake. Finally, if $c_s$ equals 1, the ellipsoid becomes a sphere (all eigenvalues are 1 and $c_l$ and $c_p$ are 0), see figure 1.

After this segmentation of the volume, we can trace stream lines in linear regions and diffusion surfaces in planar regions. We use thresholds on $c_l$ and $c_p$ to classify regions.
A tensor can be classified as being isotropic \((c_l = 0, c_p = 0, c_s = 1)\), planar \((c_l = 0, c_p = 0.61, c_s = 0.39)\), or linear \((c_l = 0.57, c_p = 0, c_s = 0.43)\).

3.2 Distributing Points

We render stream tubes and diffusion surfaces as collections of points. Different colors are used to distinguish points from different entities. The tubes and surfaces shall be point sampled with the density described by the inverse of the diffusion rate given by the symmetric tensors.

Depending on the entity, that points were sampled from, the points are lit differently. Points from stream tubes are lit with the lighting model for lines as proposed by Zöckler et al. [7], whereas points on the diffusion surfaces are lit with the standard Blinn-Phong lighting model provided by OpenGL.

3.3 Stream tubes

Similar to [5] we look at the tensor field in linear regions as a vector field consisting of the major eigenvectors of the tensors. This vector field is traced.

We subdivide the volume into a regular grid which may be given automatically by the resolution of the image data. Otherwise, the resolution of the grid is specified explicitly.

To create stream tubes, we first create a stream line for every stream tube. This stream line is the trajectory of the latter stream tube.

We place a seed point into every grid cell. The tensor of this seed point needs to have linear anisotropy. We integrate a stream line starting at the seed point into both forward and backward direction using a second-order Runge-Kutta integrator [8]. A stream line consists of a list of ordered points \(p_i\) and is linearly approximated as a line segment between two successive points \(p_{i-1}\) and \(p_i\).

We want to place the points along a stream line such that the density of the points on a stream line represents the linear anisotropy of the tensors involved as described above.

To realize this behavior, we control both the arc length of the integration process and the step width of the second-order Runge-Kutta integrator.

The integration of the stream line stops if any of these cases happens:

- Outside of data volume.
- Left region of linear anisotropy \(c_l > C_l\) where \(C_l\) is the threshold for linear anisotropy.
- Extended point is "too" close to a previously calculated point.

The third point is motivated by artificial datasets where a stream line may be a (closed) circle. In order to stop the integrating process, the integrator needs to check if it reaches a part of the stream line that was previously integrated. For a fast local access to the points and line segments of the stream line, we sort the extended points into an octree. The integrator just needs to look up the octree to find proximate points.

A stream tube follows the trajectory defined by the stream line. Our approach is to render the (extended) points of the stream line only instead of rendering the whole tube around the stream line.

3.4 Diffusion surfaces

A crucial point for our visualization technique is to distribute the points across diffusion surfaces. We want the points to be distributed according to the diffusion rate over the diffusion surface. The higher the diffusion rate, the closer have to be the points. Remember that we want to render points instead of shaded surfaces. The human eye connects points that lie closer together and therefore follows automatically the more likely diffusion direction.

Although we are only interested in rendering points, for the integration of the diffusion surface it is advantageous to also build a triangle mesh to ensure a proper diffusion surface. Furthermore the connectivity information allows for smoothing and successive remeshing steps.

Although the diffusion surfaces do not have to be isosurfaces of an implicit function, they have to be manifold for differentiable tensor fields. This can be easily shown from the def-
inition. The diffusion surface is restricted to the subdomain of the tensor field, which is classified planar. Therefore, the minor eigenvector is defined everywhere and varies continuously over the planar subdomain as we supposed the tensor field to be differentiable. As the minor eigenvector uniquely defines the normal direction of the diffusion surface, the diffusion surface has to be manifold with border loops at the border of the planar domain.

4. DIFFUSION SURFACE INTEGRATION

As the diffusion surface has to be manifold with border, the integration process that approximates the diffusion surface via a triangle mesh is very similar to the encoding or rather decoding process of a face-based compression scheme. We used exactly the same building scheme to build the triangular diffusion surface. There are three advantages with this approach. Firstly, we can re-use the minimum set of building operations that allow to create manifold meshes of arbitrary genus. Secondly, we can re-use the data structures used for face-based coding such that the implementation of the diffusion surface integrator becomes very simple. And the third advantage is that we can directly encode the triangular diffusion surface into a space efficient representation, such that we can easily build in-core a large number of diffusion surfaces of high resolution.

We used a face-based compression scheme similar to the cut-border machine [9] and the edge-breaker [10] (see also [11] for an introduction to face-based coding) for coding and to steer the integration. A short review of the method and the basic building operations are given in the next subsection.

4.1 Face-Based Coding of Triangle Meshes

Face-based coding techniques compress triangle meshes which consist of a list of vertices and a list of triangles, each triangle containing three vertex indices and the indices of the edge-adjacent triangles.

The schemes are based on a region growing traversal of the triangle mesh. The traversal begins for example with an arbitrary seed triangle. The border of the growing region is called the cut-border. It divides the mesh into the inner and the outer part, which contain the already processed and the untouched triangles respectively. Triangles are added to the inner part at a distinguished current cut-border edge which is called the gate. After each addition of a triangle the gate moves on to another cut-border edge, until all triangles of an edge-connected component of the triangle mesh have been compressed. This is done for each edge-connected component. The choice of the next gate location defines the traversal order and steers how the cut-border grows over the mesh.

The face-based coding scheme encodes a bit-code each time a new operation is added. The decoding performs the same traversal and builds the face according to the encoded operation. The different possible operations by which the next triangle is incorporated into the inner part at the gate are illustrated in figure 2. The center operation C in (a) adds a new triangle to the growing region that is incident only to the old gate and to a new vertex. The gate is moved to the right newly added cut-border edge such that it cycles around its target vertex. The current face in the right/left operation R/L in (b/c) is incident to the gate and the next/previous edge on the cut-border. The neighborhood of the pivot vertex is closed and a new pivot vertex is chosen with the new gate location. In the end operation E in (d) all edges of the current face are incident to the cut-border and the cut-border closes. The other growing operations describe cases when the third vertex of the current face is on the cut-border. (e) shows the split operation S, where the cut-border grows into itself and is split into two loops with two gate locations. One cut-border loop is pushed onto a stack and processed after the other one is eliminated by an end operation. In order that the decoder can replay the split operation the position of the third vertex relative to the gate is encoded. The hole operation H (f) merges the current cut-border with a border loop. We will handle border loops in a different way as done by the cut-border machine [9]. We encode a border operation B, whenever the gate is a border edge of the mesh. As triangle meshes describe two dimensional surfaces in three dimensional space, two cut-border loops can grow together again, actually once for each handle of the triangle mesh. The operation which unifies two loops is called merge operation M (g). It merges the current cut-border loop with another cut-border loop and takes two indices, the index of the other cut-border loop and the location inside that other loop.

The cut-border data structure consists of a stack of doubly linked lists of cut-border edges. Each cut-border edge stores
**Input:** seed location \( x \)
integrate edge from \( x \) to \( y \)
init cut-border to \((x, y)\)
while cut-border not empty
    choose gate
    decide operation
    apply operation to cut-border

**Figure 3:** Structure of the integration algorithm.

the indices of its start vertex and of its adjacent triangle in the inner part. The initialization creates a cut-border with three edges. Each C,S or M operation inserts a cut-border edge after the gate. The R and L remove the next or previous cut-border edge and the E operation closes the loop on top of the stack.

4.2 Cut-Border Based Surface Construction

As the face-based coding scheme can encode any manifold mesh, one can also build any triangulated diffusion surface during the integration.

The input to our diffusion surface integrator is a seed location in a planar region of the volumetric domain of the tensor field. We integrate the diffusion surfaces with the face-based building operations C,L,R,E,S,M,B. Opposed to the face-based scheme we start the building process with a single edge and initialize the cut-border to a loop of two cut-border edges around the edge like the face-fixer proposed by Isenburg [12]. This first edge is created by integrating from the seed location in the direction of the major eigenvector of the tensor field. The integration of a new edge is described in the next subsection in detail.

The diffusion surface is built from the initial cut-border by extending it at one of the cut-border edges, which is called the gate. The choice of the gate steers how the cut-border grows the diffusion surface and is described in subsection 4.4 in detail.

We summarized the integration algorithm in figure 3. After the cut-border has been initialized from the seed location we successively select a gate edge, decide which of the operations C,L,R,E,S,M,B to perform and apply it to the cut-border until no cut-border edges are remaining. This can either happen, when the diffusion surfaces closes up or when all cut-border edges were transformed into border edges of the mesh by a B operation at the boundary of the planar domain of the seed location. Figure 4 illustrates the growing process for a spherical diffusion surface.

In this subsection it remains to explain how we decide for a given gate location, which operation has to be performed. We first propose one of the operations E,L,R,B or C and then check for E,L,R or C if we should not have performed an S or M instead.

First we decide if we should propose an L,R or E operation based on the exterior angles between the gate edge and the previous or next edge on the cut-border as depicted in Figure 5 a). If one of the angles \( \alpha_0 \) or \( \alpha_1 \) is smaller than a threshold angle, that we set to seventy degrees, we decide for an L or R operation. When the length of the current cut-border loop is only three, we decide for an E operation instead of L or R.

If both angles are larger than the threshold, we try to propose a B operation by trying to integrate the left edge of the new triangle as shown in Figure 5 b). The direction for integration should be in the planar case always sixty degrees. For curved surfaces this is not possible anymore. Here we chose to subdivide the angle \( \alpha_0 \) into \( k \) equal parts such that the resulting angle is closest to sixty degrees. In the example of Figure 5 b) \( k \) is two such that we chose the direction \( \alpha_0/2 \) away from the gate.

The integration process is in the next subsection. It returns a target location or reports failure if the integration left the planar domain. In case of failure we decide for a border operation B, that will always be performed. Otherwise we can construct with the target point a new triangle labeled with C in the Figure 5 b).

Now we either perform a border operation or propose a new triangle added by E,L,R or C. This triangle can intersect a distant cut-border part as illustrated in Figure 5 c). Here we should rather perform a S or M operation with the vertex closest to the gate. Whether S or M is performed can be decided by the cut-border data structure from the vertex to which the gate will be connected. To find out if an S or M has to be performed, we entered all cut-border edges in an octree and checked for the proposed E,L,R and C triangles if they cover other cut-border edges. In order to also connect to close outside cut-border vertices we enlarge the proposed triangles by 30% before we checked in the octree.
covered cut-border edges and vertices we selected the cut-border vertex closest to the center of the gate and proposed an S or M operation. See Figure 5 c) for an illustration of this process.

4.3 Integration of New Edges

For every search direction, we define the search plane as the plane that is orthogonal to the tangential plane at the start point, contains the search direction vector and the start point itself. This approach is similar to [5].

Every search plane defines a vector field along which we integrate from the start point. A vector of this vector field is, simply spoken, defined by the cut of the search plane and the plane which is spanned by the major and medium eigenvector of a tensor. The length of this vector is just the distance between the location of the tensor and the cut point of the search plane with the ellipse defined by the major and medium eigenvector, see figure 6.

To perform this operation quickly, we use the following simple mathematics. Let \( n \) be the normal of the current search plane, and \( T \) be the tensor matrix at the current point. The vector field is then defined by the following operations:

\[
\begin{align*}
\tilde{n} &= T^t n \\
n_\perp &= \begin{pmatrix} -\tilde{n}_1 \\ \tilde{n}_0 \\ 0 \end{pmatrix} \\
v_t &= T n_\perp
\end{align*}
\]

We transform the normal vector of the search plane into the coordinate system defined by the eigenvectors. That is just a matrix–vector multiplication. Because we want the vector to lie in the plane of the two largest eigenvectors, we project the vector into this plane by setting the last coordinate to 0. Then, an orthogonal vector is set up by multiplying the vector to a 90 rotational matrix. This can be done explicitly by just changing the coordinates. This orthogonal vector is then transformed back into the world coordinate system to be the vector of the vector field within the search plane.

The step width for the integration process of one search direction is adapted to the eigenvalues in order to distribute the points according to the anisotropy of the tensor field

\[
h = C \sqrt{\sum_i w_i^2 n_{\perp i}^2}
\]

where \( n_\perp \) is the vector orthogonal to the normal vector of the search plane as defined above, \( h \) is the step width, and \( C \) is a user-controlled constant which can further influence the density of the points. The weights \( w_i \) are chosen to represent the proportional behavior to the anisotropy as

\[
w_i = \frac{1}{\lambda_i}
\]

4.4 Traversal Order

In order to avoid a large number of S or M operations we used a similar strategy as proposed by Alliez and Desbrun [13]. The method is based on the observation that S operations arise very seldom at cut-border edges that are in a convex region. Therefore we measured for each cut-border edge the two exterior angles \( \alpha_0 \) and \( \alpha_1 \). The smaller these angles are, the more convex is the region around this cut-border edge. As it is already fine if one of the angles is small, we sorted the cut-border edges according to the minimum if the exterior edges into a priority queue. Each time a new gate has to be chosen, we extracted the most convex from the priority queue. After each basic operation we updated also the
cut-border edges adjacent to newly added or removed ones and re-positioned them in the queue. Figure 4 illustrates that the cut-border stays nicely shaped during the integration process.

5. RESULTS & ANALYSIS

We tested our algorithm with tensor fields that include singularities which need to be bypassed by the integration and meshing process. We created artificial tensor fields to simulate different behavior.

The Spiral example has singularities on the z axis. The tensors have planar anisotropy around the z axis, see figure 7. The diffusion surfaces are rendered as triangular meshes to show how the mesh generation follows exactly the diffusion surface and thereby bypasses the singularities.

The Sphere dataset, figure 4 d, presents the ability to handle self-intersections. If the mesh grows into a previously integrated region, the algorithm detects this intersection and connects the boundaries to build a triangular mesh.

A dataset similar to sphere, see figure 11, demonstrates the usefulness and power of the point-rendering approach. The interior stream lines are completely enclosed by a mesh but they remain visible.

The Sinosoidal example shows both stream tubes and stream surfaces within a dataset that varies anisotropy very frequently. Figure 9 shows a simple preview of the dataset displaying the major, medium, and minor axes of the tensors within a 10x10x10 grid. The color encode the different anisotropies. Green is planar anisotropy, red linear, blue isotropy, and gray is undefined. Figure 9 shows the point-rendered stream tubes and diffusion surfaces.

The crucial part of the performance of our algorithm is the integration process itself. Thus the step width of the integration step is the limiting factor and needs to be adapted very carefully to the anisotropy of the tensor field.

Note that the distribution of the points over the diffusion surface may not be optimal. We did not implement an optimization algorithm for placing the points. We can ensure that the whole diffusion surface is extracted because our approach starts with an initial triangle and grows it’s border until it reaches a non-linear region, and that the distance between points along edges that were integrated is optimal in terms of that this distance accords to the diffusion rate along this edge. But there may be edges that were not integrated but artificially inserted by the cut-border operations. The optimal distribution of points over a diffusion surface is topic of further research.

6. CONCLUSION

We presented an algorithm that tracks stream tubes and diffusion surfaces of tensor fields according to the anisotropy of the tensor field. The resulting points are rendered as point clouds where different entities are distinguished by color. Because the points of an entity are much closer than the distance between the single entities, the human eye is able to differentiate between these entities, additionally to the color coding. Furthermore, the point rendering enables a deeper insight into the volumetric information of the tensor field.

The generation of the diffusion surfaces is very similar to the decoding process of a faced-based compression scheme. Exactly the same building scheme can be used. This allows for a simple implementation and a compact representation of the generated mesh. The diffusion surface can thereby be of arbitrary genus.

References


**Figure 8:** The preview for the dataset shown in Figure 9. Green is planar anisotropy, red linear. Shown are the axes of the eigenvectors.

**Figure 9:** Stream tubes and diffusion surfaces rendered as point clouds.
Figure 10: A classic approach for visualizing tensor fields is to render ellipsoids. Note the lack of visibility due to occlusion of the ellipsoids.
Figure 11: The power of point rendering. The interior stream lines are still visible although they are encapsulated by a closed surface.
ABSTRACT

Generation of surface meshes remains an active research problem despite the many publications addressing this topic. The main issues which must be treated by a good remeshing algorithm are: element quality, sizing control, approximation accuracy, robustness and efficiency. One reason surface meshing is such a challenging problem is the fact that using the Euclidean metric to measure distances between points on the surface can generate large discrepancies between the original surface and the constructed mesh. We solve this problem by using geodesic distances on the surface. The ability to accurately and efficiently compute geodesic distances, and propagate them across the mesh, permits us to generate quality surface meshes which closely approximate the input without using costly parameterization techniques.

Keywords: Triangle mesh, surface meshing, geodesic distances.

1. INTRODUCTION

3D models are used in applications ranging from animation and cinematography to heavy industry and scientific visualization. However, most existing surface mesh models can hardly even be called satisfactory. Most of them are not sampled properly and their basic elements — triangles — have poor, almost random shapes because of the 3D mesh acquisition process. Whether this is done using interactive solid modeling software or semi-automatically using a scanning device, it remains a tedious and error-prone procedure. Models generated by CAD software usually reflect regular sampling of the underlying parametric domain instead of the model features. The process of simplifying scanned models with millions of points is primarily concerned with preserving model geometry and topology and does not emphasize the quality of sampling and triangles. This results in meshes which usually cannot be used as-is for 3D applications. An intermediate step, correcting the basic mesh geometry and connectivity, while preserving features, is required. Such corrections, commonly known as remeshing, are a fundamental component in the field of digital mesh processing.

Remeshing tries to accurately approximate the model geometry with well-shaped elements. It also adjusts the sampling rates locally to match the amount of detail present. High-quality meshes are necessary for engineers performing numerical computations, such as finite element analysis that, for example, calculate mechanical stress, solve heat and flow differential equations or simulate such systems. A high-quality mesh conditions the system well, eliminating numerical errors and singularities that might otherwise arise. Hence, within the engineering community the emphasis is on quality. The computer graphics and modeling community, on the other hand, is concerned with another aspect of remeshing. Their focus is on the tradeoff between the visual quality of the result, the speed of the remeshing operation, and the optimization of the number of polygons in order to achieve interactive rendering speeds.

Over the last decade, an abundance of remeshing algorithms have been proposed. One group of algorithms, e.g. [10,11], is based on partitioning the 3D mesh into patches, and treating each patch separately, usually with subdivision techniques. While these techniques yield reasonable results, they are very sensitive to the patch structure, and the vertex sampling (or distribution) is difficult to control. More recent remeshing algorithms, e.g. [2,9,12,14] are based on global parameterization of the original mesh, followed by a resampling of the parameter
domain. After this, the new triangulation is "projected" back into 3D space, resulting in an improved version of the original model. The main drawback of the global parameterization methods is the sensitivity of the result to the specific parameterization used. Embedding a non-trivial 3D structure in the parameter plane severely distorts this structure, and important information, which is not specified explicitly, may be lost on the way. Even if the parameterization minimizes the metric distortion of the 3D original in some reasonable sense, it is impossible to eliminate it completely. Moreover, global parameterization is very slow, usually involving the solution of a large set of (sometimes nonlinear) equations. Because of the size of the system, this solution may be numerically imprecise, especially in regions where the connectivity has bad isoperimetric ratios. These regions correspond to protruding extremities in the 3D mesh (e.g. the legs of an animal), and they may be lost in the process. Additionally, 2D parameterization requires the surface to be cut to a disk-like topology, both to parameterize closed surfaces or surfaces with genus higher than zero, and to reduce the parametric distortion. These cuts introduce visible artifacts in the mesh.

The main alternative to global parameterization is to work directly on the surface. Remeshing algorithms using this approach [2,8,16,18] usually involve difficult, inefficient and limited optimizations in 3D. For example, Frey and Borouchaki [8] perform local modifications in the tangent plane. In a subsequent work, Frey [7] uses a paraboloid to obtain a better approximation of the surface. These complex approximations are extremely slow and not always robust.

The common denominator of all the meshing algorithms is that they use Euclidean metrics, in the sense that the distance between points, even on the surface, is measured as the Euclidean distance between them. Since in most cases we should actually be using the geodesic distance, i.e. the distance between the points along the surface, this may introduce error culminating in distortion. The Euclidean distance may be considered a good approximation to the geodesic distances only at short ranges and in regions of low curvature. Otherwise it is quite different. Nonetheless, the published methods use Euclidean distances because they are much simpler to compute.

Assuming a constant sizing function, the vertices of the new mesh should be positioned on the surface such that the distances along the surface between close vertices are approximately equal. One way of achieving this is to build a "front" of vertices which advances across the surface at uniform "velocity" [3,17]. The front forms strips of triangles as it advances. The main problem with this method is that the front may meet itself, hence split and merge as it propagates. This complicates matters significantly, and a number of heuristics are required to control the process. The fact that Euclidean distances are employed here as well results in suboptimal results. Figure 1 shows the result of an advancing front technique implemented in a commercial package. The loss of high curvature features is a typical artifact. The result of applying our method on the same model with similar uniform sizing is shown in Figure 2(e). Due to these difficulties, advancing front techniques are not considered to be very attractive.

1.1 Our Contribution

This work introduces a novel remeshing method which operates directly on the 3D surface in a manner similar to the "advancing front" methods. So on the one hand, it does not involve any costly parameterization methods. On the other hand, it avoids all the pitfalls of the existing advancing front methods. Firstly, it uses geodesic distances instead of Euclidean distances. Secondly, it does not have to deal with major topological changes in the advancing front. This is achieved by segmenting the mesh into regions such that the treatment of each region is relatively straightforward. This also allows us to process meshes with arbitrary topologies.

2. ALGORITHM OVERVIEW

Our mesh generation procedure avoids the need for costly planar parameterization by computing accurate geodesic distances directly on the surface. The geodesic distances are computed using the "fast marching on triangulated domains" technique of Kimmel and Sethian [13]. We provide a brief overview of the technique in Section 3.

The basic meshing technique we use was first proposed by Adi [1]. It is based on generation of equidistant curves on the surface. An equidistant curve is the locus of all points on the surface at some fixed geodesic distance from a given point. Adi computed such equidistant curves from a single root point on the surface and then simply triangulated the strips between consecutive curves.

The difficulty with this simple approach, as with the advancing front techniques, is that equidistant curves may have complex topologies. The saddle regions where a single curve splits into several components can have arbitrary shape (Figure 2(a)). Without special treatment, the mesh in such regions will both disobey the sizing requirement and contain badly shaped triangles (Figure 2(b)). Adi did not provide a satisfactory solution to these problems, hence he was able to generate good meshes only for very simple models. We avoid this pitfall by first segmenting the surface into regions, such that the distance function is mono-
tone inside each region and therefore does not contain saddle points. Once the regions and the equidistant curves inside them are computed, each strip between two adjacent equidistant curves is meshed using a Voronoi tessellation of vertices distributed on the two curves. The next four sections describe the main components of the algorithm:

1. Computation of geodesic distances and equidistant curves.
2. Mesh preprocessing.
3. Segmentation into regions.
4. Triangle generation within each region.

The various stages of the algorithm are illustrated in Figures 2 and 4.

3. COMPUTING GEODESICS

The easy, but inaccurate, way to compute a "geodesic" distance between two vertices of a triangle mesh surface is to run a shortest-path algorithm on the mesh graph, where the weight associated with an edge is its length. Efficient algorithms, such as the Dijkstra algorithm [15], can compute these path lengths very efficiently, but can be shown to produce paths quite different from true geodesic paths. This is because the geodesic path does not necessarily pass through the mesh vertices, rather takes shortcuts through edges. See Figures 3(a) and 3(b) for a comparison. In our work we utilize the "fast marching on triangulated domains" algorithm of Kimmel and Sethian [13]. It computes approximate geodesic paths between two vertices in $O(n \log n)$ time per path ($n$ is the number of vertices in the mesh). Unfortunately, this algorithm does not always guarantee a correct result, in particular when the mesh contains triangles with obtuse angles. Kimmel and Sethian offer a solution to this problem, but it is rather complex and not always correct. Alternatively, the problem may be reduced by a preprocessing step which reduces the relative number of obtuse triangles in the mesh. The standard way of doing this is to refine the obtuse triangles so that most of the affected area is covered by smaller, but less problematic triangles. This is not guaranteed to remove all obtuse angles, but removing just the worst cases suffices to produce reasonably accurate geodesic distance computations. Fortunately, obtuse triangles do not occur so frequently, so the

Figure 2: The effect of mesh segmentation on geodesic remeshing with uniform sizing of the horse model of Figure 1(a). (a) Geodesic curves and zoom on a saddle region. The root vertex is on the back left foot (marked by a star) (b) Resulting mesh with artifacts in saddle regions. (c) Segmented regions. The centers of leaf regions are highlighted by circles. (d) Geodesic curves formed with two-site distances on segmented mesh. Note that the saddle of (b) has disappeared. (e) Resulting mesh with no artifacts. (f) Mesh after post-processing.
algorithm produces quite good results in general, even without these two workarounds, although we employ them both.

The fast-marching method can also be adapted to non-uniform geodesic distances. The input may contain an arbitrary weight per vertex, where larger weights mean that the region surrounding that vertex is "harder" to pass through. The geodesics then take this information into account. This is a feature which is very useful to us, as we will see later.

We use the fast-marching method to compute the equidistant curves. This is done by computing the geodetic distance from the source to all other vertices of the mesh. The equidistant curve is then formed by connecting linear segments between points on the edges at a given distance, these two interpolated linearly between vertex distances. This approximation is obviously not correct when large triangles are involved, since the equidistant curves will be quite different from what they should be. See Figure 3(c); the equidistant curves on a plane are not circles, as would be expected. Here too, a possible solution, which we adopted, is to refine the mesh to contain many smaller triangles, forcing the algorithm to output more detailed geodetic information. Obviously, this increases the computation complexity.

![Figure 3: Shortest paths between the two yellow vertices. (a) Dijkstra. (b) Geodesic. (c) Equidistant curves based on geodesic distances.](image)

4. MESH PRE-PROCESSING

Our meshing algorithm supports different sizing requirements on different regions of the mesh. Curvature-based sizing defined for each vertex is usually used to provide more accurate geometry approximation. Other per-vertex sizing data, such as those derived from analysis requirements, can be incorporated similarly. This data is sent as input to the fast-marching method, which conveniently, is able to use it.

We approximate the curvature at the mesh vertices using the method described in [5]. The sizing is then based on a combination of Gaussian and mean curvature. The relative weight of the two components is controlled by the user. In addition the user also controls the contrast or the gradient of the sizing gradation. This is achieved by transforming the sizing by some polynomial magnifying function.

The required number of triangles determines the desired edge length in the case of uniform sizing. This in turn determines the distance between consecutive equidistant curves on the surface.

This distance $C_D$ is the desired edge length scaled by $\sqrt{3}/4$ (the ratio between the height and the side in an equilateral triangle).

Now the root vertex for the distance computations is located. We compute the two vertices on the surface forming the maximal surface distance between them (the diameter) and select one of them as the root $r$. The computation of these two vertices is done using the following well known iterative procedure [6]:

1. Set $r$ to some arbitrary vertex. Set $D_{max}$ to zero.
2. Find the farthest vertex $v$ from $r$. Set $D_t$ to the distance between $r$ and $t$.
3. If $D_t > D_{max}$, set $r := t$, $D_{max} := D_t$, and goto 2.

This procedure is actually not guaranteed to find the mesh diameter, as it might get stuck in a local minimum, but for reasonably well-behaved meshes, this is rare. The root vertex $r$ and the maximal distance $D_{max}$ are used in the following stage to segment the mesh into regions. Figures 2(a) and 4(b) show the root vertex and the equidistant curves surrounding it for the horse and cactus models.

5. REGION SEGMENTATION

Once the root vertex $r$ is found we compute the geodesic distance $D(v)$ from $r$ to every other vertex $v$ on the mesh. $D(v)$ is then used to segment the mesh into regions to avoid mesh artifacts like those seen in Figure 2(b). The regions are constructed such that each equidistant curve within the region will be well-behaved. To guarantee this, each region must be a simply connected region. See Figures 2(c) and 4(f).

The regions form a tree structure containing three types of region nodes: leaves, interior nodes, and a root. The distinction between different types of regions reflects the properties of the distance function $D$. Leaf regions are formed around the maxima of $D$. Interior regions roughly correspond to the saddle points of $D$, and the root region is formed around the root vertex $r$ (the global minimum of $D$). The tree construction algorithm runs in two stages. First the topological structure of the tree (Figure 4(e)) is determined, and then the mesh regions corresponding to each node are computed (Figure 4(f)).

Tree structure construction: The set of vertices $L$ which are the local maxima of $D$ define the tree leaves (Figure 4(c)). Initially each leaf defines a degenerate tree consisting of a single leaf, producing a forest. A bottom up construction is performed with groups of trees connected by interior region nodes. Finally all the trees are joined into a single tree with a single root region node.

The tree construction uses a front propagation procedure on the surface starting from the set of leaf vertices $L$. The front propagation is based on the distance metric $D$. The fronts emanating from the leaf vertices are propagated so that at each step the vertex with smallest value of $D$ is added to the front of the appropriate leaf. When two fronts meet at a vertex, an interior region node is added to the tree as the parent of the two leaves (Figure 4(d)). The vertex is stored for further processing. The two fronts are merged, and the new front continues to advance using the minimal distance $D$ from among its leaves. Whenever two fronts meet, interior nodes are created joining the sub-trees (Figure 4(c)). When only one front remains, i.e. all the leaves are connected into a single tree, the root node is introduced as the common parent. At the end of this procedure a region tree
structure (Figure 4(e)) is defined, but the boundaries of the regions still need to be determined.

**Tree regions construction:** The region boundary definition proceeds yet again bottom up, from the leaves to the root. When considering a region, the *top* of the region is defined as:

- the center vertex \( l \) for a leaf region;
- the boundary curve between the region and its children for an interior or root region.

The region boundary construction proceeds as follows:

While not all the boundaries have been constructed, find an interior node at which both the following hold:

- the top boundary is not defined,
- in both of its sub-trees the top boundaries are defined for all the nodes.

Note that at the beginning of the procedure all interior nodes with two leaf children satisfy this condition. Let \( t_1 \) and \( t_2 \) be the tops of the root nodes of the sub-trees. The vertex \( v \) at which the fronts of the sub-trees meet is known from the tree structure construction stage above. We now use it to compute the top boundary of the interior node which is the parent of the two sub-trees, as follows:

- Compute the equidistant curve \( C \) at distance \( D(v) \) from the root \( r \). Note that \( C \) will contain \( v \).
- Find two vertices \( v_1 \) and \( v_2 \) on \( C \) at minimal distance from \( t_1 \) and \( t_2 \) respectively.
- Compute the two equidistant curves \( C_1 \) and \( C_2 \) at distance \( C_{d1}D(t_1, v_1)/C_{d1} \) from \( t_1 \) and at distance \( C_{d2}D(t_2, v_2)/C_{d2} \) from \( t_2 \) respectively. This distance roundoff makes the distance from the tops of the two sub-tree root node regions to their bottom a multiple of \( C_{d} \). It will result in an even curve distribution inside the regions at the meshing stage.
Connect $C_1$ and $C_2$ into a “spectacles” shape using the shortest path between them which passes through $v$. This provides a single loop as the top boundary of the parent node.

The procedure is continued upwards inside the tree until all the region boundaries are computed. Figure 4(b)-(e) illustrates this procedure on the cactus model, resulting in the regions in Figure 4(f). Figure 2(c) shows the regions for the horse model. Note that many leaf patches on the head and legs of the latter are generated around minor local maxima. As a result $D(l,v)<C_d$ and the patch degenerates to its center vertex $l$.

6. MESH GENERATION

We now proceed to place equidistant curves inside each region. The distance from the top of each leaf or interior region to its bottom is, by construction, a multiple of $C_d$. Hence curves can be spread evenly within each such region. Within the root region an error of up to $C_d/2$ can occur. The algorithm spreads it equally during curve distribution by slightly increasing or decreasing the curve offset. This is illustrated in Figure 4(g)-(i). The equidistant curves constructed during this process are incorporated into the surface and used to generate the final mesh.

![Image](image1)

**Figure 5:** Meshing strips on the cactus model. (a) Simple strip. (b) Strip with original triangles and new mesh vertices (circles). (c) Voronoi cells with new vertices as sites. (d) Dual Delaunay mesh. (e) "Spectacles" strip (f)-(h) Delaunay meshing. The T-junctions are marked by arrows.

After the curves are computed the mesh can be generated using a final bottom-up sweep thru the tree. Starting from the leaves, vertices are distributed along each equidistant curve at edge length intervals. These curves are the boundaries of the strips (and spectacles). These vertices are added to the surface mesh (Figure 5(b) and 5(f)). When distributing vertices on spectacles they are first placed at the T-junctions, which are the meeting points of two or more equidistant curves (Figure 5(f)). The vertices placed on the curve connecting these form vertices interior to the strip. The vertex placement is then adjusted to optimize the element shape. The final mesh is constructed by triangulating the strip between consecutive curves. To generate the triangles, the algorithm computes the Voronoi cells around each vertex using geodesic distances (Figure 5(c),(g)). The Delaunay triangulation dual to this Voronoi cell structure is computed and used as the mesh for the strip (Figure 5(d),(h)). The meshing stage can preserve surface features by incorporating feature edges into the mesh (see the gear model in Figure 7). Feature edges are detected using a dihedral angle criterion. The features are handled similarly to equidistant curves, with vertices distributed along them. Vertices are placed at intersections of feature edges with equidistant curves and the vertex distribution is adjusted accordingly. When meshing the strips, each piece of a feature edge which falls inside a strip is handled as part of the strip. The resulting strip triangulation contains both vertices distributed along the curves forming the strip and vertices distributed on the parts of feature edges overlapping the strip.

Once all the strips are meshed, the resulting mesh (Figures 2(e) and 4(j)) can be improved further using standard smoothing and local edge flip techniques (resulting in Figures 2(f) and 4(k)). This final stage distributes the vertices better on the surface, removing the visual "lines" formed by vertices placed along the equidistant curves. Even though these techniques improve the mesh regularity, they cannot be used excessively as they also cause damage to the mesh fidelity to the original.

7. EXPERIMENTAL RESULTS

We demonstrate the results of our remeshing method on several examples (in Figures 6 and 7). Figure 6 also compares to the result from the algorithm of [7]. The statistics for the resulting meshes are summarized in Table 1.

The sizing gradation control is demonstrated on the Venus head model (Figures 6 and 7). Our algorithm can successfully mesh both smooth and CAD-type models with corners and creases, as demonstrated by the foot and gear models. The foot model also demonstrates our method’s ability to handle significant sizing gradations resulting from large variations in curvature across short distances.

![Image](image2)

**Figure 6:** Remeshing methods comparison. All models contain approximately 5,400 vertices. (Left) Local technique [7]. (Right) Our method. The statistics for both appear in Table 1.
The example models showcase the method's ability to correctly capture sock-like shapes such as the animal's legs without requiring global parameterization. As is evident in the griffin and figure eight models, our method does not require any special treatment to handle models with genus greater than one. This, in addition to generating seamless meshes is yet another advantage of this technique over methods which utilize a parameter domain [2,9,12,14]. Our advantage over local methods [7] is in the mesh quality (Figure 6), with only a slight penalty in approximation error in some cases.

Both approximation and quality measures are shown in Table 1. The quality is demonstrated by the statistics of the minimal angle. The angles in the inputs are arbitrarily bad, but in most of the results, not many angles are less than 30° and the average angle is consistently above 50°. The statistics also include the Hausdorff distance from the original model measured using the Metro tool [4]. This is approximately 0.5% of the bounding box diagonal for most models, which is quite negligible. The superiority of using geodesic distance based advancing front instead of classical advancing front techniques in terms of approximation is clearly demonstrated by the horse meshes in Figure 1.

The run-time for the models varies between 60 seconds for the horse model to 600 seconds for the griffin on a 1.7 GHz Pentium 4 PC with 512 MB RAM. The time reflects the input and output sizes as well as the level of complexity which determines the number of regions generated by the algorithm. As demonstrated by the examples, the main advantage of our method is the ability to generate the number of triangles desired by the user while simultaneously approximating the input well and generating high quality elements.

### 8. CONCLUSION

We have presented an efficient remeshing method which operates directly on the model surface without resorting to any sort of parameterization. At the heart of our algorithm lies convenient segmentation of the mesh according to geodesic distances. This allows us to incorporate any sizing function to control the distribution of mesh triangles. Each segment is meshed independently, and the result delicately smoothed. The results seem to be superior on all the models we have tested.

Future work will extend this method to mixed quad and triangle meshing, as well as pure quad meshing.

### REFERENCES


Figure 7: Remeshes. Left: Input mesh. Middle: Remesh. Right: Zoom on remesh (except the Venus model which has a different contrast).
Figure 7 (cont.): Remeshes. Left column is the input mesh. Middle is the remesh and right is a zoom.
Figure 8: The effect of segmentation. The statistics of the results appear in Table 1. (a) Input tree mesh. (b) Remesh without segmentation. (c) Segments generated by our algorithm. (d) Remesh with segmentation.
TOWARD QUALITY SURFACE MESHING

Jean Cabello

EDS/PLM Solutions, 2000 Eastman Dr, Milford, OH, USA jean.cabello@eds.com

ABSTRACT

This paper presents recent progress and extensions to TriQuaMesh (TQM) [1], targeted at providing good quality surface meshes: Increased robustness of the 1D mesh generator to handle highly non linear size variations; interior node generation driven by a size variation interpolation domain; improved mesh distortion reduction between the parameter space and the physical space. The concepts of Size Control, Size Map and Triangle Map are introduced to increase the flexibility and the control on the final mesh. These concepts are general and apply to any meshing algorithm, although they will be illustrated with TQM.

Keywords: surface mesh generation, triangular, size map, curvature adaptation.

1. INTRODUCTION

The most common tetrahedral meshing algorithms, advancing front and Delaunay, require the surface mesh to be generated first, prior to filling in the interior with tetrahedral elements. For volume skin surfaces with geodesic distances between two points on the surface high compared to the Euclidian distance (i.e. narrow and high curvature passageway), adaptation of the surface mesh to the curvature might be critical to the success of an automatic volume mesher by preventing geometrical surface mesh intersection. The success and the quality of the volume mesher is then directly impacted by the success and quality of the surface mesher.

Although tremendous progress has been made with regard to meshing algorithms in both two and three dimensions, it still remains a difficult task to surface mesh any collection of surfaces with good quality and size control. Many approaches are available depending on the surface definition available (continuous or discrete).

For CAD parametric surfaces, a 2D parameter space representation of the surface is available and surface meshing is reduced to a 2D meshing problem. However, the surface can be poorly parameterized leading to high distortion when mapping the mesh back from 2D to 3D space. Some methods have been presented to account for the distortion between the 2D and 3D space using the CAD Riemannian surface evaluators [2].

Most CAD systems can export an STL or facetted representation of any parametric surface. This is a lower level definition of the surface that has the advantage of a simple and common format independent of the CAD system.

For discrete data representations of the surface (STL data or legacy data), some techniques work directly on the 3D discrete data to obtain a good quality mesh [3] while others use a divide and conquer approach to select a region and derive a parameter space to reduce the surface meshing problem to 2D. The two most common techniques used to derive a parameter space are: projection techniques, for example Maximum Area Plane (MAP) in I-DEAS, and flattening techniques based on angles [4] or based on lengths [5].

Adaptive meshing based on error estimation is another instance where controlling the mesh size variation to refine in areas of high error and coarsen in areas of low error is critical to obtain a good solution with reduced node and element count.

In this paper a simple method to account for the distortion between the 2D and 3D space for a surface represented by STL data is presented. Adaptive meshing based on discrete surface curvature is also presented in order to increase the mesh fidelity to the original surface at an economical cost compared to a constant size mesh.

1. TQM MESHING ALGORITHM IN A NUTSHELL.

The TQM algorithm is a divide and conquer meshing algorithm. Boundary loops are discretised using a 1D mesh generator. They are then joined into one single contour loop resulting in a loop of nodes. The contour loop is recursively subdivided into two sub contour loops along a “best split line” until the sub contour loop has been reduced to a trivial loop i.e. a loop with 3 points for a triangles or a loop with 4 points for quadrangles. All the details can be found in reference [1][6]. The remainder of this section recalls the two main points of interest for our discussion.

• Generation of contour points: Let’s assume that we have a curve Γ parametrised using the arc length with total arc length L. Let’s assume that we have a continuous grading function g(s) that represents the grading (or size) value along the parameter location. The 1D mesh generation problem can be stated as 1) how many points (nPoin) to generate? 2) Compute the parameter location for these points that will satisfy the grading function requirements. In [1] a solution was
proposed assuming a) Grading function is known discretely at \((nSample)\) sample or basis points b) the grading function is assumed to be piecewise harmonic (fig. 1a).

\[
g(s) = \frac{g_j + g_{j+1}}{2} + \frac{g_j - g_{j+1}}{2} \cdot \cos \left( \frac{s - s_j}{s_{j+1} - s_j} \pi \right)
\]

\[s \in [s_j, s_{j+1}], j = 0, \ldots, nSample-1; nSample \geq 2\]

In equation (3) the number of sample points, their parameter locations and their grading values is known. \(nPoin\) is computed by rounding the result of equation (3) to the nearest integer. The parameter locations of the 1D mesh points (fig.1b) are obtained by solving the system:

\[
\frac{s_i - s_{i-1}}{\left( g_i + g_{i-1} \right)} = \frac{S_{i-1} - S_i}{\left( g_{i+1} + g_i \right)} \quad \forall i = 1, nPoin - 1
\]

\[s_0 = 0; s_{nPoin} = L.
\]

derived from the equidistribution equation (2). More details on the solution of equation (4) are given in section 2.1.

Once the boundary loops have been discretised, boundary nodes are assigned grading values. The loops are joined into one single contour loop. A “best split line” criterion is used to join the loops and the 1D mesh generation technique is applied to determine how many points and their locations along the split line. The sample points are the two end points of the split line where the grading values are known. This process is applied recursively.

\[
nPoin - 1 = \frac{1}{C^{te}} \int_0^L g(s) \, ds = \frac{1}{C^{te}} \sum_{j=1}^{j=nSample} \frac{1}{2} \left( g_j + g_{j-1} \right)
\]

Given \(nSample\) points, their respective parameter locations and grading values \(\{s_j, g_j\}\), the number of points is derived by requiring to meet “at best” the equidistribution for each interval:

\[
\frac{s_i - s_{i-1}}{\left( g_i + g_{i-1} \right)} = C^{te} \quad \forall i = 1, nPoin
\]

Notice that in the equation above \(nPoin\) and \(s_i\) are unknowns as well as \(g_i = g(s_i)\). Summing equation (2) over all the intervals we obtain:

\[
F_p (Q) = W_{\text{angle}} \delta(\alpha) + W_{\text{Length}} \delta(l) + W_{nPoin} \delta(n)
\]

The weight factors \(W\) are constants. The first term, is minimal when the angles \(\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}\) (fig. 2) tend to multiples of 60 or 90 (respectively for triangles and quadrilaterals). The second term is minimal when the split line length tends to the minimum diameter of the boundary loop (i.e. the diameter of the smallest inscribed circle passing through two boundary points). The last term is minimal when the round off in equation (3) to obtain the integer \(nPoin\) is minimal. Other choices for the function are possible, see [6], [7].
2. TQM MESHING ALGORITHM SHORTCOMINGS

In this section we try to highlight some of the limitations of the TQM approach as it is currently implemented in I-DEAS.

2.1 1D Boundary discretisation

Given \( N\text{poin} \), their location is found by solving the nonlinear system of equations (4). One has to find the solution \( S = (s_1, s_2, \ldots, s_{n\text{poin}-1}) \) that satisfies the system of nonlinear equations:

\[
\begin{align*}
    f_i(S) &= 0; \quad \forall i = 1, n\text{poin} - 1 \\
    f_i(S) &= L_i \times s_{i-1} + D_i \times s_i + U_i \times s_{i+1} \\
    L_i &= g(s_{i+1}) + g(s_i); \quad U_i = g(s_i) + g(s_{i-1}) \\
    D_i &= -(g(s_{i+1}) + 2 \times g(s_i) + g(s_{i-1})
\end{align*}
\]

Using Newton-Raphson method for the non linear system of equations reduces to solve the tridiagonal system of equations:

\[
T \triangle S = \text{RHS}
\]

The index \( n \) represents the iteration number. The system is solved using LU decomposition with forward-backward substitution. The initial solution is taken as uniform distribution of the interior points. The convergence of the system depends on the matrix \( T \) condition number, which is not known. However, notice that if the system is linearised, setting \( g'(s) = 0 \), the matrix becomes well conditioned and is diagonally dominant. Based on this observation, a strategy is described in section 3.2 that improves the robustness of the 1D boundary mesh generation.

2.2 Boundary driven control only

The TQM algorithm first creates nodes along “best” split lines and then creates the elements. The node distribution along the split line is mainly driven by the grading value at the end points. The effect is that the boundary mesh is the main driver for the interior mesh, and undesirable boundary effects can propagate in the interior. The user has a good control on the element size variation on the boundary, but the control in the interior and the mesh transition is more difficult. For example, in I-DEAS, the user can input a local element length in the interior, but he cannot control its radius of influence. Also, when the surface exhibits curvature in its interior but its boundaries are flat there is no easy way to automatically refine the mesh with respect to the curvature. In most cases the user will have to manually add interior local element lengths in these areas to get the desired effect. To overcome these drawbacks and provide a mean to automatically refine the mesh in the interior of a surface, with no user interaction, TQM was extended to work with a background mesh, presented in section 3.2.

2.3 High distortion

TQM is a 2D mesh generator that generates a triangular or quadrangular mesh in a parameter domain. This parameter domain can be developed directly from a CAD parameter space or indirectly through projection or flattening techniques. In many cases, the mapping between the parameter domain and the physical domain is not isometric and elements size and quality need to be adjusted in the 2D domain to result in the desired mesh size and quality in the 3D domain. There are many approaches available to account for the local mesh distortion during the mapping. The most common approach [2] relies on the CAD query of continuous operators such as Curvature. These can turn to be expensive queries.

Currently in I-DEAS, to minimize the computational cost and account for length distortion between the 2D and 3D space, each split line is sampled with \( n\text{Sample} \) interior points. These points are mapped back in 3D space and we compute the variation \( ds \times ds \) where \( ds \) is the arc length variation of the curve poly-line in 3D space and \( ds \) is the corresponding variation of the split line in 2D space. This local scaling factor is then used to map the 3D mesh size to a corresponding 2D mesh size in the parameter domain. This is a very simple and robust approach, however one main drawback is that the scaling is unidirectional (along the split line). In section 3.3, a different approach to account for the local distortion is discussed.

3. TQM EXTENSIONS AND ENHANCEMENTS

The main usage of TQM in I-DEAS is for structural analysis with a constant mesh size. In this range the software performs fairly well. For boundary curvature adaptation the process is automated but might sometimes become unstable, resulting in poor node distribution transition. The interior surface curvature adaptation is not automated and has to be done through user input of interior local element sizes.
From now on, a stitched tessellation representing “accurately” the 3D surface or surfaces to be meshed (for example an STL representation from a CAD system) is assumed to exist. The corresponding 2D tessellation in the parameter domain, thus a discrete one to one mapping between 2D and 3D space, is also assumed to exist. In short, a triangle map, discuss in section 6, is available (see figures 7a, 7b).

They are many possible answers to the critical and common adaptive sampling questions: How many sample points? Where? What size? One answer could be to delegate the responsibility to the user. To get the desired sampling adaptation to the curvature both on the boundary and on the surface, it is proposed to leverage the surface STL representation. For the boundary curves, the facet points provide an adaptive sampling of the boundary curvature (see fig. 4).

### 3.1 1D boundary discretisation

For uniformly distributed and smoothly varying grading values, the non-linear tridiagonal system (7) exhibits a unique solution because the non-linear terms cancel out and the matrix is still well conditioned. However, when using an adaptive sampling point strategy with high grading values gradients, the system becomes ill-conditioned and might never converge to a solution. The solution algorithm has been enhanced by monitoring the convergence of the non linear system and when the solution oscillates and does not converge, after a fixed number of iterations, we restart the solution using the current solution but this time solving the linear system rather than the non linear one. This approach has proven to be very robust and is able to handle highly non-linear distributions and grading variations, even extreme cases with noisy input data. This strategy has the desired effect to smooth out the non-linearity due to high frequency input data.

### 3.2 Flexible background mesh approach

In order to provide better control over the interior element size variation, as is necessary in surface curvature adaptation, the TQM algorithm was enhanced to work in conjunction with a background mesh that provides an interpolation domain for the size variation. There are two type of background meshes used: 1) a background mesh resulting from the flattened faceted representation that is used as an interpolation domain for the 2D to 3D mapping function (see fig. 7d) and 2) a background mesh resulting from an initial sample mesh (see fig. 7e).

The split line node generation was modified. As was discussed in section 2.3, the split line is still sampled with 10 uniformly distributed sample points, however, the grading at the sample points is determined by interpolation. As the nodes are equally spaced along the straight line, the interpolation is quite fast since the result of the previous node triangle location is used to start a triangle walk to locate the next one. Also, special attention has been given to the robustness of the triangle walk algorithm in order to handle highly stretched, even flat, triangles that often occur in STL data.

### 3.3 Distortion correction

Our approach to account for the local distortion is to first create a sample mesh in the 2D domain, map it back to 3D space using the facet triangle map and compute the length distortion at the sample points. This gives a length scale factor $\lambda_i = \sum_{j S(i)} I_{2D}^{ij} / \sum_{j S(i)} I_{3D}^{ij}$ at each sample points, that multiplied to the 3D grading value represents the 2D grading value. $S(i)$ represent the ring of first neighbor nodes to node $i$. The length scale factor provides a local and isotropic estimate of the size distortion, is computationally inexpensive. One drawback is that there is no attempt to create stretched elements in the 2D space, only size varies. Given the 3D size variation, the scaling factor is applied. For example, a 3D constant size in 3D space will result in a varying element size in 2D space. The 2D mesher is then instantiated again with the 2D sample mesh as the background mesh with computed 2D sample grading values that drive the resulting final mesh. The 2D final mesh is mapped back to 3D space using the facet background grid.

### 3.4 Mesh transition

For constant size meshing, using the interpolation domain to determine the grading values along the split line can lead to sudden jumps in mesh size. One could smooth out the field of mesh size to get a smoother distribution. Instead, a parabolic distribution of the mesh size was simulated by keeping the two end points of the split line and adding a sample point half way with grading value equal to the global size. The grading at the sample points along the split line is then obtained as the minimum value from interpolation and the harmonic interpolation using the grading values at the end points and the midpoint along the line. This strategy proved to be valuable in cases where the split line connected two small features (i.e. end points have small grading values) so that the small feature size did not propagate along the split line.

### 4 SIZE CONTROL

There are various types of size control that a user may want, each one with different computational cost. Three types are defined, ranging from the lower cost to the higher cost: None, constant, curvature.

- **No size control:** The sample mesh is a coarse mesh formed by the boundary nodes and with no additional interior nodes. This approach is fast and can be used if the quality/distortion of the final mesh is not critical or if the space strategy used produces very little distortion (fig. 8b, 8c, 8d).
- **Constant size control:** The sample mesh is the initial mesh obtained without any account for the distortion. Distortion at sample points is computed leading to a 2D size interpolation domain that drives the 2D mesher. This is the preferred approach if the final mesh quality/size control is critical for a given constant size (fig. 8e, 8f, 8g).
- **Curvature size control:** The sample mesh is the same as in the constant size control case but this time the 3D
mesh size is computed as a function of the curvature (fig. 9b, 9c).

Although, we discuss a self-contained approach with the sample mesh internally generated, all the concepts are general and the sample mesh could be provided as input with the field of 3D sizes derived from an analysis, as is the case of adaptive meshing to a solution. With that data as input, the mesh generator will provide the desired mesh.

Smart size control [8] is another important variation on the size control that has not yet been implemented.

4.1 Curve Size Control

In order to adapt the boundary to the curvature we first need to compute the curve curvature. Two options are possible: line curvature or surface curvature.

4.1.1 Line curvature

The line curvature does not take into account the adjacent surface curvature. The curve has a poly-line representation formed by facet points (fig. 4). At each interior point, to the curve, the line curvature is computed as the inverse of the radius of the circle passing through 3 consecutive points. When these 3 points are collinear, the radius is set to infinity. At the curve end points, the curvature is computed by extrapolation. Furthermore, the minimum line curvature at end points is taken from all curves that share the vertex.

4.1.2 Surface curvature

In section 5 we will present discrete surface operators to evaluate curvature. The minimum radius of curvature is used at the boundary points to estimate the local surface curvature.

4.1.3 Line curvature versus surface curvature.

Either type of boundary curvature, line or surface, an average or a minimum can be chosen depending on the type of adaptation the user wants. The line curvature tends to highly refine small holes in flat areas. These can be very small geometry features compared to the mesh size that only need to be represented with a minimum of 3 to 4 points (fig. 6b). In all examples presented in this paper, only the surface curvature has been used.

4.1.4 Sampling refinement

For curvature size control, the facet point representation of boundaries near long flat regions close to fillets (fig. 5) need special attention. In a sense, these boundaries are “under sampled” and sample points need to be added to properly capture the flatness of the curve. The algorithm works as follows:

- Given a curve $C$, the arc length parameter location $t_j$ of its sample points $P_j$ and a global mesh size $S_g$
- Loop over segments $[t_j, t_{j+1}]$
  - Compute it length $L_j$
  - Segment grading average
    $$ g_j = \frac{1}{2} (g_j + g_{j+1}) $$
  - If $\lambda_g g_j S_g < \lambda_i L_j$
    - Add extra sample point at the mid point.
    - Assign grading value equal to $S_g$ at this extra sample point.

The parameter $\lambda_g$ is a constant representing the grading ratio between the local grading and the global size, while the parameter $\lambda_i$ represents the inverse of the minimum number of intervals desired. The first part of the inequality states that the grading at the segment end points is very small compared to the global size. The second part of the inequality states that the segment length is large compared to the global size. By adding a point halfway, a parabolic node distribution will result. In the examples, values of $\lambda_g = 4.0$ and $\lambda_i = 1/3$ have been chosen.

5 SURFACE SIZE CONTROL

For a constant size control the 3D mesh size is a field of constant values. For a curvature size control the mesh size becomes function of the local surface curvature evaluated at the sample points.

5.1 Continuous surface operators

Given a surface $S$, the two principal curvatures $K_1$ and $K_2$ of the surface along the two orthogonal principal direction vectors $(\vec{e}_1, \vec{e}_2)$ are the extrema values of all the normal curvatures. The normal curvature $K_N(\alpha)$ to the
surface $S$ at a point $P$ with unit normal $\vec{N}$ along a unit tangent vector $\vec{t}_\alpha$ is defined as the line curvature of the curve formed by intersecting the plane $(P, \vec{N}, \vec{t}_\alpha)$ with the surface $S$. The mean curvature $K_H$ is defined as the average of the normal curvature:

$$K_H = \frac{1}{2\pi} \int_0^{2\pi} K_N(\alpha) d\alpha$$  

(8)

The Gaussian curvature $K_G$ is defined as the product of the two principal curvatures:

$$K_G = K_1.K_2$$  

(9)

and the mean curvature is expressed as the average of the two principal curvatures:

$$K_H = \frac{1}{2}(K_1 + K_2)$$  

(10)

### 5.2 Discrete surface operators

Given a triangle map, the first and second order attributes of the surface (normal vector, mean curvature $K_H$, Gaussian curvature $K_G$) can be approximated [9], [10].

The discrete normal curvature is derived from the formula:

$$\theta(P_i) = \sum_{k \in S(i)} \alpha_{kj}^{3D}$$  

(12)

with $\theta(P_i)$ representing the total vertex angle.

The radius of curvature at a point $P_i$ along the direction of the edge $P_iP_j$ is:

$$\rho_j = \frac{1}{2} \frac{||P_iP_j||^2}{<\vec{N}_i, P_iP_j>}$$  

(13)

and the minimum radius of curvature at point $P_i$:

$$\rho_i = \min_j \rho_j$$

At each point $P$ one can compute the vertex angle excess $2\pi - \theta(P)$ that also represents the (total) Gaussian curvature at an interior point:

$$\int \int K_G dA = 2\pi - \theta(P)$$  

(14)

The discrete Gaussian curvature at point $P$ can be approximated by:

$$K_G(P) = (2\pi - \theta(P))/\text{Vor}(P)$$  

(15)

with $\text{Vor}(P)$ computed as the voronoi area at a point if all triangles are acute and for obtuse triangles the containment circle is used instead of the circumscribed circle criteria (i.e. instead of joining adjacent edges midpoints to the center of the circumscribed circle they are joined to the midpoint of the (opposite) longest edge (fig. 5).

The discrete normal curvature is derived from the formula:

$$\int \int 2K_H(P) . \vec{N}(P) dA = \frac{1}{2} \sum_{j \in S(i)} (\cot{\alpha_{ij}^{3D}} + \cot{\beta_{ij}^{3D}})(\vec{P}_i - \vec{P}_j)$$  

(16)

with $\alpha_{ij}$ and $\beta_{ij}$ the two angles opposite to the edge $(P_i, P_j)$ in the two triangles sharing the edge (see fig. 5).

The discrete mean curvature normal is given by:

$$\vec{K}_H(P_i) = \frac{1}{2 \times \text{Vor}(P_i)} \sum_{j \in S(i)} (\cot{\alpha_{ij}^{3D}} + \cot{\beta_{ij}^{3D}})(\vec{P}_i - \vec{P}_j)$$  

(17)

and the approximation of the normal curvature is obtained as:

$$K_H(P_i) = \frac{1}{2} || \vec{K}_H(P_i) ||$$  

(18)

All the above formulas are the discrete counterparts of the continuous first and second order attributes of the surface.

In [11] a measure of the deformation of a triangle between 2D and 3D space is proposed:

First we compute the discrete normal as an angle weighted average of the normals to the facets surrounding the point.

$$\vec{N}_i = \frac{\sum_{k \in S(i)} \alpha_{kj}^{3D} \vec{N}_k}{\theta(P_i)}$$  

(11)
\[
\text{Def} \left( T_i^{3D} \rightarrow t_i^{2D} \right) = \\
\frac{\cot \alpha_{3D} \| t_i^{2D} \| ^2 + \cot \alpha_{4D} \| t_i^{2D} \| ^2 + \cot \alpha_{2D} \| I_i^{2D} \| ^2}{2 \times \text{Area}(t_i^{2D})}
\]

By summing over all the triangles, this formula provides a measure of the total distortion induced by the mapping used in the triangle map between the 2D and 3D space. The global measure could be used as criteria to select the space development strategy with least distortion and/or to improve an initial parameter domain by minimization of the global distortion measure. Currently the curvature adaptation strategy only considers the minimum radius of curvature, but experimentation with other criteria is underway.

The mapping between the discrete curvature and the 3D size is as follows [10]:

- Given $\varepsilon$: a percent deviation to the original geometry.
- Given a global size $S_{global}$
- Compute the constant $\gamma = \sqrt{\varepsilon(1-\varepsilon)}$
- Look up in the triangle map for the minimum radius of curvature, $\rho_i$.
- Compute local 3D size: $S_i^{3D} = \rho_i \times \gamma$
- Bound $S_i^{3D}$, $\frac{1}{d\text{LenRatio}} \times S_{global} < S_i^{3D} < S_{global}$

A value of $d\text{LenRatio} = 10$ was chosen to control the minimum size allowed during curvature adaptation.

6 TRIANGLE MAP

The triangle map keeps a map between the 3D mesh and the 2D mesh and also provides a wealth of information about the surface. There are two triangle maps that we use. The STL triangle map (fig. 7a, 7b) and the sample mesh triangle map. One starts with a given STL of a surface to mesh (fig. 7a). The node coordinates in 3D space and the mesh connectivity are stored in the triangle map. Using a space development strategy (in all the examples presented a flattening strategy is used), the 2D parameter domain (fig. 7e) is created and the map $x(u,v), y(u,v), z(u,v)$ is stored for the facet points. The 2D parameter domain is an interpolation domain for the mapping between the 2D space and the 3D space. Next, all the discrete operators are computed as well as the distortion between the 2D space and the 3D space. The STL triangle map is always used to map the nodes back to 3D space. Another use of the STL triangle map is during the boundary node generation with curvature size control.

As mentioned in section 4.1, we use the surface curvature that we obtain directly in the look up table of the triangle map. The boundary nodes were generated in 3D space and they need to be mapped to their corresponding 2D space value. To do so one could perform an exhaustive 3D point in triangle location. Instead, as the curve is represented by a poly-line of facet points, we store the parameter location of the facet points along the curve. For a mesh point generated along the boundary curve at the parameter location $t$, we find the facet point interval $[t_i, t_{i+1}]$ that contains $t$ and use a linear interpolation to find the corresponding 2D parameter location $(u,v)$. The 3D boundary node loop is then mapped to the 2D plane. The grading value at the boundary nodes is computed as an average of the two adjacent edges length at the points. This gives us the “real” 2D size that already accounts for distortion.

The sample mesh is generated in 2D space using the TQM meshing algorithm with the desired mesh size. At this stage, the size map has not been created yet. The grading values along the split line are computed using the piecewise harmonic interpolation (equation 1). The sample mesh is mainly uniform (unless we are using no size control) and provides a sampling field of interest for the given mesh size. The 2D mesh is transformed back to 3D space using the STL triangle map, and a sample mesh triangle map is created. This latter triangle map provides a look up table to compute for each point in the sample mesh, its distortion, its surface curvature etc... The data (distortion, curvature etc...) is computed at once for the whole mesh and stored in the triangle map. The boundary discretisation of the sample mesh and the final mesh are identical and need not be regenerated. The size map is created and will be used as a size interpolation field for the final mesh.

7 SIZE MAP

The size map is a combination of the size control and the triangle map. The size control provides the 3D size variation on the surface for the sample mesh while the triangle map provides the size distortion. The size map combines both data into one single value. For example, for a constant 3D size mesh, the size control has a field of constant values and only the distortion factor varies at each point of the sample mesh. The 2D scaled mesh size is the product.

8 EXAMPLES
8.1 Smooth transition from small holes to large constant size

Figure 6a represents the final mesh for a square with a small hole. The diameter of the hole is 1 while the square size is 100. A mesh size of 12.5 has been used. The hole has been represented by 5 elements and the mesh transitions smoothly from the small to the large size (fig. 6b).

8.2 Developed space distortion comparison.

Figures 7 (b,c,d,e) illustrates the advantage of the flattening technique [5] over the Maximum Area Plane (M.A.P.) projection used in I-DEAS. Figure 7a represents the STL of half a sphere. Figure 7b represents the 2D parameter space resulting from the projection technique. Notice the high distortion along the boundary where triangles have been “squashed”. Figure 7c represents on top the 2D final mesh and on the bottom the corresponding 3D mesh obtained with the option of constant size control. Highly distorted elements are generated along the boundary. Clearly, a projection technique is not satisfactory in local areas where the normal to the surface is orthogonal to the direction of projection and a small change $d\varepsilon$ in the 2D space tends to infinity in the 3D space.

Figure 7d represents the flattened STL mesh. This time the lengths have been preserved along the boundary and the highest distortion seems to occur around the pole. The final mesh with constant size control is presented in figure 7e. The flattening space strategy produced a more isometric mapping leading to the good mesh quality.

Projections techniques are computationally inexpensive but they are restricted to domains that can be projected and therefore work well with low curvature domains. On the other hand, flattening techniques, depending on the type of domain at hand, result in more isometric mapping (works well for developable domain independent of the curvature) but are in general more computationally expensive. They also have their own limitations (cannot flatten a closed surface without cutting it), but they are less stringent than projections techniques.

8.3 Size Control comparisons.

This example illustrates the results obtained with various types of size control. The mesh size is 5 in all cases. Figure 8.a represents the initial STL. In figure 8b and 8e the sample meshes for no size control and constant size control options are presented. Figures 8c and 8f provide a comparison of the resulting 2D final meshes for respectively no size control and constant size control. The mesh in fig. 8f has a smoother variation of the the size than the one in fig. 8c, due to a richer sampling of the curvature variation and therefore a richer interpolation domain for the distortion scaling factor.
8.4 Curvature adaptation.

Figure 9a represents a bracket with fairly complex curvature patterns. Figure 9b demonstrates how the 2D mesher is able to accurately adapt to the curvature pattern and figure 9c shows that the refinement, when mapped back in 3D space did occur in the correct locations. Notice also, the 1D boundary curvature adaptation and how the holes in flat regions were not refined as the surface curvature, not the line curvature, was used in these examples.

Figure 10a, 10b and 10c is another example of curvature adaptation. Notice in figure 10b that the 2D mesher accurately captured the high curvature areas and started to pick up the lower curvature of the rear flaps both on the boundary and the interior.

9 CONCLUSION

Recent progress and extensions to increase TQM flexibility to handle large variations in mesh size all across a surface have been presented and demonstrated.

An approach that uses the surface STL data as sample points for the boundary discretisation and automatically generates a sample mesh for the interior has also been presented. A natural way of getting the sample mesh, based on the final mesh global size, proved to be a good sampling strategy. The price to pay for the additional quality is the cost of meshing the surface twice. It is the user’s choice whether to incur this extra cost.

The curvature adaptation presented is robust and transitions smoothly between high and low regions of curvature.

Finally, we have tried to isolate independent concepts such as size control, triangle map and size map that put together provide tremendous flexibility.

This work is still at a preliminary stage with emphasis on flexibility and surface mesh quality. Future work should include smoothing techniques that are “adaptation preserving”, a study of the viability of using the STL as sample mesh, smart sizing, and study and development of “best practices/strategies” to get a good quality surface mesh at lowest cost.

10 ACKNOWLEDGMENTS

The author expresses his sincere thanks to his colleagues at EDS/PLM solutions for their active support. In particular, Michael Hancock, Nilanjan Mukherjee, Hui Xiao, Radhika Vurputoor and Kirk Beatty for their active involvement.

11 REFERENCES.


Figure 7: Developed space distortion comparison between (left) Maximum Area Plane and (right) flattening.
Figure 8 – Size control comparison between (left) size control none and (right) size control constant.
Figure 9b – 2D final mesh, curvature size control

Figure 9c – 3D final mesh, curvature size control

Figure 10b – 2D final mesh

Figure 10c – 3D final mesh

Figure 9 – Curvature adaptation of a bracket with complex curvature patterns

Figure 10 – Curvature adaptation of a bracket around high and low areas of curvature.
We present a method for isotropic remeshing of arbitrary genus surfaces. The method is based on a mesh adaptation process, namely, a sequence of local modifications performed on a copy of the original mesh, while referring to the original mesh geometry. The algorithm has three stages. In the first stage the required number or vertices are generated by iterative simplification or refinement. The second stage performs an initial vertex partition using an area-based relaxation method. The third stage achieves precise isotropic vertex sampling prescribed by a given density function on the mesh. We use a modification of Lloyd’s relaxation method to construct a weighted centroidal Voronoi tessellation of the mesh. We apply these iterations locally on small patches of the mesh that are parameterized into the 2D plane. This allows us to handle arbitrary complex meshes with any genus and any number of boundaries. The efficiency and the accuracy of the remeshing process is achieved using a patch-wise parameterization technique.

**Keywords:** surface mesh generation, isotropic triangle meshing, centroidal Voronoi tessellation, local parameterization

### 1. INTRODUCTION

Mesh generation has received a great deal of attention by researchers in various areas ranging from computer graphics through numerical analysis to computational geometry. Quality mesh generation amounts to finding a partition of a domain by elements—typically, triangles or quads. The shape, angles or size of these elements must match certain criteria (see [4, 5]). In most cases the boundary of the domain is given, as well as an importance map that must be discretized. The problem of surface remeshing, being of particular interest for reverse engineering, is different in the sense that the input domain is itself discrete. The mesh is often highly irregular and non-uniform, since it generally comes as the output of a surface reconstruction algorithm applied to a point cloud obtained from a scanning device.

Isotropic sampling leads to well-shaped triangles, and thus high-quality meshes when the notion of quality is related to the shape of the triangles. Such meshes are important for simulations where the quality of the mesh elements is critical. For digital geometry processing [35], most scanned models must undergo complete remeshing before processing. Many geometry processing algorithms (e.g., smoothing, compression) benefit from isotropic remeshing, combined with uniform or curvature-adapted sampling.

### 1.1 Related Work

Parameterization-based remeshing techniques [2, 3, 15] have benefited from recent renewed interest in efficient parameterization methods for surface meshes [7, 19, 23, 29]. Here, the key is to parameterize the original mesh to obtain a bijective mapping and minimize the distortion due to the flattening process. The sampling and meshing stages are then considerably simpler on the (planar) parameter space. This allows both undersampling and oversampling with a high level of control by the user. Despite their recent popularity, these remeshing techniques (so-called “global approaches”) have many drawbacks:

- **surface cutting:** each patch should be homeomorphic to a disk, therefore, closed or genus $> 0$ models have to be either cut along a cut graph to extract the polygonal schema [20], or decomposed into an atlas [23]. Finding a “smart” cut graph is not only known to be a delicate procedure [10, 15, 30], but also introduces a set of artificial boundary curves, associated pairwise. These boundaries, sampled as a set of curves (i.e., 1-manifolds, while the surface has to be sampled as a 2-manifold), generate a visually displeasing seam tree. Some authors propose to apply a local mesh adaptation
process to hide the seam a posteriori [2] but this solution is not fully satisfactory. Another solution to reducing the influence of the seam [19] consists of computing a globally smooth parameterization by decomposing the surface into patches and minimizing the distortion simultaneously across all patches. Although elegant, the latter solution does not remove the need for handling the patch boundaries during a global sample partitioning process.

- **parameterization and overlapping:** instead of constraining the user to fix the boundary onto a predetermined convex polygon, two recent methods minimize the distortion due to the parameterization by letting the solver find the “best” boundary while solving a linear system [7, 23]. Even though the gain in term of distortion is obvious, this approach does not solve the overlapping issues, contrary to other methods that may introduce additional seams [30] or generate an atlas [31].

- **numerical issues:** despite recent efforts for efficient computation of global parameterizations [23], the latter remains a time-consuming process for large models. Moreover, models with bad isoperimetric properties (e.g., sock-like shapes) are numerically intractable for most state-of-the-art techniques.

- **lack of guarantees:** the conformal parameterization [7, 9, 23, 27] has often been the method of choice for irregular surface remeshing, isotropic [2, 3] or anisotropic [1]. Unfortunately, there exist triangulations for which this parameterization is not valid (see [18]), even when the boundary is fixed to be convex. Although the triangulation can be enriched by vertex insertion to produce a valid embedding, it is still unclear how many additional vertices are needed and what the guarantees are when using a scheme with a free, possibly concave, boundary.

The main alternative to global parameterization is known as the mesh adaptation process. It consists of performing a series of local modifications directly on the mesh, in embedding space. Remeshing algorithms using this approach [12, 13, 16, 17, 28, 36] usually involve computationally expensive optimizations in 3D. To improve efficiency, Frey and Borouchaki [13] use a less accurate optimization in the tangent plane. In a subsequent work, Frey [12] uses a paraboloid to obtain better approximation. The main issue of this approach is the fact that the mesh vertices must remain on the original mesh during the adaptation process. Otherwise, fidelity is quickly lost because of error accumulation. To solve this problem, the new optimal vertex positions are projected back to the original surface. Projecting the vertex involves a complex, computationally expensive and inaccurate computation that may even lead to topological errors during the remeshing process.

### 2. MAIN CONTRIBUTIONS AND OVERVIEW

In light of the drawbacks listed in the previous section, our main contribution is to combine the mesh adaptation process with a set of local, overlapping parameterizations. This allows us to handle large meshes of arbitrary genus. Another motivation of this paper is to formulate the issue of isotropic surface sampling using the concept of centroidal Voronoi tessellation. This way we shift from the so-called unit length paradigm used for numerical analysis [14] to the unit cell tiling paradigm, well suited for our application, i.e., sampling of 2-manifolds. The first technique aims at generating meshes with unit edge length measured in a control space metric, while our algorithm aims to partition the surface with unit density integrated over the cells of a centroidal Voronoi tessellation. In particular, we show how the latter property is directly related to the notion of isotropic surface sampling.

Our technique uses two meshes: one is the piecewise smooth geometric reference, which we call the geometric mesh $M_O$ (see Section 3). The second mesh $M$ is initialized with a copy of the original mesh and evolves during the remeshing process until the desired mesh properties are achieved. Our technique falls into the category of local adaptation methods since remeshing is performed by a series of well-known local modifications: edge-flip, edge-collapse, edge-split and vertex relocation. The modifications are always applied sequentially to achieve desirable mesh characteristics.

The technique has three main stages: complexity adjustment, vertex partitioning and precise vertex placement. The first stage achieves the required number of vertices by applying iterative mesh simplification or refinement on the evolving mesh (see Section 4). The second stage uses a novel area-based remeshing technique to approximately partition the vertices in accordance with a density function specified on the original mesh (see Section 4). The second stage performs a precise isotropic placement of the vertices by constructing a weighted centroidal Voronoi tessellation (see Section 5). Section 7 shows some experimental results and Section 8 concludes.

### 3. GEOMETRIC BACKGROUND

The input to our remeshing scheme is a 2-manifold triangle mesh $M_O$ of arbitrary genus, possibly with boundaries. We consider $M_O$ to be a piecewise linear approximation of a smooth surface, which is $C^1$-continuous except at boundaries and a set of curves specified by feature edges. These feature edges can be provided by the user or computed automatically by feature detection techniques [40].

Surface reconstruction requires normal information at the mesh vertices. If the normals at the mesh vertices are not given, we use a method similar to [26, 28] to generate them: Every vertex is assigned a normal which is the weighted average of the normals of the faces adjacent to it. The weights
are proportional to the angles of the corresponding faces at
the vertex and sum to unity. Normals of a vertex lying on
feature edges are not the same within all its adjacent faces.
They are also defined by the weighted average of the face
normals but as if the mesh was cut along the feature edges at
the vertex.

3.1 Surface Approximation
Similarly to [34], we perform an estimate of the smooth sur-
facing in the vicinity of a mesh triangle. This may be ob-
tained by reconstructing an approximation of the surface us-
using triangular cubic Bézier patches for every face of
\( \mathcal{M}_0 \). Vlachos et al. [37] presented a simple and efficient,
yet robust and accurate, method to construct such curved patches
called PN triangles. The triangle vertex normals together
with vertex coordinates are used to construct a PN trian-
gle. PN triangles usually (but not always) maintain a \( \mathbf{G}^1 \)-
continuous surface along adjacent triangles when their com-
mon vertices have identical normals. The normal of any
point within a PN triangle is defined as a quadratic interpola-
tion of the normals at the triangle vertices. Although Walton
and Meek [39] presented a more complex and computa-
tionally expensive method to create triangular patches that guar-
antees \( \mathbf{G}^1 \)-continuity on the patch boundaries, we use PN
triangles as a good tradeoff between accuracy and efficiency.
Given a point \( q \) inside a triangular face \( f = (q_1, q_2, q_3) \), the cor-
responding point on the surface of the PN triangle of \( f \), as
well as the normal at this point, can be uniquely defined
by the barycentric coordinates of \( q \) with respect to \( f \).

3.2 Controlling Fidelity
Our remeshing scheme performs a series of local mesh
modifications. To ensure fidelity of the new mesh to the
geometry of the original mesh, two measures are used to
evaluate the distance between the two meshes. These mea-
sures are evaluated for every local modification on the region
of the mesh affected by the modification. The modification
is applied only if it does not violate the error conditions
defined by the measures. The measures we use are con-
ceptually similar to those of Frey and Borouchaki [13]
are defined for a face instead of an edge. These measures
were formulated in [34]. We briefly describe the measures and
discuss their advantages.

Let \( f = (v_1, v_2, v_3) \) be a face whose error is to be estimated.
The first measure \( E_{\text{smth}} \) captures the degree of smoothness
and should not exceed some threshold angle \( \theta_{\text{smth}} \):

\[
E_{\text{smth}}(f) = \text{max}_{i \in \{1, 2, 3\}} \langle N_f, N_{v_i} \rangle < \cos \theta_{\text{smth}}. \tag{1}
\]

where \( N_f \) and \( N_{v_i} \) are unit normals of \( f \) and its vertex \( v_i \),
respectively; \( \langle \cdot, \cdot \rangle \) denotes the dot product. \( N_v \) is taken from
the original surface. Intuitively, \( E_{\text{smth}} \) describes how well \( f \)
coincides with tangent planes of the surface at the vertices of
\( f \). The second measure \( E_{\text{dist}} \) captures the distance between
\( f \) and the surface:

\[
E_{\text{dist}}(f) = \text{max}_{i \in \{1, 2, 3\}} \langle N_{v_i}, N_{v_{i+1}} \rangle < \cos \theta_{\text{dist}}. \tag{2}
\]

Vertex indices are modulo 3; \( \theta_{\text{dist}} \) is a threshold angle.
A larger value of the maximal angle between the normals
of two face vertices corresponds to a more curved surface
above \( f \), and thus, to a greater distance. The beauty
of these two measures is that they involve only normal
directions. In addition to their computational efficiency,
when used together, these two measures are also robust and
accurate.

4. INITIAL VERTEX PARTITION
To achieve the target mesh complexity, we apply local
refinement or simplification operations. We perform a series
of edge-collapse or vertex-split operations until the required
number of vertices is achieved. Edges whose faces have
minimal/maximal error metrics are simplified/refined first.

The heart of our remeshing scheme is the construction of the
weighted centroidal Voronoi tessellation on the 3D mesh
to achieve precise vertex placement (see Section 5). However,
being optimal both in terms of sampling and isotropy,
generating the weighted centroidal Voronoi tessellation is an
extremely slow iterative process. This process first brings
the mesh to the required sampling prescribed by a density
function, then the mesh isotropy is optimized. It turns out
that the first stage of the process is even slower than the
second one, in contrast to many other iterative processes.
The reason is that the process inherently maintains the local
isotropy during resampling. To accelerate this process we
first generate a coarse, initial sample partition by using a
novel efficient area-based relaxation technique.

Alliez et al. [2] introduced an algorithm based on error
diffusion that efficiently finds a good initial sampling.
Unfortunately, this algorithm cannot guarantee fidelity of
the resulting mesh to the original. Features that are not
specified explicitly may be easily lost by this algorithm.
In order to guarantee the mesh fidelity of the initial sampling
we use an “area-based remeshing” technique, which is based
on a series of local mesh modifications, while validating the
mesh fidelity by the error measure described in Section 3.2.

The area-based remeshing technique was first introduced by
Surazhsky and Gotsman [33, 34]. It is based on the idea of
locally equalizing the area of triangles or bringing the areas
to the ratios specified by the density function. After this,
it remains to achieve a precise isotropic vertex placement.

5. PRECISE VERTEX PLACEMENT
Our goal is to isotropically sample a density function speci-
fied on the original surface mesh \( \mathcal{M}_0 \). There are, thus, two
terms (sampling and isotropy) to be defined:
- Sampling: to partition a density function among a set
  of samples. The density function is defined over a
  bounded domain, which must be partitioned so that
  we obtain a tiling, or tessellation, where each tile cor-
  responds to exactly one sample, without overlapping
or holes. The density partition must be done so that we obtain the so-called equal-mass enclosing property, namely, each tile contains the same amount of density.

- Isotropy: the shape of each tile is not biased with respect to any particular direction. In other words, each cell is as compact (i.e., as “round”) as possible. In the uniform case the ideal tile is a disk, which maximizes the compactness, but does not produce a tiling of the domain. The hexagonal lattice better conforms with uniform tiling along with optimal compactness. The non-uniform case leads to a tradeoff between compactness and partition of the density function.

5.1 Centroidal Voronoi Tessellation

The initial triangulation gives us a vertex partition, which defines a tiling of a 2D parameter space. Each triangular tile corresponds to three samples (the vertices of the triangle) instead of one as desired. We, therefore, use the dual of the triangulation, i.e., the tessellation in which each tile is now associated with exactly one sample. We aim at obtaining a special class of Voronoi tessellations, the so-called centroidal Voronoi tessellation, with the two properties mentioned above, i.e., equal-mass enclosing and isotropy.

Given a density function defined over a bounded domain \( \Omega \), a weighted centroidal Voronoi tessellation \[8\] (denoted WCVT) of \( \Omega \) is a class of Voronoi tessellations, where each site coincides with the centroid (i.e., center of mass) of its Voronoi region. The centroid \( c_i \) of a Voronoi region \( V_i \) is calculated as:

\[
c_i = \frac{\int_{V_i} x \rho(x) dx}{\int_{V_i} \rho(x) dx}
\]

where \( \rho(x) \) is the density function. This structure turns out to have a surprisingly broad range of applications for numerical analysis, location optimization, optimal partition of resources, cell growth, vector quantization, etc. (see \[8\]). This follows from the mathematical importance of its relationship with the energy function

\[
E(z, V) = \sum_{i=1}^{n} \int_{V_i} \rho(x)|x - z_i|^2 dx
\]

where \( V \in \Omega \) and \( z \in V \). It is proven in \[6\] that (i) the energy function is minimized at the mass centroid of a given region, and (ii) for a given set of centers \( Z = \{ z_i \} \), the energy function \( E(Z, V) \) is minimized when \( V \) is a Voronoi tessellation.

5.2 Building a WCVT on a 3D Mesh

One way to build a weighted centroidal Voronoi tessellation is to use Lloyd’s relaxation method. The Lloyd algorithm is a deterministic, fixed point iteration \[25\]. Given a density function and an initial set of \( n \) sites, it consists of the following three steps:

1. Construct the Voronoi tessellation corresponding to the \( n \) sites;

2. Compute the centroids of the \( n \) Voronoi regions with respect to the density function expressed in local parameter space, and move the \( n \) sites to their respective centroids;

3. Repeat steps 1 and 2 until satisfactory convergence is achieved.

Since a Delaunay triangulation and its corresponding Voronoi tessellation are dual, we do not need to work explicitly with a Voronoi tessellation but rather with its dual triangulation. We adapt the Lloyd algorithm in the following manner. Instead of constructing the Voronoi tessellation for the point set of the current mesh, we modify the mesh by a series of Delaunay edge flips in order to maintain the local Delaunay property of the mesh. For every vertex, we then compute its Voronoi cell in a local parametric domain, and move the vertex to the new 3D location corresponding to the centroid of the cell. We now describe these steps in detail.

Updating the local Delaunay property Notice that the usual definition of the Voronoi tessellation holds for a set of sites in Euclidean space, i.e., in the 2D plane for partitioning a 2-manifold. As demonstrated in \[21\], Voronoi diagrams can also be constructed in Riemannian manifolds for sufficiently dense sets of points. In our algorithm, the current 3D triangulation is the result of a series of local mesh adaptations performed for initial vertex partition. Each local mesh adaptation has been performed while maintaining a “local” 2D Delaunay property. Instead of building a new Voronoi tessellation at each step of the Lloyd relaxation process, we restore the local Delaunay property by performing a series of edge flips in 3D. This maximizes the smallest angle property. This task is performed efficiently by updating a priority queue sorted by the angles.

Computing the centroid Every relaxation step in the sequence of Lloyd iterations moves a vertex \( v \) from the newly generated mesh to the centroid of its “Voronoi” cell (we abuse the word Voronoi here, since the cell is not planar or even convex). To proceed we first need to define a planar Voronoi cell for \( v \). Denote the vertices incident to \( v \) as \( v_1, \ldots, v_k \), where \( k \) is the degree of \( v \). Let \( S(v) \) be a sub-mesh of \( M \) containing only \( v, v_1, \ldots, v_k \) and faces incident on \( v \). We reduce the problem in 2D by mapping \( S(v) \) onto the plane using a natural and simple method approximating the geodesic polar map \[32\], described by Welch and Witkin \[41\] and later by Floater \[11\]. The method preserves the lengths of edges incident to \( v \), and the relative angles of \( S(v) \) at \( v \). This method is an efficient and precise approximation of a conformal mapping of \( S(v) \) onto the plane. Let \( p, p_1, \ldots, p_k \) be the positions of vertices \( v, v_1, \ldots, v_k \) within the resulting mapping \( S_P(v) \). \( p \) is mapped to the original. Then we construct a Voronoi cell of \( v \) in \( S_P(v) \) with respect to the circumcenters of the triangles built from \( p \) and \( p_1, \ldots, p_k \), and compute the centroid \( p_{new} \) of this cell with respect to an approximation of the density function specified over the original mesh. The latter approximation consists of
evaluating the density function at the new mesh vertices and piecewise linearly interpolating the resulting density over the new mesh triangles.

5.3 Vertex Relocation

Knowing the new vertex position of \( v \) \( (p_{\text{new}}) \), we need to bring it back to the original surface of the given mesh, namely, to find the position of \( v \) denoted by \( x_{\text{new}}(v) \) on \( \mathcal{M}_O \) that corresponds to \( p_{\text{new}} \). Existing remeshing methods, e.g., \([13, 17, 28]\) solve this problem by computing the vertex projection onto the original surface. As stated in Section 1.1, projecting the vertex involves an expensive and possibly inaccurate computation that may even lead to topological errors. We solve this problem using a mesh parameterization with low distortion and guarantee of bijective mapping. This way we can deduce \( x_{\text{new}}(v) \) precisely and efficiently.

We now briefly describe how to find \( x_{\text{new}}(v) \) using mesh parameterization. For every vertex of \( \mathcal{M} \) we maintain its exact position on the original surface using barycentric coordinates of the vertex within a specific face of \( \mathcal{M}_O \). Note that this gives us a unique point on the reconstructed surface defined by PN triangles over \( \mathcal{M}_O \); see Section 3.1. The central idea in using parameterization to locate \( x_{\text{new}}(v) \) is to use barycentric coordinates of \( p_{\text{new}} \) with respect to a face of \( S(v) \) that contains it. Using these barycentric coordinates together with the barycentric coordinates of the face vertices, we locate a point in the parametric domain of \( \mathcal{M}_O \). This point is then elevated to the original surface.

However, this simple scheme is only applicable when we have a well-defined parametric domain embedded in the 2D plane. Since not all 3D meshes are isomorphic to a disk, such a 2D parametric domain may not exist. To solve this problem we use a novel dynamic patch-wise parameterization technique introduced independently by Vorsatz \textit{et al.} \([38]\) and Surazhsky and Gotsman \([34]\). This technique aims to overcome the problems of global parameterization (see Section 1.1) and allows the handling of meshes of arbitrary genus and boundaries. It maintains a set of small (usually manifold) overlapping patches and their corresponding conformal parameterization. Every patch is constructed on demand depending on a specific local modification and contains the region required to locate a new vertex position in the 2D parametric domain. Reuse of the patches already parameterized guarantees the efficiency of this technique both in terms of computational cost and memory consumption. See Figure 2 which demonstrates how this technique is used for vertex relocation.

6. PRESERVING FEATURES

Note that near feature creases and boundaries, the computation of the centroid must be more sophisticated. To proceed we \textit{clip} the Voronoi cells with the set of feature edges \([24]\). This allows us to disconnect two smooth regions separated by a feature crease during the computation of the centroid. It leads to a nice sampling quality in the vicinity of the features, obtained through the non-symmetric behavior of the algorithm (the feature edges influence the surface samples but the surface samples do not influence the samples on a
feature edge). At the intuitive level, two samples adjacent in the Voronoi tessellation and separated by a feature do not influence each other, and the samples close to a feature edge are repulsed by the constraints (see Figure 3). Geometrically, clipping a cell by the set of constraints removes some regions from the computation of the centroid, making the Lloyd relaxation consistent with respect to the features. Once the centroid has been computed, it remains to relocate the vertex v to the centroid.

7. EXPERIMENTAL RESULTS

The algorithm described in this paper has been implemented in an interactive software package. Similarly to [34], the user can control the remeshing via the definition of the density function, either uniform, or adapted to the curvature. We also provide an option to smooth the density function and therefore obtain a smoother mesh gradation.

We have run our technique on a variety of models of arbitrary genus and complexity. Figure 4 illustrates a curvature-adapted remeshing of the rocker-arm model with 10,000 vertices (the same as the original model). The tessellation shown in the middle is drawn by tracing an edge between the circumcenters of two incident triangles, every circumcenter located in the support plane of the corresponding triangle. The genus 1 feline model is remeshed both uniformly and with a curvature-based density. The original model of 50,000 vertices was first simplified to 20,629 vertices by local mesh adaptation. The initial sampling using area-based remeshing required only 8 iterations. To obtain the same resampling using just the Lloyd procedure required 45 iterations. Note also that every area-based iteration that relocates each of the mesh vertices is about twice as fast as a Lloyd iteration. The polishing of the isotropy then took 15 Lloyd iterations. The entire remeshing was performed in less than two minutes on a Pentium 4 2.4GHz machine with 512MB of memory. Figure 6 shows a uniform remesh of the piecewise smooth model handisk, containing 5,000 vertices. The helmet, a genus 3 model, is remeshed with a curvature-related density function (see Figure 7). Figure 9 illustrates a uniform remeshing of the David model, part of the digital Michelangelo project [22]. The irregular and non-uniform input mesh contains 350,000 vertices, while the remeshed model has 100,000 vertices. The initial vertex partition stage runs for 5.5 minutes, and the vertex placement runs for 4 minutes. We chose this model for illustrating the scalability and the adaptability of our technique to handle both complex models and arbitrary genus. Figure 8 shows a closeup of the same model to emphasize the quality of sampling obtained by centroidal tessellation.

8. CONCLUSION

This paper has introduced a technique for efficient and precise isotropic surface remeshing. Our approach first performs efficient sampling of the mesh with respect to a density function using the area-based remeshing technique. A Lloyd relaxation stage that constructs a weighted centroidal Voronoi tessellation is then directly applied on the mesh to ensure precise isotropic placement of the vertices. Using a patch-wise parameterization technique to apply a local 2D Lloyd relaxation on the 3D mesh allows us to handle complex models with arbitrary genus and any number of boundaries. Thus, by combining state-of-the-art techniques we are able to efficiently produce high quality isomorphic remeshings. One limitation of our method is the convergence be-
behavior of the Lloyd relaxation process for precise isotropic vertex placement. As explained in [8], local convergence is guaranteed in 2D when the density function is log concave. Since in our case the density function is either uniform when requested, or a function of the curvature, this does not guarantee the local convergence in all cases. Nevertheless, it was not an issue in our experiments. As future work we plan to accelerate further the Lloyd relaxation.

References

Figure 6: Uniform remeshing of the fandisk model (piecewise smooth).

Figure 7: Curvature-adapted remeshing of the helmet model (genus 3).


Figure 9: Left: Digital Michelangelo David model (350k vertices). Right: uniform remeshing (100k vertices).
Session 2B
Quality
AN ADAPTABLE SURFACE PARAMETERIZATION
METHOD

P. Degener, J. Meseth and R. Klein
University of Bonn
Institute of Computer Science II
Römerstrasse 164
D-53117 Bonn, Germany

August 12, 2003

ABSTRACT

Parameterizations of triangulated surfaces are used in an increasing number of mesh processing applications for various purposes. Although demands vary, they are often required to preserve the surface metric and thus minimize angle, area and length deformation. However, most of the existing techniques primarily target at angle preservation while disregarding global area deformation.

In this paper an energy functional is proposed, that quantifies angle and global area deformations simultaneously, while the relative importance between angle and area preservation can be controlled by the user through a parameter. We show how this parameter can be chosen to obtain parameterizations, that are optimized for an uniform sampling of the surface of a model. Maps obtained by minimizing this energy are well suited for applications that desire an uniform surface sampling, like re-meshing or mapping regularly patterned textures.

Besides being invariant under rotation and translation of the domain, the energy is designed to prevent face flips during minimization and does not require a fixed boundary in the parameter domain. Although the energy is non-linear, we show how it can be minimized efficiently using non-linear conjugate gradient methods in a hierarchical optimization framework and prove the convergence of the algorithm.

The ability to control the tradeoff between the degree of angle and global area preservation is demonstrated for several models of varying complexity.

Keywords: parameterization, uniform sampling, metric, re-meshing

1. INTRODUCTION

Parameterization denotes the task of finding a two dimensional map for a surface in a higher dimensional space. In computer graphics such maps have recently gained much interest, since they are used in many applications ranging from re-meshing, texture mapping and surface reconstruction to 3d painting systems, surface editing [2] and geometry images [8].

Most of these applications demand parameterizing maps (in the following also called parameterizations) that preserve the metric structure of the surface, i.e. respect area and angles of shapes. Unfortunately, in general such an angle and area preserving parameterization does not exist, thus angle preservation has to be traded off against area preservation. Many existing methods focus on angle preservation only, which often leads to large global area distortion resulting in visually disturbing artifacts on textured surfaces even if angular distortions are small.

Simultaneously optimizing angle and global area deformation is often superior as shown in figure 1. Furthermore combined global area and angle optimization is
also important whenever an uniform surface sampling is desired, as in 3D painting systems or surface editing applications. However, by now only few approaches considering both angular and area distortion simultaneously exist, all of which exhibit certain drawbacks.

1.1 Previous Work

Over the last years a lot of research has been done in the area of surface parameterization. Besides methods that optimize the parameterization for a given surface signal like Balmelli et al.[1] and Sander et al.[21], most approaches aim at minimizing a metric distortion.

In the context of parameterization, harmonic maps [6, 5] were first used by Eck et al.[4]. To compute harmonic maps, Eck et al. derive appropriate weights for a system of edge springs which can be efficiently solved. However, the texture coordinates for boundary vertices must be fixed a priori and harmonic maps may contain face flips (adjacent faces in texture space with opposite orientation) which violate the bijectivity of a parameterization. Based on earlier work by Tutte[27], Floater[7] proposes a different set of weights for the edge spring model that guarantees bijectivity if the texture coordinates of the boundary are fixed to a convex polygon. Desbrun et al.[3] define a space of measures spanned by a discrete version of the Dirichlet energy [19], and a discrete authalic energy. While the authalic energy remedies local area deformations, it requires fixed boundaries and results cannot achieve the quality of methods targeted at global length preservation such as Sander et al.[22].

In Hormann and Greiner[12] mostly isometric parameterizations are introduced that minimize a non-linear energy. A variant of this energy is also used in this paper. Mostly isometric parameterizations do not require boundary texture coordinates to be fixed and avoid face flips. Furthermore, mostly isometric parameterizations approximate mathematically well studied continuous conformal maps, i.e. maps that perfectly preserve angles.

Another approach to minimize angular distortion is proposed by Sheffer and de Sturler[24]. They define a non-linear energy in terms of the corner angles of the mesh in texture space. Lévy et al.[17] formulate the discrete conformality problem as an unconstrained quadratic minimization problem and prove the uniqueness and existence of its solution. Using a standard numerical conjugate gradient solver they are able to compute least squares approximations to continuous conformal maps very efficiently without requiring fixed boundary texture coordinates. However, in seldom cases triangle flips may occur.

In addition, some methods exists which compute parameterizations over a non planar domain. Haker et al.[9] compute conformal maps from a spherical domain onto a three dimensional surface. In Lee et al.[16] a mesh simplification[10] is used to parameterize a surface over a base mesh. A similar approach is taken by Khodakovsky et al.[14] but with emphasis on globally smooth derivatives.

Besides angle preserving methods, only a few approaches explicitly optimize global area or global length distortion: Maillot et al.[18] minimize an edge length distortion, but cannot guarantee the absence of face flips. The authors also propose an area preserving energy and combine both energies in a convex combination. Sander et al.[22] minimize the average or maximal singular value of the Jacobian to prevent undersampling of the surface. However, since they only penalize undersampling, oversampling of a triangle may nevertheless occur. To optimize for a uniform sampling Sorkine et al.[26] minimize the maximum of the maximal singular value and the inverse of the minimal singular value, which penalizes both under- and oversampling. While they obtain impressive results, their functional is not differentiable and thus not suitable for fast non-linear minimization techniques as the conjugate gradient method for example.

Iterative smoothing of an overlay grid is proposed by Sheffer and de Sturler[25] as a post-processing step for angle preserving parameterization algorithms. However, it is not clear what impact the post-processing has on the angle preservation.

1.2 Contribution

In this paper, we propose a metric energy that simultaneously measures angular and global area deformations imposed by a parameterization. On surfaces with non zero Gaussian curvature, the unavoidable deformation of angles and areas is traded off by the energy in an user-controlled way. Furthermore, we show how
this functional can be used to optimize parameterizations for a uniform surface sampling.

It is designed to prevent face flips during optimization and does not require fixed boundary texture coordinates. Furthermore it is invariant under rotation and translation of the domain. Although the derived energy is non-linear, it is differentiable and well suited for a hierarchical minimization as proposed by Hormann et al. [13]. We show how angle and global area optimized parameterizations can be computed efficiently with guaranteed convergence using non-linear conjugate gradient methods.

Usually models are cut into charts before being parameterized. In the present paper we do not tackle this problem, but our method can be combined with any charting and seaming algorithm available like the ones introduced in [23, 17, 22].

Besides face flips the bijectivity of the parameterization can also be violated if the texture mesh intersects itself. Although the method proposed here does not prevent these self intersections, they occur only in seldom cases and can be handled in a post processing step as proposed in [24].

2. ISOMETRIC DISTORTION

2.1 General Setup and Notation

Given an orientable 2-manifold surface patch \( S \subset \mathbb{R}^k \) a parameterization is defined as a homeomorphism

\[
\phi : \Omega \subset \mathbb{R}^2 \rightarrow S \subset \mathbb{R}^k
\]

from the parameter space \( \Omega \) into \( S \). In the following we consider the problem of finding a parameterization for a set \( S \) that has a triangulation

\[
\mathcal{M} = \{1 \ldots n\}, T, (p_i)_{i=1 \ldots n}
\]

where \([1 \ldots n]\) denotes the vertices, \( T \subset [1 \ldots n]^3 \) represents triangles and \( p_i \) is the location of vertex \( i \) in \( S \). Furthermore, we require the inverse parameterization \( \psi := \phi^{-1} \) to be linear within the triangles of \( \mathcal{M} \).

Such a mapping \( \psi \) is uniquely determined by its values \((\psi_{ui}, \psi_{vi})_{i=1 \ldots n} := (\psi(p_i))_{i=1 \ldots n} \) on the mesh vertices and

\[
\mathcal{M} = \{1 \ldots n\}, T, (u_i, v_i)_{i=1 \ldots n}
\]

is a parameter domain triangulation for the image \( \psi(S) \). The inverse parameterization \( \psi \) maps vertices and faces of \( \mathcal{M} \) onto vertices and faces of \( \mathcal{M} \) respectively. In the following \( \Delta_M((l, m, n)) \) with \((l, m, n) \in T \) denotes the triangle \((p_l, p_m, p_n) \) in \( S \). Analogously, \( \Delta_M(T) \) will be used to denote triangles in \( \Omega \).

Since a homeomorphism respects the topology and as we assume a planar domain \( \Omega \subset \mathbb{R}^2 \) the surface patch is required to have genus zero.

2.2 Measuring Distortion

Given a differentiable parameterization

\[
\phi : \Omega \subset \mathbb{R}^2 \rightarrow S \subset \mathbb{R}^k
\]

the first fundamental form \( I_\phi \), which captures the metric structure of \( S \), is defined as

\[
I_\phi = \nabla \phi \cdot \nabla \phi = \begin{pmatrix} a & b \\ b & c \end{pmatrix}
\]

with \( a = \left\| \frac{\partial \phi}{\partial u} \right\|^2 \), \( b = \left( \frac{\partial \phi}{\partial u}, \frac{\partial \phi}{\partial v} \right) \) and \( c = \left\| \frac{\partial \phi}{\partial v} \right\|^2 \).

Since \( I_\phi \) is a symmetric positive definite 2x2 matrix in every \( \omega \in \Omega \) it induces a scalar product on \( \mathbb{R}^2 \) which describes the lengths and angles of vectors in \( \mathbb{R}^2 \) after being mapped by \( I_\phi \).

In the following we briefly review an angle preserving condition in terms of \( I_\phi \) and formulate a similar condition for global area preservation. In section 2.5 an energy functional \( \mathcal{E} \) on the space of valid parameterizations is proposed that quantifies both angle and area deformation.

2.3 Conformal Maps

A result dating back to 1851 known as the Riemann mapping theorem guarantees for surface patches homeomorphic to a disk the existence of a conformal differentiable parameterization with continuous derivatives. A parameterization is said to be conformal if for every \( \omega \in \Omega \)

\[
I_\phi(\omega) = \lambda(\omega) \cdot I
\]

where \( I \) denotes the 2x2 identity matrix.

In other words the derivatives of the iso-u and iso-v curves passing through \( \phi(\omega) \) are orthogonal and of the same magnitude. Thus conformal mappings preserve the angles. Denoting the maximal and minimal eigenvalue of \( I_\phi \) by \( \lambda_{\text{max}} \) and \( \lambda_{\text{min}} \) respectively, the conformality can equivalently be expressed as

\[
\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} = 1
\]

Since \( 0 < \lambda_{\text{min}} \leq \lambda_{\text{max}} \), one is the minimal value of the ratio of the eigenvalues and we choose to minimize this ratio to optimize angular distortion.

2.4 Area Distortion

The conformality condition allows the directional derivatives to be uniformly scaled by a factor \( \lambda(\omega) \) that may vary if we travel from point to point on the surface. If this factor does not equal one, a shape in the
domain appears stretched or shrunk when mapped onto the surface and its area is distorted.

Since $\lambda(\omega)$ is continuous, around every $\omega \in \Omega$ a sufficiently small neighborhood exists, where the variation of $\lambda(\omega)$ is arbitrarily small. Thus, area is locally but not globally preserved by a conformal map. Conformal maps are therefore well suited for applications where angle preservation is required, but global area preservation is less important.

If in addition to angles, area is to be preserved globally, the magnitude of the directional derivatives has to be fixed leading to the notion of isometry. A parameterization is said to be isometric if

$$\lambda(\omega) = 1$$

for all $\omega \in \Omega$. Stated differently the first fundamental form equals the identity matrix in every point.

Isometry is stronger than conformality in the sense that it requires the tangent vectors to be orthogonal and have unit length in every point of the surface. An isometric parameterization preserves angles and area globally. Unfortunately, isometric parameterizations exist only for surfaces with zero Gaussian curvature. In the general case of non zero Gaussian curvature, angle and area preservation have to be traded off.

To find the area deformation imposed by a map $\phi$, we consider a sufficiently small axis aligned square in $\Omega$ of area $A$. The image of this square is a trapezoid spanned between the directional derivatives in $u$ and $v$ whose area is given by $A \cdot \sqrt{\det I_\phi}$ and thus $\phi$ preserves area if and only if

$$\sqrt{\det I_\phi} = 1$$

2.5 A Combined Energy

To enforce the area preservation condition proposed above, we choose $f(x) = x + \frac{1}{2}$ as objective function, since it is convex and attains its minimum in one. Furthermore it grows to infinity for both $x \to \infty$ or $x \to 0$. In the case of the area deformation energy

$$E_{\text{area}}(\omega) := f(\sqrt{\det I_\phi(\omega)}) = \sqrt{\det I_\phi(\omega)} + \frac{1}{\sqrt{\det I_\phi(\omega)}}$$

which is obtained by substituting $\sqrt{\det I_\phi(\omega)}$ for $x$, this property ensures that the orientation of all faces is preserved during the minimization and thus face flips cannot occur.

Using the same objective function for the angle deformation yields the conformal energy

$$E_{\text{angle}}(\omega) := f(\sqrt[\frac{1}{2}]{{\lambda_{\text{max}} \over \lambda_{\text{min}}}}) = \sqrt{{\lambda_{\text{max}} \over \lambda_{\text{min}}}} + \sqrt{{\lambda_{\text{min}} \over \lambda_{\text{max}}}}$$

that was proposed by Hormann in [11] and which is nothing but the MIPS energy that was used in [12] to compute angle preserving maps. The additional square root is used because the eigenvalues measure scale squared instead of scale.

Although a minimization of the area deformation energy alone is possible in theory, it causes severe numerical problems. The reason for this lies in the invariance of $E_{\text{area}}$ under shears: Since a shear does not change the area of a triangle, during the optimization of $E_{\text{area}}$ triangles may be arbitrarily sheared. Unfortunately such an extremely sheared triangle causes numerical problems in the minimization algorithm. We have thus decided to choose a combined energy as follows:

$$E_{\text{combined}}(\omega) := E_{\text{angle}}(\omega) \cdot (E_{\text{area}}(\omega))^\theta$$

where the parameter $\theta$ varies between 0 and $\infty$ and controls the relative importance of area and angle preservation. Our algorithm was able to minimize the combined energy function at least for values of $\theta < 2$. However, for higher values of $\theta$ numerical problems prevented the minimization of the energy in some cases. These problems are due to very tall and narrow texture triangles caused by shearing. Minimizing the energy on such triangles has a bad condition.

For the special choice of $\theta = 1$, the combined energy becomes the simple product

$$E_{\text{angle}}(\omega) \cdot E_{\text{area}}(\omega) = f(\sqrt[\frac{1}{2}]{{\lambda_{\text{max}} \over \lambda_{\text{min}}}}) \cdot f(\sqrt{\det I_\phi(\omega)}) = f(\lambda_{\text{max}}) + f(\lambda_{\text{min}})$$

where the fact $\det I_\phi = \lambda_{\text{min}} \cdot \lambda_{\text{max}}$ was used in the second equation. As the eigenvalues $\lambda_{\text{max}}$ and $\lambda_{\text{min}}$ measure the greatest and the smallest stretch respectively that the parameterization $\phi$ imposes on a vector of unit length, the energy obtained for $\theta = 1$ enforces an uniform sampling of the surface, and - similar to the energy proposed by Sorkine et al. in [26] - penalizes oversampling ($\lambda_{\text{min}} < 1$) as well as undersampling ($\lambda_{\text{max}} > 1$).

A parameterization $\phi$ can now be assigned a combined area and angle distortion by integrating over the surface patch $S$

$$E(\phi) := \int_S E_{\text{combined}}(\phi^{-1}(p)) \, dp$$

2.6 Discretization

For the special case of a piecewise linear parameterization over a triangulation, $\nabla \phi$ and $I_\phi$ are constant within each triangle of $M$ which in turn causes the energies $E_{\text{area}}(\omega)$, $E_{\text{angle}}(\omega)$ and $E_{\text{combined}}(\omega)$ to be constant within each triangle.
As shown by Hormann in [11] the MIPS energy of the linear map \( \phi|_{\Delta_M(T)} \) can be written using the notation from figure 2 as
\[
E_{\text{angle}}(T) = \frac{\cot \alpha |a|^2 + \cot \beta |b|^2 + \cot \gamma |c|^2}{2 \text{area}(\Delta_M(T))}
\]
Furthermore we have for the linear map \( \phi|_{\Delta_M(T)} \)
\[
\sqrt{\det I_{\phi|_{\Delta_M(T)}}} = |\det \nabla \phi|_{\Delta_M(T)}| = \frac{\text{area}(\Delta_{M'}(T))}{\text{area}(\Delta_M(T))}
\]
and therefore the area distortion measure within a triangle is given by
\[
E_{\text{area}}(T) = \frac{\text{area}(\Delta_{M'}(T))}{\text{area}(\Delta_M(T))} + \frac{\text{area}(\Delta_M(T))}{\text{area}(\Delta_{M'}(T))}
\]
Finally the integral becomes the finite sum
\[
E(\phi) = \sum_{T \in \mathcal{T}} E_T \cdot \text{area}(\Delta_{M'}(T))
\]
where
\[
E_T := E_{\text{angle}}(T) \cdot E_{\text{area}}(T)^\theta
\]
only depends on the coordinates and texture coordinates of the three vertices in \( T \).

### 2.7 Properties

The distortion measure \( E \) derived in the previous section has some important properties that should be briefly mentioned:

1. **Invariance under Rotation and Translation**
   Since \( E \) is defined in terms of \( I_\phi \) which is in turn defined in terms of \( \nabla \phi \), it is invariant under the translation \( \phi(\omega + t) \) of the domain by a constant vector \( t \). If the domain is transformed by some orthogonal transformation \( R \), the first fundamental form becomes
   \[
   I_{\phi \circ R} = R^T \nabla \phi^T \nabla \phi R,
   \]
i.e. its eigenvalues and determinant do not change. Depending only on the eigenvalues and the determinant of the first fundamental form, \( E \) is thus invariant under such a transformation \( R \). However, the distortion measure is not invariant under uniform scalings.

2. **Differentiability** The partial derivatives \( \frac{\partial E}{\partial u_i} \) and \( \frac{\partial E}{\partial v_i} \) exist for any valid piecewise linear parameterization, allowing for an efficient minimization of the functional in \( (u_i, v_i) \).

3. **Infinite Error for Degenerate Mappings** For a valid parameterization, the mesh in the domain \( M \) contains no triangles degenerated to a point or a line and all faces are consistently oriented. If a triangle in \( M \) tends to degenerate, the parameterization has to stretch an infinitesimal small triangle onto the non-degenerated surface triangle. As \( \lambda_{\text{max}} \) measures the greatest stretch imposed on a vector of unit length it tends to infinity as a triangle tends to degenerate. This in turn causes both the MIPS energy and the area distortion energy to attain arbitrary high values.

Having this property, we can - following Sander et al.[22] - continue the error functor on degenerate configurations \( ((u_i, v_i)), i=1...n \) by assigning them an infinitely high error. The minimization then automatically avoids such degenerated configurations, thus a consistent face orientation can be guaranteed. The property described above ensures that the continuation on degenerated configuration is continuous, which is essential for the numerical minimization.

### 3. MINIMIZING ISOMETRIC DISTORTION

#### 3.1 Hierarchical Optimization

To minimize the non-linear isometric energy described in the previous section we use the hierarchical parameterization algorithm proposed by Hormann et al.[13]. A hierarchical approach is reasonable since it speeds up the computation and helps to circumvent local minima of the energy functional. Since in contrast to other energies boundary vertices do not need to be fixed, the proposed energy is well suited for a hierarchical optimization.

The method proposed by Hormann et al. computes a progressive mesh sequence [10] of \( M \), grouping independent splits in sets. These sets define a natural hierarchy for the optimization, with each set containing approximately 25% of the vertices of the subsequent stage. For further details on the generation of the split sets and the hierarchy please refer to Hormann et al.[13].

The actual optimization of the energy functional \( E \) on each level of the hierarchy uses a relaxation method, which is further described in the next section. Algorithm 1 shows a short overview over the basic steps of the algorithm.
Listing 1: Basic steps of the optimization algorithm

```
// build sets of independent splits
sets = buildSets();

// relax base mesh
mesh = baseMesh;
while(!convergence){
    relax(mesh);
}

for(int i=0;i<sets.size (); i++){
    // apply splits of the next set to the mesh
    mesh.applyToMesh(sets[i]);

    // find save texcoords for new vertices
    generateSaveTexcoords();

    // relax mesh
    while(!convergence){
        relax(mesh);
    }
}
```

3.2 Vertex Relaxation

The texture coordinates \((u_i, v_i)\) of vertex \(i\) affect only those \(E_T\) for which \(i\) is incident with \(T\). More specifically only the partial sum

\[
E_i := \sum_{T \in \text{1-ring}(i)} E_T
\]

of \(E\) is influenced by \((u_i, v_i)\).

Given an initial configuration \(((u_i, v_i))_{i=1...n}\), the vertex relaxation consists of two steps: First all vertices are ordered by the error \(E_i\), that they contribute to the overall error. Then for each vertex \(i\) \(E_i\) is optimized in \((u_i, v_i)\) while keeping all other texture coordinates fixed.

Since the functional provides partial derivatives, the Polak Ribiere method [20] — a non-linear conjugate gradient optimizer — was used to optimize \(E_i\). This method includes a line search as a subtask, which was restricted to search for optimal vertex texture coordinates \((u_i, v_i)\) only within the kernel of the vertex’ 1-ring. The kernel of a polygon with a counterclockwise directed boundary is the intersection of all the half-planes lying to the left of the polygon’s edges (see [16]). Since the faces of a mesh \(M\) in the plane are consistently oriented if and only if every vertex lies within the kernel of its 1-ring, the relaxation does not cause any face flips, provided that the initial configuration is free of flips.

Minimizing the partial sum \(E_i\) of \(E\) in each step, the relaxation decreases \(E\) monotonously and as the energy has a lower bound of zero, \(E\) is guaranteed to converge, ensuring that the ‘while’ loops in algorithm 1 terminate eventually.

3.3 Initial Vertex Placement

Since the surface patch is assumed to have genus zero, the simplification produces a base mesh that consists only of a single triangle. The texture coordinates of its vertices are initialized to a congruent triangle in the plane centered in the origin.

Whenever a set of splits is applied to the mesh during the hierarchical optimization, texture coordinates for the newly inserted vertices have to be found. In [13] barycentric coordinates obtained by exponential mapping are stored for each vertex during the simplification of the mesh. After a split these stored coordinates are used to assign a texture coordinate to the newly inserted vertex. However, in some cases the texture coordinate obtained this way is invalid, i.e. some of the triangles in the 1-ring are flipped. In order to ensure a valid configuration for the subsequent relaxation step, we take a different approach here.

To ensure a flipless configuration, the texture coordinate of the new vertex has to be inside the kernel of its 1-ring. One possible choice is certainly to use the center of the kernel as an initial texture coordinate for the new vertex. However, this choice requires the explicit computation of the 1-ring kernel. In contrast, the intersection of a polygon kernel with a line does not require the computation of the kernel. It is simply the intersection of the line with the halfplanes defined by the edges of the polygon.

To find a valid position for the new vertex, we repeatedly cast a ray from the texture coordinate of the vertex that is to be split in a random direction and intersect it with the 1-ring kernel. As soon as a non-empty intersection is found we choose the center of this intersection interval as an initial texture coordinate for the new vertex. Usually one or two kernel-ray intersections suffice to position the new vertex.

4. RESULTS

We applied our method to various models which are listed in table 1 together with the isometric distortions that were used in Sander et al.[22] are listed. The timings were taken on an AMD Duron 800MHz with 256MB memory.

Our initial motivation was to compute parameterizations that not only preserve angles but also minimize global area deformations. The results shown in Figure 4 were obtained for an angle/area weighting of \(\theta = 1\). All of these surfaces have large areas of non zero
Table 1: Models and Statistics: In addition to our own energy, values of the $L_2$ and $L_\infty$ energies are listed. Stages corresponds to the number of independent split sets. Errors were normalized to the surface area. For all models we chose $\theta = 1$.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vertices</th>
<th>Error</th>
<th>$L_2$</th>
<th>$L_\infty$</th>
<th>Stages</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxPlanck</td>
<td>25445</td>
<td>9.35</td>
<td>1.45</td>
<td>2.92</td>
<td>41</td>
<td>308.90</td>
</tr>
<tr>
<td>Venus</td>
<td>29322</td>
<td>4.55</td>
<td>1.21</td>
<td>1.77</td>
<td>41</td>
<td>300.46</td>
</tr>
<tr>
<td>Cat</td>
<td>4539</td>
<td>4.65</td>
<td>1.36</td>
<td>3.73</td>
<td>33</td>
<td>47.65</td>
</tr>
<tr>
<td>Horsehead</td>
<td>2893</td>
<td>4.85</td>
<td>1.36</td>
<td>3.53</td>
<td>32</td>
<td>33.79</td>
</tr>
<tr>
<td>Ear</td>
<td>2150</td>
<td>2.21</td>
<td>1.03</td>
<td>1.56</td>
<td>30</td>
<td>23.03</td>
</tr>
</tbody>
</table>

Gaussian curvature but only small boundary loops and thus can only be parameterized with high angle or area distortions. As a reference conformal maps were computed (right column of figure 4) using the LSCM method proposed in Lévy et al.[17]. The comparison shows that the maps obtained by our method trade part of the angle preservation to improve global area preservation as expected.

In Figure 5 the impact of the parameter $\theta$ on the parameterization is shown. As expected intuitively, a high value favors global area preservation, while smaller values emphasize the preservation of angles. For $\theta = 0$ the resulting map is similar to those obtained by the LSCM [17] or MIPS [12] method.

This observation can also be verified in the distortion histograms for the horse head dataset shown in figure 6. As in Lévy et al.[17] the area distortions were computed in each triangle as the ratio of texture area to model area. The angle histogram shows the distribution of the angles between the $u$ and $v$ directional derivatives in each triangle. For values of $\theta$ close to zero, the angle histogram shows a distinct peak, while the deviation in the area histogram is much higher. For higher values of $\theta$ this relation is reversed.

As mentioned above, the proposed energy does not require boundary vertices to be fixed. Thus, for any value of $\theta$ the minimization can also find an optimal boundary for $\mathcal{M}$. Figure 3 shows the parameterization obtained for a S-shaped model and the corresponding triangulation $\mathcal{M}$ in texture space.

In practice models are usually preprocessed by a chart or seam cutting algorithm before a parameterizing algorithm is applied. These cutting algorithms generate one or more charts with lower Gaussian curvature and larger boundaries which facilitate parameterization. But since the resulting parameterization heavily depends on the quality of the cutting method used, the parameterizations presented in this paper were obtained for uncut models. Only minor modifications were made to ensure a disk like topology. However, the preprocessing was only skipped for means of demonstration and the results certainly improve a lot by using a charting or seaming algorithm like those proposed in Sheffer and Hart[23] or Lévy et al.[17].

Figure 3: Parameterization obtained for a dented S-shaped model with non-convex boundary and the corresponding mesh in texture space.
Figure 4: These models are parameterized as they appear, without charting or seaming. The column on the left shows the results obtained with our method for $\theta = 1.0$. On the right a conformal mapping is shown.
5. CONCLUSION

In this paper we have proposed an energy functional that measures an isometric distortion of a parameterization. On surfaces for which no isometric parameterization exists, the functional weights global area and angle deformation in an intuitive and user-controlled way. We also showed, how the functional can be used to optimize for an uniform surface sampling.

Besides basic desirable properties, the functional can be continuously continued on degenerated parameterizations and does not depend on fixed boundary vertex texture coordinates which makes it possible to compute parameterizations without face flips and with optimal boundaries. Using conjugate gradient methods and hierarchical optimization we showed how the functional can be minimized efficiently.

In future works we would like to experiment with different edge collapse schedules in the generation of the splits sets during hierarchical optimization, to further speed up the computation of parameterizations. Furthermore we would like to address the numerical problems for higher values of $\theta$.

References


Figure 6: Per triangle distortions in area and angle for the horse head dataset for (from top to bottom) $\theta = 0.3, \theta = 1.0$ and $\theta = 3.0$, as shown in figure 5.


THE MESQUITE MESH QUALITY IMPROVEMENT TOOLKIT

Michael Brewer\textsuperscript{1}  Lori Freitag Diachin\textsuperscript{1}  Patrick Knupp\textsuperscript{1}  Thomas Leurent\textsuperscript{2}
Darryl Melander\textsuperscript{1}

\textsuperscript{1}Sandia National Laboratories, Albuquerque, NM U.S.A
\{mbrewer,ldiach,pknupp,djmelan\}@sandia.gov
\textsuperscript{2}Argonne National Laboratories, Chicago, IL U.S.A. tleurent@mcs.anl.gov

ABSTRACT

We describe the design goals and architecture of the Mesquite toolkit, a stand-alone library consisting of state-of-the-art algorithms for mesh quality improvement. The primary considerations in Mesquite design are to ensure that it is comprehensive, effective, efficient, and extensible. We give an overview of the Mesquite architecture and highlight the core classes, their inter-relationships, and functionality. We describe the interfaces developed to obtain information from the mesh and geometry and to provide user-control of the quality metrics, algorithms, and termination criterion. Smoothing results for several meshes with a broad range of characteristics are given which showcase Mesquite’s versatility.

Keywords: mesh quality, mesh improvement, mesh smoothing, mesh generation

1. INTRODUCTION

Mesh quality is critical for accuracy and efficiency in the solution of PDE-based applications. Quality can be affected in many stages of the solution process from de-featured CAD models, to mesh generation procedures and \( h \)- and \( r \)-adaptive schemes. At any stage after a mesh is created, its quality can be improved by node point movement methods, commonly called mesh smoothing, and topology modification schemes such as edge and face flipping. There are a number of techniques that have been developed for mesh improvement ranging from simple Laplacian smoothing [1] to more sophisticated algorithms such as Winslow smoothers for structured meshes [2] and the numerical optimization methods and topology modification schemes recently developed for unstructured meshes (for example, [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]).

The software associated with many of these references is largely contained in larger mesh generation or application frameworks. While many meshing codes such as ICEM-CFD can perform mesh improvement, there currently exist few stand-alone, portable libraries for mesh quality improvement that can be easily linked via functional interfaces to many different mesh generation and application codes. One such library is Opt-MS [13], a stand-alone package that uses state-of-the-art optimization algorithms to improve homogeneous simplicial meshes. However, Opt-MS only provides patch-based smoothing algorithms for isotropic elements in volumes or on planar surfaces. No topological changes are supported. Under the auspices of the Tera-scale Simulation Tools and Technologies (TSTT) Center [14], we have recently begun development of a new toolkit, called Mesquite (Mesh Quality Improvement Toolkit), that is much more comprehensive than Opt-MS and, in this paper, we present our progress to date.

Successful development of Mesquite will provide a robust and effective mesh improvement tool to the broader scientific community. This will allow both mesh generation researchers and application scientists
to benefit from the latest developments in mesh quality control and improvement. For example, Mesquite has already been used to smooth Geodesic meshes for the climate group at Colorado State University [15] and is presently being used to improve the quality of meshes used in the design of particle accelerators [16]. Preparations are also underway to link Mesquite to various TSST meshing codes including CUBIT [17], Overture [18], and NWGrid [19].

The Mesquite design goals were first introduced in [20], and in this paper we compare the current status of Mesquite with that vision (Section 2), give an overview of the Mesquite design and architecture including the application programming interfaces (Sections 3 and 4), and show several examples of meshes with a broad range of characteristics that have been improved by Mesquite (Section 5).

2. MESQUITE DESIGN GOALS

Mesquite design goals are derived from a mathematical framework and are focused on providing a versatile, comprehensive, effective, inter-operable, and efficient library of mesh quality improvement algorithms that can be used by the non-expert and extended and customized by experts. In this section we highlight the current status of Mesquite in several of our design goal areas.

Mathematical Framework. Mesquite design is based on a currently evolving but rigorous mathematical framework that poses the mesh quality improvement problem as an optimization problem. That is, suppose a mesh \( M \) contains \( n \) vertices, \( v_i \), and \( k \) elements, \( e_k \). Then the quality of each element or vertex is given as a general function \( q_i(x) \) of the coordinates of the vertex locations. The index \( i \) ranges from 1 to \( n \) or 1 to \( k \) depending on whether a vertex-based or element-based metric is chosen. A mesh quality objective function \( F = \int q_i(x) \) for \( i = 1, \cdots, n \) or \( i = 1, \cdots, k \) is formed to give an overall measure of mesh quality where \( n \) and \( k \) are the number of vertices or elements in a mesh sub-domain needing improvement, respectively. Mesquite is designed to solve the minimization problem \( \min F \) for a broad collection of quality metrics \( q_i \) and objective functions \( F \).

Versatile and Comprehensive. Mesquite works on structured, unstructured, and hybrid meshes in both two and three dimensions. The design permits improvements to meshes composed of triangular, tetrahedral, quadrilateral, and hexahedral elements. Prismatic, pyramidal, and polyhedral elements can be easily added. It currently incorporates only methods for node movement; plans for topology modification and hybrid improvement strategies lie in the future. Node movement strategies include both local patch-based iteration schemes for one or a few free vertices and global methods which improve all vertices simultaneously. Mesquite will be applicable to both adaptive and nonadaptive meshing and to both low- and high-order discretization schemes, but currently works with non-adaptive meshes containing linear elements.

Effective. Mesquite uses state-of-the-art algorithms and metrics to guarantee improvement in mesh quality. Because the definition of mesh quality is application specific, we provide quality metrics that allow the user to untangle meshes, improve mesh smoothness, element size, and shape. In the future these metrics will be referenced to permit non-isotropic smoothing and adaptivity. The software is easily customizable, enabling users to insert their own quality metrics, objective functions, and algorithms and also provides mechanisms for creating combined approaches that use one or more improvement algorithms.

Inter-operable. To ensure that Mesquite is inter-operable with a large number of mesh generation packages, we will use the common interfaces for mesh query currently under development by the TSST center. These interfaces provide uniform access to mesh geometry and topology and will be implemented by all TSST center software including several DOE-supported mesh generation packages. We are working with the TSST interface design team to ensure that Mesquite has efficient access to mesh and geometry information through strategies such as information caching and agglomeration. We are also participating in the design of interfaces needed to support topological changes generated by mesh swapping and flipping algorithms and to constrain vertices to the surface of a geometrical model.

Efficient. The outer layers of Mesquite use object-oriented design in C++ while the inner kernels use customizable coding constructs such as arrays and inlined functions. To ensure efficient use of computationally intensive optimization algorithms, we employ inexpensive smoothingatives, such as Laplacian smoothing, as “preconditioners” for the more expensive optimization techniques. In addition, mesh culling algorithms can be used to smooth only those areas of the mesh that require improvement. Considerable attention has been devoted to understanding and implementing a variety of termination criteria that can be used to control the computational cost of the optimization algorithms.

3. MESQUITE ARCHITECTURE

The Mesquite architecture, shown in Figure 1, closely follows the abstractions defined by the mathematical framework to describe the optimization problem. In particular, the core abstract classes needed to define a mesh quality improvement algorithm
are \texttt{QualityMetric}, \texttt{ObjectiveFunction} (which takes a \texttt{QualityMetric} as input), and \texttt{QualityImprover} (which takes an \texttt{ObjectiveFunction} as input).

In addition, a number of other classes have been created to support the needs of mesh quality improvement algorithms:

- \texttt{QualityAssessor}: to provide an evaluation of mesh quality using standard statistical procedures,
- \texttt{TerminationCriterion}: to customize the stopping criteria used with a mesh quality improvement algorithm,
- \texttt{InstructionQueue}: to compose quality improvers and quality assessors together to form efficient mesh quality improvement and evaluation methods, and
- \texttt{MeshSet} and \texttt{PatchData}: to provide the mechanisms for managing the application mesh and geometry information and the mesh sub-domains used in optimization procedures.

The Mesquite architecture uses as much dynamic polymorphism in the form of inheritance and virtual functions as is possible without degrading performance. This allows developers to easily add new functionality to Mesquite by inheriting from the appropriate abstract class and implementing its interface (i.e. its abstract virtual functions). For example, to implement a new quality metric, a user must inherit from the base \texttt{QualityMetric} class and implement only a single function that returns the value of the metric for a given mesh entity. We note that the mesh entities, defined by class \texttt{MeshEntity}, are the only exception in our architecture. Mesh entities, i.e. triangles, tetrahedrons, quadrangles and hexahedrons, are implemented in a single class, without the use of dynamic polymorphism, to eliminate the performance impact of runtime resolution.

In this section, we describe the details of the core classes \texttt{QualityMetric}, \texttt{ObjectiveFunction}, \texttt{QualityImprover}, \texttt{TerminationCriterion}, \texttt{QualityAssessor}, \texttt{MeshSet}, \texttt{PatchData}, and \texttt{InstructionQueue}. In each case, we give details of the design and discuss the functionality currently implemented in Mesquite. Often, we will re-
for to software patterns widely used in object-oriented design — the details of those design patterns are described in [21].

3.1 Quality Metrics

In Mesquite, the QualityMetric class provides a measure of the quality of individual mesh entities. Quality metrics can evaluate either element quality (for example, the mean ratio shape quality metric) or vertex quality (for example, the sum of the adjacent edge lengths squared can be used as a measure of vertex smoothness). The primary functionality associated with the QualityMetric class is the ‘evaluate’ function which returns a single quality value for a given mesh entity.

To increase the flexibility of the QualityMetric class, we also provide a number of mechanisms that allow the user to modify the metric in a variety of ways. For example, the condition number quality metric for a quadrilateral or hexahedral element is computed by evaluating the condition number at a number of sample points in the element. Mesquite allows the user to select which set of sample points are used in this calculation (e.g., the element’s vertices) and how these values are combined to form a single metric value (e.g., linear averaging). In addition, Mesquite is currently being extended to allow the user to reference certain quality metrics to a non-isotropic element.

In addition to the quality metric function value, the QualityMetric class also provides the gradient and Hessian information needed for many optimization algorithms. Numerical approximations of the gradient and Hessian are automatically provided by the QualityMetric base class. Concrete instantiations can also optionally include analytical expressions which are potentially more computationally efficient. For example, in the case of the mean ratio quality metric implementation, the analytic calculation is approximately twice as fast as the numerical computation although this improved efficiency often comes at a higher implementation cost. The cost of implementing the analytical gradients and Hessians can often be alleviated, however, by the use of automatic differentiation tools (for example, [22]).

Mesquite also allows the user to scale metric values or combine multiple metrics together to form a composite metric. However, only metrics which are evaluated on the same type of mesh entity can be composed together. That is, Mesquite does not allow element-based metrics and vertex-based metrics to be added or multiplied together because the result is not a meaningful measure of either element or vertex quality.

Currently Available Quality Metrics. There are currently nine quality metrics available within Mesquite, and these are listed in Table 3.1 (detailed definitions of the metrics are given in [23]). These quality metrics are grouped by the type of mesh properties that they measure, in particular: shape, smoothness, volume, and untangle. There are also two composite metrics which allow users to multiply two metrics’ values together or to raise a single metric’s value to a given power. The latter allows for negative powers and can therefore be used to obtain the inverse of any Mesquite quality metric. We note that the implementation of the mean ratio metric has been extensively optimized, and analytical gradients and Hessians are available for that function. Other metrics currently use numerical gradients and Hessians. Future Mesquite development will include the implementation of metrics falling under other group headings such as orthogonality, shear, and alignment.

3.2 Objective Functions

While the QualityMetric class provides a way to evaluate the properties of individual mesh entities, the ObjectiveFunction class provides a way of combining those values into a single number for the domain of the optimization problem. This domain can either be the entire mesh or a sub-mesh containing a subset of the free vertices. For example, one available objective function template \( f \) is the \( \ell_2 \) function which is the standard \( \ell_2 \) vector norm squared. Given an element-based quality metric \( q \) and a mesh sub-domain \( E \), the mesh quality objective function \( \mathcal{F} \) would be the composition of the template objective function and the quality metric function

\[
\mathcal{F}(x) = f \circ q(x) = \sum_{i \in E} (q_i(x))^2 . \tag{1}
\]

The ObjectiveFunction derived class computes the value of \( \mathcal{F} \) and, for \( f \) and \( q \) satisfying the appropriate smoothness conditions, the mesh quality objective function’s gradient and Hessian with respect to the vertex positions. As with QualityMetric, Mesquite allows the gradient of \( \mathcal{F} \) to be calculated either analytically or numerically. Computing the gradient of \( \mathcal{F} \) numerically is computationally expensive but requires only the quality metric values. If the gradient is calculated analytically, the first derivative of the template objective function \( f' \) and the quality metric gradient \( \nabla q \) are both required.\(^1\) To obtain the gradient of \( \mathcal{F} \), the chain rule is applied

\[
\nabla \mathcal{F}(x) = \bigcup_{i \in E} \left[ \nabla q_i(x) (f' \circ q(x)) \right] , \tag{2}
\]

where \( f' \circ q(x) \) is a scalar and \( \bigcup_{i \in E} \) denotes the assembly over all the elements or vertices, for element-based

\(^1\)The quality metric gradients \( \nabla q \) can be provided either numerically or analytically.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Group</th>
<th>Mesh Type</th>
<th>Feasibility Region</th>
<th>Entity Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area Smoothness</td>
<td>Smoothness</td>
<td>Any</td>
<td>No</td>
<td>Elements</td>
</tr>
<tr>
<td>Aspect Ratio</td>
<td>Shape</td>
<td>Tri/Tet</td>
<td>No</td>
<td>Elements</td>
</tr>
<tr>
<td>Composite Mult.</td>
<td>Composite</td>
<td>Any</td>
<td>No</td>
<td>Either</td>
</tr>
<tr>
<td>Composite Power</td>
<td>Composite</td>
<td>Any</td>
<td>Yes/No</td>
<td>Either</td>
</tr>
<tr>
<td>Cond. Num.</td>
<td>Shape</td>
<td>Any</td>
<td>Yes</td>
<td>Elements</td>
</tr>
<tr>
<td>Corner Jacobian</td>
<td>Volume</td>
<td>Any</td>
<td>No</td>
<td>Elements</td>
</tr>
<tr>
<td>Edge Length</td>
<td>Smoothness</td>
<td>Any</td>
<td>No</td>
<td>Vertices</td>
</tr>
<tr>
<td>Edge Length Range</td>
<td>Smoothness</td>
<td>Any</td>
<td>No</td>
<td>Vertices</td>
</tr>
<tr>
<td>Mean Ratio</td>
<td>Shape</td>
<td>Any</td>
<td>Yes</td>
<td>Elements</td>
</tr>
<tr>
<td>Untangle Beta</td>
<td>Unangle</td>
<td>Any</td>
<td>Yes</td>
<td>Elements</td>
</tr>
<tr>
<td>Vert. Cond. Num.</td>
<td>Shape</td>
<td>Any</td>
<td>Yes</td>
<td>Vertices</td>
</tr>
</tbody>
</table>

Table 1: List of the current Mesquite quality metrics. The table also indicates the metric group, the mesh types for which the metric is valid, whether the metric is only valid within a feasible region, and the type of entity for which the metric is defined (elements or vertices). Note that there may be a feasible region for composite metrics depending on whether the underlying metrics require such a constraint.

3.3 Quality Improvers

The mesh quality improvement algorithms are a crucial component of the Mesquite framework. The two main types of improvement schemes designed into Mesquite are VertexMover and TopologyModifier for vertex relocation or topology modification, respectively. These methods take as input an ObjectiveFunction and often make extensive use of the gradient and Hessian information provided therein. When writing a new algorithm, the concrete QualityImprover always acts on an ObjectiveFunction pointer to retrieve the function value and gradient for a certain mesh and concrete ObjectiveFunction / QualityMetric combination.

A new quality improver is defined by inheriting from either the VertexMover or the TopologyModifier abstract class. Both classes are intermediate abstract classes inheriting from the QualityImprove class (see Figure 1). One important aspect of both vertex movers and topology modifiers is the ability to seamlessly perform their operations globally on the entire mesh or on local sub-patches of the mesh. The behavior is chosen by the user at compile time using the set_patch_type function from QualityImprover. The primary functionality in these intermediate classes is the implementation of the loop_over_mesh virtual function which shields the optimization algorithm developer from the need to distinguish between local or global patches. The appropriate mesh information is gathered into a “patch” by the MeshSet and PatchData classes and given to the QualityImprover in the loop_over_mesh function. The VertexMover base class also checks the outer termination criterion to stop iterating over mesh subsets (see section 3.4) and updates the application mesh after optimizing a patch.

or vertex-based quality metrics respectively. Analytical gradients have been implemented for all of the continuously differentiable template objective functions in Mesquite, and an analytical Hessian calculation has been implemented for the \( \ell_p \) template objective function.

Available Template Objective Functions. Mesquite currently has objective function templates for the standard \( \ell_p \) (where \( P \) is a positive integer) and \( \ell_\infty \) vector norms. A separate template is provided for \( \ell_p \), a function which has several nice properties including a sparse Hessian matrix. All of Mesquite's quality improvers are designed to minimize a given objective function. For objective functions that need to be maximized (e.g. an \( \ell_1 \) objective function using an inverted mean ratio metric), the function value is multiplied by negative one to obtain the equivalent minimization problem.

Mesquite has four composite objective functions that allow the user to add and multiply objective functions by each other or by scalar values. These can be used to modify the optimization problem, to ease interpretation of the objective function value, or to ensure compatibility with termination criterion based on objective function values. Unlike quality metrics, two objective functions can be combined even if the underlying quality metrics are defined on different entity types. That is, an objective function which operates on a vertex-based quality metric can be added to an objective function which operates on an element-based quality metric. This allows the user to have the maximum flexibility in defining an objective function with which to measure the quality of a mesh.
Available Quality Improvement Algorithms. There are currently three major concrete optimization algorithms implemented as VertexMovers in Mesquite: the conjugate gradient algorithm, the feasible Newton algorithm, and the active set algorithm. The optimization of mesh topology has been accounted for in the architecture, but no implementation is available yet.

Conjugate Gradient Algorithm. This algorithm is appropriate for optimizing any combination of $C^1$ objective functions and quality metrics (see [5] for more details). The conjugate gradient algorithm has a linear convergence property for most problems. By using the Polak-Ribière scheme to select a search direction which is a combination of the gradient at the current iteration with the gradient from one or more previous iterations, it avoids the zigzagging behavior exhibited by the steepest descent algorithm when the equal cost surfaces of the objective function are elongated (e.g., in narrow valleys). The algorithm requires the objective function value and gradient and can be used on mesh patches of any size.

Feasible Newton Algorithm. Newton’s method minimizes a quadratic approximation of a non-linear objective function. Newton’s method is known to converge super-linearly near a non-singular local minimum. Mesh-optimization problems that are performed within the neighborhood of the minimum are a perfect application for Newton’s method. The algorithm requires the objective function value, gradient, and Hessian information. In Mesquite this algorithm can be used with mesh patches of any size, making it appropriate to optimize all vertex positions simultaneously for any $C^2$ objective function with a sparse Hessian. In practice, users will often observe an order of magnitude improvement in computation time when using the feasible Newton algorithm instead of the conjugate gradient algorithm, making feasible Newton a worthwhile choice when applicable (see [5] for more details).

Active Set Algorithm. The active set algorithm has been developed for non-continuously differentiable objective functions such as those computing the maximum value of a quality metric within a patch of elements. This method is based on a non-smooth steepest descent algorithm which efficiently computes a search direction and step size based on the gradients of the values contained in the active set. Currently, this algorithm works on a patch of triangular or tetrahedral elements that contain a single free vertex. Repeated sweeps over the free vertices in the mesh leads to over-

all mesh improvement (see [24] for more details).

3.4 TerminationCriterion

Mesquite’s TerminationCriterion class contains functionality to customize the termination of the mesh quality improvement process. As mentioned previously, many quality improvement algorithms can perform mesh optimization on either the global mesh or on sub-meshes. In the latter case, the algorithm must be capable of determining when to terminate two processes: 1) an inner criterion is used to terminate the optimization algorithm on a sub-mesh and 2) an outer criterion is used to terminate the iteration over the sub-meshes. Typically, quality improvers that support both local and global optimizations always use two termination criteria. In the global case, the outer criteria is set to terminate the optimization process after one iteration.

Available TerminationCriterion. A wide range of cost, quality, and progress-centric termination criterion types have been studied. These criteria have two basic types - absolute or relative - depending on whether or not the criterion is scale dependent. Fourteen of these have been implemented in TerminationCriterion. Among the implemented types are criteria which terminate optimization procedures due to exceeding a set number of iterations, exceeding an allotted amount of time, or reaching a mesh with a sufficiently small objective function gradient. Any combination of the available criteria can be set on a given TerminationCriterion object. Compound criteria types consist of statements joined by ‘OR’, for which the optimization process will be terminated when any of the criteria have been satisfied. Currently we have found little use for compound criteria joined by ‘AND’, but this also could be implemented.

3.5 Quality Assessors

Mesquite’s QualityAssessor class encapsulates functionality to evaluate quality metric values for a given mesh, to accumulate statistical information about those values, and to report that data to the user. In particular, a QualityAssessor object takes a QualityMetric class as input to evaluate a given mesh and then reports information like the maximum, average, and standard deviation of those values.

3.6 Mesh Data Classes

There are two mesh data classes in Mesquite, MeshSet and PatchData, which have been designed to meet two different needs. The MeshSet class is a container that holds pointers, or handles, to the meshes provided by
the application. It does not store any detailed information about the mesh such as vertex coordinates or element connectivities, but provides the mechanisms necessary to obtain this information from the application through a well-defined, flexible API (see Section 3.6.1). Detailed mesh information obtained through the MeshSet API is stored only in the PatchData class for the sub-patches given to the quality improve procedures. The PatchData class makes Mesquite scalable in that prohibitive memory costs associated with making a copy of a large application mesh can be avoided by dividing the mesh into patches on which to perform the optimization sequentially.

3.6.1 MeshSet Interactions with Application Meshes

The MeshSet class is responsible for gathering information from the application's mesh and geometry and placing this information into PatchData. Because Mesquite is designed as a library to work on a broad assortment of mesh and element types on complex geometrical domains, a general, data-structure neutral API is needed. In general, Mesquite requires access to basic information about the mesh such as the number of vertices and elements in the mesh, the vertex locations, and the element connectivities. To move the vertex locations, Mesquite needs to set the vertex coordinate positions, and eventually, to perform swapping operations. Mesquite will need to add and delete various mesh entities. In addition, for smoothing meshes on complex surfaces, access to operations on the underlying solid model such as normal information and closest point information are required to ensure vertices are constrained to the surface.

To allow an application to expose this information to Mesquite, we have defined a set of interfaces (C++ abstract base classes) that are specifically designed for mesh quality improvement needs. There are four such interfaces: Mesh, VertexIterator, ElementIterator, and MeshDomain.

- Mesh: The Mesh interface represents the set of mesh entities that are to be operated on. It is through this interface that one retrieves information about the mesh and its entities. Examples of functionality provided by this interface include: retrieving the number of elements in the mesh, determining which elements contain a particular vertex, and modifying vertex coordinates.

- VertexIterator: The VertexIterator provides access to each vertex in a mesh. A VertexIterator is obtained from a Mesh object, and is used to iterate through the list of all vertices in the Mesh from which it was obtained.

- ElementIterator: The ElementIterator provides access to each element in a mesh. Other than the type of entity it exposes, it is identical to the VertexIterator.

- MeshDomain: The MeshDomain represents the set of geometric domains to which the mesh may be constrained. The MeshDomain interface enables an application to restrict the locations to which a vertex can be moved, such as constraining a vertex to a surface. Through the MeshDomain interface, Mesquite's algorithms can also obtain a domain's normal vector, which aids validity checking and decision making during the quality improvement process.

These interfaces are data-structure neutral and use only primitive data types; an application may implement the Mesquite interfaces without changing its existing mesh data structures. Instead of representing mesh entities with complex data structures or with typed pointers, entities are identified with opaque values called handles. Each mesh entity has a unique handle value, but otherwise handles have no intrinsic meaning to Mesquite.

Mesquite can also use the mesh interfaces currently being developed through the TSTT center. This interface definition effort focuses on providing access to information pertaining to low level mesh objects such as vertices, edges, faces, and regions through both array-based and iterator-based mechanisms. It is designed to support existing packages such as CBIT, NWGrid, PAOMD, and Overture. Considerations such as data neutrality, language interoperability (achieved through use of the SIDL/Babel tools from LLNL [25]), and achieving consensus within a large group of participants is paramount (see [17], [18], [26], and [19]). This interface definition effort is evolving, and the Mesquite team is actively participating to ensure that our needs for mesh quality improvement are adequately and efficiently addressed. A TSTT-based implementation of the Mesquite interfaces will be available soon. As such, any tool that exposes its mesh through the TSTT interfaces can be used with Mesquite without additional development.

The Mesquite-specific interfaces are fully compatible with the current TSTT mesh and geometry interfaces, and in fact, Mesquite's approach to data structure neutrality is directly derived from the TSTT interfaces. Although similar in spirit to the TSTT interface, the Mesquite-specific interface is not as general, and therefore consist of fewer functions and does not require additional tools such as Babel.
3.6.2 PatchData Interactions with QualityImprovers

Quality improvers are written to relocate nodes or modify topology within a PatchData, without any need to know whether the PatchData corresponds to the whole mesh or a subset of it. The PatchData information is generated by the MeshSet class with the get_next_patch function — the equivalent of an iterator over a series of patches covering the mesh. The user can set Mesquite to use different types of PatchData, ranging from a patch of elements containing one particular vertex to a patch of vertices connected to a central vertex through edges or a unique patch that covers the whole MeshSet.

PatchData and its associated classes (e.g. PatchDataVerticesMemento) provide much functionality to the optimization algorithms. Memento patterns [21] can remember the state of a PatchData geometry or topology at a given iteration and restore the PatchData to that state later. Simple functions can move the PatchData n vertices in a direction \( d \in \mathbb{R}^n \) while constraining the boundary vertices to their geometrical surface.

3.7 The Instruction Queue

The InstructionQueue class allows a sequence of operations such as quality assessment and quality improvement to be performed on a MeshSet object. The InstructionQueue provides a convenient framework to shield the user from the algorithm syntax and to ensure a consistent use of the Mesquite capabilities. One or more quality improvers can be associated with an InstructionQueue, but one must be designated as the master quality improver that determines the ultimate improvement goal. All progress made by Mesquite will be measured against the quality metrics set in the master quality improver. To improve the effectiveness and efficiency of the mesh quality improvement process, several quality improvers can be used as 'pre-conditioners' for the master quality improver. For example, a user may precede an optimization-based master quality improver with a mesh untangler and/or Laplacian smoothing.

Some predefined InstructionQueue objects, called wrappers, are available for high-level or novice users. Those typically consist of a quality assessor, followed by a mesh pre-conditioner such as an untangler, followed by a master quality improver, and finally another quality assessor. Once an InstructionQueue has been defined, a single call to run_instructions will perform all the contained operations. We note that once an InstructionQueue has been defined, it can be used for several MeshSet objects.

3.8 Code Testing Framework

Another integral part of the Mesquite framework is the code testing infrastructure. Several testing methodologies have been included within Mesquite. Unit testing is extensively used to facilitate development and ensure low-level robustness. Functional testing is used to ensure that user case scenarios run smoothly.

We use a broad definition for unit tests in Mesquite. Any test performed on a class without need for the entire Mesquite framework is considered a unit test. This encompasses simple assertions like checking the result of the multiplication of a matrix \( A \in \mathbb{R}^{m \times n} \) by a vector \( v \in \mathbb{R}^n \) to more complex assertions such as checking that a concrete QualityImprover correctly repositions a free vertex in a simple patch for a given objective function and quality metric. Mesquite uses a readily available testing framework called CppUnit [27] — essentially the well known jUnit testing framework ported to C++.

Applications using Mesquite may also find the testing framework useful when verifying their additions to the code. In particular, applications that prefer to implement Mesquite’s mesh interface instead of using the TSST mesh interface can heck their implementation with the corresponding unit test collection available in Mesquite. In addition, analytic gradient and Hessian implementations can be checked against the numerical version provided by the QualityMetric base class using readily available unit tests.

In addition to unit tests, the Mesquite test suite also includes a range of functional tests. While these tests also use the CppUnit framework, they differ from unit tests in that functional tests require the entire Mesquite framework. The functional tests are complete and often complex mesh optimization problems. These tests are intended to ensure that the individual units of the Mesquite code work together correctly. Performing these tests not only helps validate the code but also allows developers to evaluate the effects of code modifications in terms of Mesquite's accuracy and efficiency on 'real world' problems.

4. USER PROGRAMMING INTERFACES

We provide a number of mechanisms that lower the "expertise" barrier for using Mesquite. In the simplest usage scenario, a TSST mesh pointer is passed to Mesquite and improved using a default set of metrics and strategies determined by the mesh type (Section 4.2). If desired, the user can guide the mesh improvement process by setting a few parameter values indicating preferred metrics and strategies from a list of provided functionalities. Once these parameters are defined, Mesquite optimizes the mesh without further input from the user. Additional interfaces are pro-
vided for those who desire advanced functionalities, user-defined metrics or objective functions, or more control over the optimization process (Section 4.1).

In sections 4.1 and 4.2, we assume that a MeshSet has already been instantiated and a TSTT mesh handle given to it to populate the database as shown below:

```c
// Create a TSTT mesh
TSTT::Mesh wing = TSTT::Mesh::create();
wing.load("wing_file.xxx");

// Create a Mesquite MeshSet
Mesquite::MeshSet mesh_set;
mesh_set.add_mesh(wing);
```

### 4.1 Low-level API

Extensive control over the mesh quality improvement process is provided through the direct use of the Mesquite classes. In particular, the user can specify the quality metric, objective function template, and optimization algorithm by instantiating particular instances of each. For each, various options such as numerical or analytical gradient and Hessian evaluations or the patch size can be selected. Furthermore, the user can fine-tune the optimization algorithm performance by creating and setting the parameters of the termination criterion for both inner and outer iterations.

Once these core objects have been created and customized, the user creates an instruction queue and adds one or more quality improvers and quality assessors. The mesh optimization process is initiated with the `run_instructions` method on the instruction queue class. For example, in the code given below, a mean ratio quality metric is created and analytical gradient and Hessian computations are selected. The mean ratio quality metric is provided as input to the objective function template constructor, which will also use analytical gradient computations. The feasible Newton vertex mover is created and the patch size is set to be the global mesh. A termination criterion that checks the value of the L2 norm of the gradient of the objective function is added to the quality improver. Finally, an instruction queue that checks the quality of the mesh both before and after the feasible Newton algorithm runs is created and run on the mesh_set.

```c
// creates a mean ratio quality metric ...
ShapeQM* m_ratio = MeanRatioQM::create_new();
m_ratio->set_gradient_type(ANALYTIC_GRADIENT);
m_ratio->set_hessian_type(ANALYTIC_HESSIAN);

// sets the objective function template
LPtoPTemplate obj_func(m_ratio, 2);
```

```c
// creates the optimization procedures
FeasibleNewton f_newton(&obj_func);

// performs optimization globally
f_newton->set_patch_type(GLOBAL_PATCH);

// creates a termination criterion and
// add it to the optimization procedure
// outer loop: default behavior: 1 iteration
// inner loop: stop if gradient norm < eps
TerminationCriterion tc_inner;
tc_inner.addCriterion_type(GradL2Norm, e-4);
f_newton->set_inner_terminate_crit(&tc_inner);
```

```c
// creates a quality assessor
QualityAssessor m_ratio_qa(&m_ratio, AVERAGE);

// creates an instruction queue
InstructionQueue queue1;
queue1.addQualityAssesor(&m_ratio_qa);
queue1.setMasterQualityImprover(&f_newton);
queue1.addQualityAssesor(&m_ratio_qa);

// launches optimization on the mesh_set
queue1.runInstructions(mesh_set);
```

### 4.2 High-level API — Wrappers

To improve meshes with a minimum number of Mesquite function calls, we have provided a set of wrapper classes that encapsulate the most commonly used combination of quality metrics, improvement algorithms and stopping criterion. The wrappers inherit from the `InstructionQueue` class and set the algorithms used in their constructors. Using these wrappers, only two lines of code are required to improve a MeshSet.

For example, the `ShapeImprovementWrapper` shown below performs operations similar to the low-level code given in section 4.1, but without providing access to low-level features such as setting a specific termination criterion. This wrapper first uses the mean ratio quality metric to assess and report the quality of the mesh elements. It then untangles the mesh if necessary and improves the mesh with the feasible Newton algorithm applied to the composition of the mean ratio quality metric with the L2 objective function. Finally, the mesh quality is assessed and reported again.

```c
// Creates wrapper and improves mesh
Mesquite::ShapeImprovementWrapper shape_improver;
shape_improver.runInstructions(mesh_set);
```
Figure 2: (a) A ‘zoomed-in’ picture of a hybrid triangular and quadrilateral mesh. (b) A tetrahedral mesh. (c) A hybrid tetrahedral and hexahedral mesh. (d) A tetrahedral mesh of a tire incinerator. Meshes (a), (b), and (c) were generated with CUBIT (see [17]).

5. EXAMPLES

Although currently in a pre-release state, Mesquite has been used to improve a variety of mesh types. As mentioned previously, Mesquite is designed to efficiently handle a mesh of any size and element type. While not all element types are currently implemented, the code can handle triangular, quadrilateral, tetrahedral, and hexahedral elements, as well as hybrid combinations of those types. Examples of optimizations on each of these mesh types are given in Table 2. The selected meshes also represent a wide range of initial mesh qualities with two of the initial meshes containing invalid elements. Each mesh was optimized using the same objective functions and algorithms. They were first untangled (a process which has no effect on the initially valid meshes), and they were then optimized with respect to the mean ratio metric using the conjugate gradient algorithm and an \(\ell_2\) objective function template. The table shows the average mean ratio metric value of both the initial and the optimized meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Vertices</th>
<th>Init. (\mu_{avg})</th>
<th>Final (\mu_{avg})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tri.</td>
<td>10360</td>
<td>1.036781</td>
<td>1.036357</td>
</tr>
<tr>
<td>Quad.</td>
<td>267</td>
<td>(\infty)</td>
<td>2.485082</td>
</tr>
<tr>
<td>Hybrid 2D</td>
<td>263</td>
<td>1.164410</td>
<td>1.103776</td>
</tr>
<tr>
<td>Tet.</td>
<td>21156</td>
<td>1.168058</td>
<td>1.119870</td>
</tr>
<tr>
<td>Hex.</td>
<td>12753</td>
<td>3.497623</td>
<td>3.096999</td>
</tr>
<tr>
<td>Hybrid 3D</td>
<td>9200</td>
<td>(\infty)</td>
<td>1.098384</td>
</tr>
</tbody>
</table>

Table 2: Meshes with different element types optimized using Mesquite. The average mean ratio, \(\mu_{avg}\), is given for the initial and the final mesh. The hybrid 2-dimensional mesh, the tetrahedral mesh, and the hybrid 3-dimensional mesh are shown in Figures 2 (a), (b), and (c), respectively. The other meshes are not shown.

Figures 2 (b) and (d) show two tetrahedral meshes. The first mesh, given in Figure 2 (b) and Table 2, is a wedge-shaped block containing 112,393 tetrahedral elements and 21,156 vertices. The second mesh, given in Figure 2 (b), contains 11,098 elements and 2,530 vertices and is the mesh of a tire incinerator. These two meshes were optimized using three different quality improvement algorithms and objective function templates. Each optimization was based on the mean ratio quality metric. The quality of the resulting meshes is given in Table 3.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Algorithm</th>
<th>Func.</th>
<th>max. (\mu)</th>
<th>avg. (\mu)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tire</td>
<td>Initial Mesh</td>
<td>-</td>
<td>22.6413</td>
<td>1.30334</td>
</tr>
<tr>
<td></td>
<td>Active Set</td>
<td>(\ell_\infty)</td>
<td>6.11002</td>
<td>1.37703</td>
</tr>
<tr>
<td></td>
<td>Conj. Grad.</td>
<td>(\ell_1)</td>
<td>6.11002</td>
<td>1.25832</td>
</tr>
<tr>
<td></td>
<td>Feas. Newton</td>
<td>(\ell_2)</td>
<td>6.11002</td>
<td>1.29132</td>
</tr>
<tr>
<td>Wedge</td>
<td>Initial Mesh</td>
<td>-</td>
<td>4.19031</td>
<td>1.16800</td>
</tr>
<tr>
<td></td>
<td>Active Set</td>
<td>(\ell_\infty)</td>
<td>1.54073</td>
<td>1.18224</td>
</tr>
<tr>
<td></td>
<td>Conj. Grad.</td>
<td>(\ell_1)</td>
<td>2.26029</td>
<td>1.19377</td>
</tr>
<tr>
<td></td>
<td>Feas. Newton</td>
<td>(\ell_2)</td>
<td>2.04328</td>
<td>1.19500</td>
</tr>
</tbody>
</table>

Table 3: Metric values for tetrahedral meshes of a tire incinerator (Tire) and a wedge-shaped block (Wedge) optimized using three different quality improvement algorithms and objective functions. Each objective function uses the element-based mean ratio metric, denoted as \(\mu\). Tire is shown in Figure 2 (d), and Wedge is shown in Figure 2 (b).

Mesquite has also been used to optimize a geodesic mesh provide by the climate group at Colorado State University. This mesh is a dense, triangular element mesh called a twisted icosahedron grid. It is formed by refining the triangles of an icosahedron and projecting the new vertices to the unit sphere. This meshing scheme produces a relatively regular mesh with slight variations in the elements’ shapes and sizes. Mesquite was therefore used to decrease the magnitude of those variations to produce a smoother mesh in terms of both element areas and edge lengths.
6. SUMMARY AND FUTURE WORK

The Mesquite design evolved to address the high level goals of flexibility, comprehensiveness, efficiency, and interoperability through a careful balancing of upper-level C++ classes and a low-level optimizeable kernel. Years of prior experience with a variety of mesh quality improvement problems guided the design so that, although the design is still evolving (as is the mathematical framework which supports it), we expect the addition of new capabilites to impact the design relatively little. Mesquite has progressed rapidly during the past year to include a considerable number of mesh quality metrics, objective function templates, solvers, and termination criteria. Several important development objectives remain such as the inclusion of concrete 'topology modifiers' and new quality metrics to guide the evolution of deforming meshes, anisotropic smoothing, and R-type adaptivity based on solution features or error indicators. Preliminary applications of Mesquite include the smoothing of geodesic meshes and the improvement of mesh quality for increasing Tau3P abort time. Near-term code development is driven primarily by SciDAC applications, however, we anticipate many other eventual uses.

Acknowledgments The authors wish to thank Todd Munson at Argonne National Laboratory for his contributions on the feasible Newton solver and Todd Ringler at Colorado State University for the geodesic meshes.

References


[21] Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995


BSP-ASSISTED CONSTRAINED TETRAHEDRALIZATION

Bhautik Joshi 1, 2  Sébastien Ourselin 1

1 BioMedIA Lab, CSIRO ICT Centre, Sydney, Australia.
2 University of New South Wales, Sydney, Australia.
Bhautik.Joshi@csiro.au, Sebastien.Ourselin@csiro.au

ABSTRACT

In this paper we tackle the problem of tetrahedralization by breaking non-convex polyhedra into convex subpolyhedra, tetrahedralizing these convex subpolyhedra and merging them together. We generate a Binary Space Partition (BSP) tree from the triangular faces of a polyhedron and use this to identify the convex subpolyhedra in the polyhedron. Each convex subpolyhedron is tetrahedralized individually. Using an original merging process, the boundaries between these subpolyhedra are joined and tetrahedralized, ensuring that no tetrahedra are created outside of the original polyhedron in this merging process.

Keywords: computational geometry, mesh generation, tetrahedralization, BSP, Delaunay

1. INTRODUCTION

Closed polyhedra in 3D can be described using one or more non-self-intersecting, closed boundaries. The boundaries themselves are often constructed using triangular faces. These polyhedra have an interior and an exterior, and have a finite volume. Tetrahedralization of them involves finding a set of tetrahedra that completely fill the polyhedron and that lie exactly on or inside of its boundaries.

Generally, this tetrahedralization can be performed using a Delaunay tetrahedralization algorithm. Given a set of vertices in a polyhedron, the Delaunay tetrahedralization algorithm creates a convex set of tetrahedra with these vertices. None of the tetrahedra intersect each other, and the minimum angle (between edges or faces) in the mesh is maximised.

However, a Delaunay triangulation or tetrahedralization algorithm generates only convex meshes. This means that it can fail to recover the boundary of the polyhedron due to local non-convex regions, such as dents and holes, which tend to get meshed over and sealed. A generic approach to this problem involves first generating a Delaunay tetrahedralisation of the entire domain and removing tetrahedra that lie outside it [1]. However, this does not guarantee that all the boundary faces can be recovered.

To partially solve this, tetrahedralization of a non-convex polyhedron typically involves the placement of extra points on and inside it. These extra points, commonly called Steiner points, enable the algorithm to completely cover the domain at the added cost of extra complexity in the generated mesh. In 2D, several reliable methods for preserving the boundary with minimal point insertion have been proposed [1]. However, in 3D, an algorithm that can create a tetrahedralization preserving boundaries and with an acceptable degree of extra complexity is still a current and challenging problem.

Over the past ten years, two dominant approaches to preserve boundaries in 3D meshing have emerged: conforming and constrained tetrahedralizations. In the following, we describe two major techniques then briefly review some of the other methods available in the literature 1.

Conforming Delaunay tetrahedralization is a strict Delaunay tetrahedralization of the input polyhedron. If necessary, Steiner points are inserted into the mesh to ensure that all boundaries of the input mesh are preserved, while complying strictly with the Delaunay empty sphere criteria. In 2D, a clear upper bound on the number of added points was derived.

1 We recommend to the reader an exhaustive survey of tetrahedralization methods by Bern et al. [2].
Steiner points to perform a conforming triangulation has been established as $O(n^3)$ for $n$ input vertices [3]. However, conforming tetrahedralization can come at a higher cost in terms of number of Steiner points inserted.

**Figure 1:** In this constrained Delaunay triangulation, the boundary edges are indicated by the darkened lines. To preserve boundary edges, the highlighted triangle with its circumcircle (dashed lines) is not Delaunay.

Constrained Delaunay tetrahedralizations are not strictly Delaunay. To assist boundary recovery, the empty circle theorem is relaxed to allow for elements that are locally Delaunay [4]. Elements that lie on a boundary edge or face are allowed to be created even if they violate the empty circle theorem (see Figure 1). Constrained tetrahedralizations have fewer restrictions on the input meshes. Although they are not a strict Delaunay tetrahedralization, they still help to generate good quality meshes and require insertion of fewer points than conforming algorithms [5, 6, 7].

An extension of constrained Delaunay tetrahedralizations has been proposed by Shewchuk, that is able to tetrahedralize non-convex polyhedra by using a Delaunay refinement approach [8]. However, it is restricted to working with meshes whose faces form angles of greater than 50 degrees with each other.

The first group to create an algorithm that was able to perform a conforming Delaunay tetrahedralization without any restrictions on the input mesh were Murphy et al. [9]. The method described extended a 2D conforming triangulation method to 3D [10]. They stated that the number of Steiner points it added to the mesh was too large to be practical. The paper proved that there does exist an upper bound on the number of Steiner points needed to tetrahedralize a polyhedron, but the value of this bound is yet to be determined [9, 11].

By adapting the algorithm to the local features of the geometry being tetrahedralized, Cohen-Steiner et al. were able to reduce the number of Steiner points added for a 3D conforming Delaunay triangulation [11]. The algorithm makes use of the Delaunay refinement approach, which adds Steiner points to the mesh until the original boundaries have been recovered. In practice, the ratio of number of added vertices to input vertices varied between 3 to 1 and 10 to 1. However the technique did not state any bounds on this ratio.

Another approach performs tetrahedralization by using the existing Delaunay triangles in a boundary [12]. It made use of the triangles to accelerate a Delaunay tetrahedralization algorithm; however, it did not indicate if it was able to cope with non-convex polyhedra.

By partitioning a polyhedron into subpolyhedra, the problem of tetrahedralization is simplified as it allows for individually tetrahedralizing each subpolyhedron and merging the results together. It has been shown that the task of partitioning a polyhedron into the minimum number of convex subpolyhedra without Steiner points is NP-complete. However, good algorithms for polygon partitioning exist [13], with many more that allow for Steiner point insertion [14].

A common approach for polyhedral decomposition is to use an pre-defined grid, often made up of orthogonal planes to divide the polyhedron into finite sized cells. However, these techniques can be limited by the local geometry of the polyhedron [15], especially when complex features are smaller than the size of the cell size.

To tackle this problem and take advantage of the simplicity of grid-based mesh generation, we propose using a Binary Space Partition (BSP) tree to decompose an input mesh into convex regions. Each of these regions are individually tetrahedralized and then merged together with additional tetrahedra to reconstruct the original polyhedron. This technique deals effectively with non-convex polyhedra, and unlike other grid-based techniques, is not
limited by grid resolution because the grid conforms to the faces of the original polyhedron itself.

The proposed algorithm serves as an initialisation for quality mesh generation. Once it creates an efficient covering tetrahedralisation of the polyhedron, the generated tetrahedra can be easily subdivided without the need for edge flips, providing a guarantee that the input boundary topology is preserved.

The algorithm described can triangulate non-convex polyhedra in 2D as well; however, this paper is focused on its 3D applications. Our proposed algorithm is described in detail in Section 2, with preliminary results using non-convex polygons presented in Section 3. In Section 4, we discuss advantages and limitations of the current technique, and finally propose several paths for future research.

2. METHODOLOGY

Given an input polygon in 2D or 3D, a BSP tree of it can be created. Using this tree, convex subpolyhedra within the polyhedron are established [16]. Sets of points in each of these convex subpolyhedra are identified and tetrahedralised using an incremental Delaunay algorithm. Using the BSP tree, these convex sites are glued together using an adaptation of the same incremental Delaunay algorithm. The BSP tree can be traversed recursively to efficiently perform this gluing process with only two subpolyhedra at a time. An additional step to correct tetrahedra that are not coincident with each other is necessary to complete the tetrahedralisation.

A flowchart of the BSP assisted tetrahedralisation algorithm is shown in Figure 2. For clarity, convex decomposition and tetrahedralisation of convex subpolyhedra are shown to be grouped as a single step, and the gluing and fixing of the subpolyhedra as another. In practice, all four operations can be performed as part of a single recursive algorithm.

2.1 Generating BSP Trees

BSP trees can be regarded as the most general spatial subdivision technique, easily adapted for 2D, 3D and higher dimensions. BSP trees came to the fore in computational geometry and computer graphics as a solution for the painters algorithm [17]. We will not go into the details of BSP tree creation here, and recommend a good introduction to BSP trees from Bruce Naylor [18].

Using BSP trees to find convex subpolyhedra of polyhedra is a well known property of BSP trees [18, 16]. However, because BSP trees split faces and insert points, it is not a strict convex decomposition of the polygon. Ruppert and Sidiö have shown that the problem of determining if a given polygon can be tetrahedralized without Steiner points is NP-complete [19]. Other spatial subdivision techniques such as kD-trees or octrees are not used because in polyhedra with boundaries that have high curvature, a high degree of subdivision of the initial grid may be necessary, producing a large number of added points [20]. In addition, as mentioned in the introduction, unlike other spatial subdivision techniques, this decomposition is not limited by grid resolution because the grid is conforming to the faces of the original polyhedron itself.

2.2 BSP Trees for Convex Decomposition

The polygonal face at the root of a BSP tree divides space into two subpolyhedra. As one traverses the BSP tree in the front or behind subpolyhedra, these subpolyhedra are divided again by the faces of the leaf nodes. These leaf nodes may serve as roots of sub-trees which further divide the subpolyhedra, which may go on to have leaves that split these subpolyhedra and so on. In fact, convex subpolyhedra in the polyhedron are the intersection of groups of these subpolyhedra, and a structured traversal of the BSP tree can be used to identify them.

To clarify the process of convex decomposition using BSP trees we present in Figure 3 a 2D example. The polygon in Figure 3a has five faces. Each face has an orientation;
the side with the arrow represents front and without represents behind. These faces are added one at a time into a BSP tree, shown in Figure 3b. In the tree, relative to a face, the right leaf is the front halfspace and the left leaf is the behind halfspace. If the plane that a face lies in intersects a leaf, then the leaf face is split. Hence face 2 splits face 5 into 5f and 5b. As the tree is traversed, a list of tuples representing the face traversed and the direction taken relative to the root face is kept. If a front face is encountered that is empty, then the list is stored as a convex subpolyhedron. The convex subpolyhedron A is represented by the intersection of the halfspaces in front of 2, 1 and 5f. B is represented by the intersection of the halfspaces in front of 3, 4, 5b and behind 2.

2.2.1 Creation of BSP points

Not all of the implicit vertices of the subpolyhedra identified by the BSP tree convex decomposition algorithm are guaranteed to have a corresponding vertex in the original polyhedron. However, during the BSP tree creation process, faces that lie across the plane of a parent node are split. This splitting process introduces new points on to the boundary of the polyhedron being tetrahedralized. These BSP points help ensure that a complete tetrahedralization is performed. However, it is still unclear if there is a relationship between BSP points and Steiner points.

2.3 Tetrahedralization of subpolyhedra

Once a convex subpolyhedron has been identified, a list of points inside it can be determined. These points can be tetrahedralized, forming a mesh of the convex subpolyhedron. A randomized point insertion Delaunay tetrahedralization algorithm is used. However, it is not guaranteed that this set of points completely describe the boundary of the convex subpolyhedron. This commonly occurs when a node of a BSP tree intersects the plane of a child node but does not intersect the child node itself.

A 2D example of subpolyhedra that are not completely described is shown in Figure 4. The polygon in Figure 4a and one possible corresponding BSP tree, Figure 4b, can be split into two convex subpolyhedra, A and B. Although the plane that the child node 3 lies in intersects the parent node, 1, the face itself does not intersect the parent node. This means that for the convex subpolyhedra, there is no vertex at the intersection of bounding planes 1 and 3, and hence the subpolyhedra do not have a vertex at every corner of the subpolyhedra. Figure 4c illustrates how a triangulation of the subpolyhedra will not recover the whole polygon.

Uncorrected, this problem will lead to missing tetrahedra in the tetrahedralization. To recover these missing tetrahedra, a glue algorithm has been devised to merge subpolyhedra together.

2.3.1 Pseudocode Implementation

The tetrahedralization algorithm operates by traversing the BSP tree. A list of tuples representing the node traversed and the direction taken - for example, (node_n,[front,behind]) - is necessary to identify the convex subpolyhedra. Once a node has been visited, it is removed from the list. This pseudocode implementation is described in Algorithm 1.

2.4 Gluing subpolyhedra

When two subpolyhedra are merged using the glue algorithm, they are separated exactly by a single splitting plane, which is the root node of the current position in the BSP tree. As such, the algorithm recursively merges two growing subpolyhedra at a time until all subpolyhedra in the tree have been merged. Tetrahedra need to be generated that fill the space in-between the merging subpolyhedra. This is not a new problem; Bern and Marshall have demonstrated that it is possible to tetrahedralize the region between two convex polyhedra without the addition of Steiner points [21].

One way to generate the tetrahedra in-between the subpolyhedra is to merge the points in both the front and behind subsets and perform a Delaunay tetrahedralization on this merged set of points. A simple cross test then can be used to reject tetrahedra that do not merge the tetrahedralization. This merging set of tetrahedra is
then added to the set of already generated tetrahedra for the front and behind subpolyhedra.

For a tetrahedra to pass the cross test, it must satisfy three criteria. Firstly, all its vertices must not lie completely on and above or on and below the joining plane. Secondly, at least one edge of the tetrahedra must cross the joining plane. Lastly, none of the edges of the tetrahedra may intersect an existing face on the joining plane.

A 2D example is described in Figure 5. In this merge, the joining plane is represented by the dashed line and is generated from the edge \( cf \). Using the cross test, triangle \( \triangle de \) is legal, as its vertices lie both above and below the joining plane, an edge crosses the joining plane and none of the edges intersect \( cf \). \( \triangle afb \) is illegal because all its points above the joining plane, and \( \triangle abg \) is illegal because it has edges that cross edges on the joining plane \( cf \).

Unfortunately, the above algorithm does not work for all cases, sometimes failing to completely mesh the region in-between the subpolyhedra. To solve this problem we propose a glue algorithm inspired by the Bowyer-Watson randomized point insertion (RPI) algorithm [22, 23]. It works by joining triangles from the boundary (or hull) of the front subset, front hull, to the points of the hull below that face those triangles, below hull points.

Tetrahedralization of the front and behind subpolyhedra are performed independently of each other. Because of this, there is no guarantee that a point in one set would not violate the Delaunay empty sphere criteria of a tetrahedra in the other set. Hence the region in-between the two sets would not be guaranteed to be tetrahedralized with Delaunay tetrahedra. Therefore a non-Delaunay tetrahedralization algorithm is necessary. A version of the traditional RPI algorithm has been modified so that it performs the gluing tetrahedralization without check-

---

**Algorithm 1 Traverse and tetrahedralize convex subpolyhedra within a BSP tree**

1: tetrahedralize(bsp_tree, traversal_list)
2: frontNode = bsp_tree.front
3: behindNode = bsp_tree.behind
4: if frontNode = empty then
5:   points = getConvexPoints(bsp_tree, traversal_list)
6:   return Delaunay(points)
7: end if
8: push (frontNode, front) on traversalList
9: front tetrahedralization = bsp tetrahedralize(frontNode, traversal_list)
10: pop traversalList
11: push (behindNode, front) on traversalList
12: behind tetrahedralization = bsp tetrahedralize(behindNode, traversal_list)
13: pop traversalList
14: return glue(front tetrahedralization, behind tetrahedralization)

---

**Figure 5:** Merging two convex subpolyhedra.

**Figure 6:** Randomized Point Insertion: adding a point outside the triangulation when the point is inside the circumcircles of one or more existing triangles is shown in (a). (b) and (c) demonstrate two alternative solutions for triangulation.
In RPI, a point may be added to a tetrahedralization that lies outside the tetrahedra but infringes upon the circ-
circles of some tetrahedra. Conventionally, the tetra-
hdra that are infringed are deleted and re-meshed and the
remaining edges for which the point is behind make new
tetrahedra. With the new point insertion algorithm, in-
fringed tetrahedra are not deleted.

A 2D example of this type of point insertion is shown
in Figure 6. Triangles whose circumcircles contain the
point are deleted and re-meshed with the new point (Fig-
ure 6a). Once this is completed, the edges of the hull are
used to create new triangles (same as if the point was
lying completely outside the triangulation and circum-
circles)(Figure 6b). With the glue algorithm, as shown
in Figure 6c, no tetrahedra are deleted and triangles are
created from edges on the hull and the exterior point.
The glue algorithm operates in a similar fashion, but the
tetrahedra that are infringed are not deleted and tetra-
hedra are created from any faces for which the point lies
behind.

The below hull points are added individually to the front
hull using this technique. An example is shown in Fig-
ure 7. The half-space above the line is indicated simply as
above, below as below. The boundary edges in above
are identified and the points on the boundary of below
are also identified, as described in Figure 7a. For each
of the boundary edges in above, a test triangle for each
boundary point in below.

For the triangle to be allowed, it must not intersect any
edges on the joining plane or intersect any of the bound-
ary edges in below. So, \( \triangle abc \) is allowed, but \( \triangle abd \)
is not because one or more of its edges intersect the below
boundary. Once a legal triangle is created, it is added to
the above set and further edges may attempt to create
triangles with the added point as in Figure 7b. When
all the points are added to the above subpolyhedron, the
region between above and below is triangulated as deline-
ated as the darkened area in Figure 7c.

### 2.5 Fixing Crossed Tetrahedra Edges

While the glue algorithm completely fills the space be-
tween the above and below subpolyhedra, it is not guar-
anteed that the gluing process creates tetrahedra that
have edges that are all coincident to each other, leading
to edges that may intersect. These intersections can pos-
sibly occur on any of the planes defined by the faces of the
below hull.

![Figure 8: The addition of a point to the edge of a tetra-
hedron on the left leads to the creation of two new tetra-
hedra, shown separated, on the right.](image)

This can usually be corrected by inserting a point at the
site of an intersection and splitting the tetrahedron for
which this added point lies on. Each point added to the
dge of a tetrahedron leads to the creation of two new
tetrahedra, as described in Figure 8.

A whole face of a tetrahedron may lie on a plane for which
there are edges that are crossed. A case can occur where
adding the intersected point generates an edge which in
turn generates a new intersection, leading to an infinite
loop of new intersection points being generated. This
can be avoided by checking to see if the intersected point
generates an edge which generates more invalid crossed
edges. If this is the case, the intersected point is pushed
to the end of the list of intersected points to be added.
In this fashion, all the intersected points can be added.

Figure 9 describes an example of this process. Two tetra-
hedra have triangular faces that lie on a common plane.
Their vertices and edges are not coincident and they lie
across each other as shown in Figure 9a, creating four in-
Intersection points - $p,q,r,s$. These intersection points are pushed onto a list. $p$ is the first point to be added to the tetrahedra, and does not create any new intersections, as shown in Figure 9b. However, when $q$ is popped off the top of the list it creates edges that create new intersection points, as shown in Figure 9c. $q$ is placed at the bottom of the list to be handled later. The same applies to $r$. $s$ is now popped off the top list, and its addition does not create any new intersecting edges, as shown in Figure 9d. $q$ and $r$ are (respectively) popped of the list now, and can safely split the tetrahedra without creating any new intersected edges.

3. PROOF OF CONCEPT

To test the BSP assisted tetrahedralization algorithm, four non-convex 3D polyhedra were constructed. The first is a cube which had one corner indented, the second is the well known Schönhardt's polyhedron, the third is a cube with a hole through it on one axis and lastly a rectangular prism with opposing angular cuts on two opposite faces, making a dented polygon.

These polyhedra are closed surface meshes constructed from triangular faces. Our BSP assisted tetrahedralization algorithm has been applied to this set. During this process, we recorded several pertinent variables, such as the initial number of points in the polyhedron, the number of points added, and the final number of tetrahedra. We summarise these results in Table 1. In Table 2, we present the different polyhedra at each step of the tetrahedralization process.

4. DISCUSSION

4.1 Theoretical Bounds

Current algorithms for tetrahedralizations produce a large number of Steiner points that need to be added. In the two dimensional case, a lower bound of $O(n^3)$ has been found [3]. However, a lower bound for the number of inserted Steiner points in 3D has yet to be firmly established [11]. One could reasonably expect that 3D techniques that extend 2D would only increase the lower bound (i.e., greater than $O(n^3)$).

Given a 3D polyhedra made of $f$ triangular faces and $n$ vertices, in the worst case, a naive BSP tree generation algorithm will produce a tree made from $O(f^3)$ faces. In practice, however, empirical results have shown that BSP trees of 3D polyhedra tend to produce trees with close to $O(f \log(f))$ faces [18]. Paterson and Yao demonstrated that it is possible to construct 3D BSP trees with a provable upper bound of $O(f^2)$ and lower bound of $O(f^{3/2})$ [24]. It is expected that a BSP would produce BSP polygons with a number of vertices in the order of $O(n \log(n))$.

4.2 BSP and Steiner Points

During the process of creating a BSP tree from the faces of an input mesh, many of the faces are split. Each time a split occurs, two new points are introduced into the mesh. Rather than discard these BSP points, they are retained as part of a new mesh with no split faces as defined by the BSP tree. In practice, these points appear to fulfil the role of Steiner points, helping to ensure that convex regions can be tetrahedralized. However, a proof that links BSP points and Steiner points is yet to be established.

4.3 Practical Considerations

It is possible to construct many different BSP trees from the same input mesh. To help make the BSP trees generated more consistent, it was decided that a selection criteria for choosing root nodes in BSP tree construction was necessary. The metric used was the number of leaf nodes split if a particular root node was chosen. In practice, meshes tended to have fewer added points and tetrahedra when the choice of BSP root node was one that minimised the number of split leaf nodes.

The mesh generator does not guarantee the quality of any of the tetrahedra generated. However, the mesh generation process is not limited by the size of local geometrical features and as such is scale independent.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Dented box</th>
<th>Schönhardt's polygon</th>
<th>Box with hole</th>
<th>Dented polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Number of Points in Polygon</td>
<td>8</td>
<td>6</td>
<td>16</td>
<td>26</td>
</tr>
<tr>
<td>Total Number of Points Added</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Number of Tetrahedra</td>
<td>9</td>
<td>18</td>
<td>29</td>
<td>70</td>
</tr>
</tbody>
</table>

**Table 1:** Numerical results on non-convex polyhedra.

<table>
<thead>
<tr>
<th>Type</th>
<th>Dented box</th>
<th>Schönhardt's polygon</th>
<th>Box with hole</th>
<th>Dented polygon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td><img src="original_diagram.png" alt="Diagram" /></td>
<td><img src="schonhardt_diagram.png" alt="Diagram" /></td>
<td><img src="box_with_hole_diagram.png" alt="Diagram" /></td>
<td><img src="dented_polygon_diagram.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Before gluing</td>
<td><img src="before_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="before_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="before_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="before_gluing_diagram.png" alt="Diagram" /></td>
</tr>
<tr>
<td>After gluing</td>
<td><img src="after_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="after_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="after_gluing_diagram.png" alt="Diagram" /></td>
<td><img src="after_gluing_diagram.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Wireframe</td>
<td><img src="wireframe_diagram.png" alt="Diagram" /></td>
<td><img src="wireframe_diagram.png" alt="Diagram" /></td>
<td><img src="wireframe_diagram.png" alt="Diagram" /></td>
<td><img src="wireframe_diagram.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Table 2:** Graphical results table for non-convex polyhedra.
Because convex subpolygons are tetrahedralized individually, and only the hulls of these subpolyhedra are necessary for the bulk of the computation, completed subpolyhedra are swapped out-of-core into a cache on disk. In this way, it may be possible to tetrahedralize very large meshes by swapping out completed portions of the mesh.

As with many tetrahedralization algorithms, practical implementation is plagued by problems with computational precision.

4.4 Limitations

The weakest part of the algorithm is in the gluing process, and the author believes that it will require more development. The main problem lies in the fact that the set of generated tetrahedra between the subpolyhedra is non-Delaunay, and reliable properties and attributes of these tetrahedra have yet to be determined.

Related to this is the problem of finding the set of faces in the above subpolyhedron that face the below subpolyhedron in the gluing process. At the moment, there is no sure way to prevent faces from the other side of the subpolyhedron that are simply just facing in the right direction to be treated as candidates for gluing to the bottom subpolyhedron.

5. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have presented a covering tetrahedralization algorithm providing an efficient initialization for a quality mesh generator. As such, these tetrahedra can be subdivided and refined without any edge flips which guarantees to preserve the topology of the polyhedron input boundary.

The main innovation of this is bringing together BSP trees and Delaunay Tetrahedralization, along with a tetrahedral mesh generator that is not limited by the local complexity of the geometry.

Based on our experiments, the number of added points has been quite low. Future testing on more complex meshes will yield more conclusive information about the properties of this algorithm and how it may perform in practice.

The number of points added by a BSP decomposition of a polygon has already been quite well established. If a link between BSP points and Steiner points can be found, it may be possible that bounds on the number of extra points necessary to tetrahedralize any polyhedron can be found.

So far, the meshes generated are Constrained Delaunay Tetrahedralizations, and no consideration is given to the quality of the meshes generated. Preliminary work has shown that it is, in theory, possible to convert the constrained tetrahedralizations to conforming ones via point insertion and Delaunay refinement. This is work currently in progress.

The algorithms in this paper were implemented in C++ and made use VTK for rendering output [25]. The computational geometry functions implemented in the BSP assisted tetrahedralization algorithm will be made available online.

6. ACKNOWLEDGEMENTS

The authors would like to thank R. Li and N. Killeen for proofreading of this article, and J. Ables and B. Doyle for their support.

References


Generalization of the Kantorovich Method of Dimensional Reduction

Dr. Krishnan Suresh
Department of Mechanical Engineering
University of Wisconsin, Madison
suresh@engr.wisc.edu

ABSTRACT
Boundary value problems posed over thin solids are often amenable to a dimensional reduction in that one or more spatial dimensions may be eliminated from the governing equation. One of the popular methods of achieving dimensional reduction is the Kantorovich method, where based on certain \textit{a priori} assumptions, a lower-dimensional problem over a ‘mid-element’ is obtained. Unfortunately, the mid-element geometry is often disjoint, and sometimes ill defined, resulting in both numerical and automation problems.

A natural generalization of the mid-element representation is a skeletal representation. We propose here a generalization of the Kantorovich method that exploits the unique topologic and geometric properties of the skeletal representation. The proposed method rests on a quasi-disjoint Voronoi decomposition of a domain induced by its skeletal representation. The generality and limitations of the proposed method are discussed using the Poisson’s equation as a vehicle.

Keywords: Geometric simplification, medial axis transforms, dimensional reduction, plate theory, CAD/CAE.

1. INTRODUCTION
Engineering analysis typically entails solving boundary value problems via computational procedures such as the finite element method. When the underlying geometry is relatively thin, boundary value problems are amenable to a dimensional reduction in that one or more spatial variables may be eliminated from the governing equation, prior to a finite element discretization. This results in significant computational gains with minimal loss in accuracy [Donaghy 96]. A popular means of achieving dimensional reduction is the Kantorovich method [Kantorovich 64], [Pilkey 94], [Shames 85]. The essential aspects of the method are summarized below for the Poisson’s equation. This summary will also help identify an important limitation of the method that we address here.

1.1 The Kantorovich Method
Consider a thin rectangular domain illustrated in Figure 1-1, where \( h \ll l \). Let \( U(x,y) \) be a field that satisfies the Poisson’s equation:
\[
\nabla^2 U + k = 0 \quad \text{for} \ (x,y) \in \text{interior}
\]
\[
\text{Subject to:} \ U = 0 \quad \text{for} \ (x,y) \in \text{boundary}
\]
The problem of determining \( U(x,y) \) is two-dimensional, but since \( h \ll l \) it may be reduced to an approximate one-dimensional problem via the Kantorovich method.

![Figure 1-1: Mid-element of a rectangle.](image)

The first step in the Kantorovich method is to express the Poisson’s equation as an equivalent variational statement [Reddy 84], [Shames 85]:
\[
\text{Minimize} \ \Pi(U) = \frac{1}{2} \int_{0-h}^{h} \left( U_x^2 + U_y^2 - 2kU \right) dy dx \quad (1-1)
\]
\[\text{Subject to:} \ U = 0 \quad \text{for} \ (x,y) \in \text{boundary}\]
The next step is to seek an approximate solution \( \hat{U}(x,y) \) that satisfies the boundary conditions on the ‘dominant parallel edges’, i.e., on \( y = \pm h \). A non-trivial function satisfying this requirement is:
\[
\hat{U}(x,y) = \left[ 1 - \left( \frac{y}{h} \right)^2 \right] u(x)
\]
Higher-order polynomials in \( y \) or even trigonometric functions may be used, provided \( \hat{U}(x,\pm h) = 0 \). \( \hat{U}(x,y) \) is referred to here as a Kantorovich trial function; it defines the ‘function space’ in which a solution is being sought. Note that \( \hat{U}(x,0) = u(x) \) where \( u(x) \) is an unknown function over the line-segment \( y = 0 \). This line-segment is incidentally called a mid-element of the rectangle.
In the assumed function space, one can find \( u(x) \) by substituting the trial function in the variational statement and integrating over \( y \), i.e., eliminating \( y \). This results in:

\[
\text{Minimize } \Pi(u, u_x) = \int_0^l \left[ 16h u_x^2 + 8u^2 \frac{2}{3h} - 4kh u \right] dx
\]

Subject to: \( u = 0 \) for \( x = 0, 1 \)

Thus a 2-D variational problem has been reduced to a 1-D variational problem over the mid-element involving \( u(x) \). One can now proceed to minimize the 1-D problem using standard 1-D finite element techniques [Reddy 84], [Shames 85].

Thus the above Kantorovich method may be viewed as a two-stage approximation process as opposed to a single stage finite-element method, as illustrated in Figure 1-2 [Babuska 94].

Kantorovich trial functions

\( n \)-D problem

\( (n-1) \)-D problem over mid-element

\( (n-1) \)-D finite-element trial functions

Numerical Solution

\( n \)-D finite-element trial functions

**Figure 1-2: Mid-element dimensional reduction.**

By considering Kantorovich trial functions that completely span the solution space, a hierarchical system of solutions \( \hat{U}_n(x, y) \) that converge to the exact solution may be obtained; see hierarchical methods proposed by Babuska and others [Vogelius 81], [Babuska 94]. The two-stage approximation results in considerable computational gains since the first stage is executed once in a symbolic sense.

Various lower-dimensional theories of beams and plates are derived along similar lines. The starting point for such theories is the principle of virtual displacement, a generalization of the above variational statement, which states that for a system in static equilibrium, the work done by a virtual displacement must be zero [Shames 85]:

\[
\int_V \sigma_{ij} \delta u_{ij} dV - \int_V f_i \delta u_i dV - \int_{S_2} \Gamma_i \delta u_i dD = 0
\]

If the solid is sufficiently thin, one may assume certain trial functions for the displacements \( u_i \), and a spatial variable may be eliminated resulting in a lower dimensional problem over the mid-element [Shames 85], [Wang 00].

**2. LIMITATIONS OF THE MID-ELEMENT BASED KANTOROVICH METHOD**

We now identify a serious drawback of the mid-element based Kantorovich method. Consider the notched rectangle illustrated in Figure 2-1. For simplicity, we shall assume that a field defined over the solid satisfies, as before, the Poisson equation and zero Dirichlet conditions.

**Figure 2-1: A notched rectangle.**

Recall that, in the Kantorovich method, one must seek a trial function that satisfies the boundary conditions along the ‘dominant parallel edges’. Due to the irregularity of the solid, it is not possible to define a single analytic function over the entire domain that meets this requirement. The domain is therefore divided into 3 quasi-disjoint regions \( \Omega_1 \), \( \Omega_2 \) and \( \Omega_3 \) as illustrated in Figure 2-2. Further one can define a mid-element \( M_i \) and a thickness \( 2h_i \) with each region. The pairs \( (M_i, h_i) \) constitute the mid-element representation of the solid that unambiguously captures the geometry of the notched rectangle.

**Figure 2-2: Mid-element based decomposition.**

Observe that, due to the decomposition, the variational statement of Equation (1-1) can now be expressed as:

\[
\text{Minimize } \sum_{i=1}^{3} \int_{0}^{h_i} \left( \bar{U}_{i,x}^2 + \bar{U}_{i,y}^2 - 2k \bar{U}_i \right) dy dx
\]

where the \( x \)-axis conveniently coincides with the mid-element \( M_i \). We now define three trial functions \( \bar{U}_1, \bar{U}_2 \) and \( \bar{U}_3 \), one in each of the three domains per:

\[
\bar{U}_i(x_i, y_i) = 1 - \left( \frac{y_i}{h_i} \right)^2 u_i(x_i)
\]

Observe that \( \bar{U}_1(x_i, \pm h_i) = 0 \), and \( \bar{U}_1(x_i, 0) = u_i(x_i) \) where \( u_i(x_i) \) are unknown functions.

With these definitions in place, one can eliminate \( y_i \) as before by substituting the assumed trial functions in the above variational formulation. This results in a variational statement governing three unknown functions \( u_i \) over the 3 independent mid-elements \( M_i \).

At first glance, it appears that the above formulation is no different from that associated with a rectangle. However, this is not true ... we have now violated the admissibility criterion of a variational formulation! It is a well-established fact that in a variational formulation, whenever a domain is sub-divided, and different trial functions are defined over each sub-domain, the trial functions must satisfy an admissibility criterion [Strang 73]. The admissibility criterion states that if the variational statement involves derivatives up to order \( m \), then the trial functions must be at least \( C^{m-1} \) continuous across the boundaries of adjacent sub-domains.

In our case the variational statement involves only the first derivative of \( U(x, y) \). However, one can easily verify
that the assumed trial functions $U_i(x,y)$ are not $C^0$ continuous across the common boundaries, violating the admissibility criterion. This fact is often ignored, leading to both automation and numerical problems.

Since the assumed trial functions are not admissible from a variational standpoint, any attempt to couple the three functions $u_1, u_2$ and $u_3$, and their derivatives is necessarily ad hoc and approximate ... it does not follow from the mid-element based geometric decomposition. More importantly, since the admissibility criterion is violated, no formal claims can be made about the convergence or accuracy of the mid-element based Kantorovich method, as it applies to such solids.

Further, a mid-element based decomposition does not always exist since the mid-element representation is incomplete for a large class of solids. For example, consider a dovetail section illustrated in Figure 2-3. Since there exists no mathematical definition of a mid-element, we rely on the dimensional reduction process to yield appropriate mid-elements. This would yield the mid-elements illustrated in Figure 2-3. However, it is now impossible to assign a thickness – even a varying one – to each of the mid-elements such that the solid may be recovered, i.e., the mid-element representation is incomplete. The two problems identified above are much more pronounced and difficult to resolve in 3-D.

![Figure 2-3: Disjoint mid-elements for a dovetail](image)

### 2.1 Prior Work

The Kantorovich method (and its variations) has been extensively investigated, as it applied to uniform-thickness plates and shells. The works of Reissner, Hencky, Mindlin, Lo, Reddy and others (see references in [Reissner 85]) fall into this category, so does the modern work on hierarchical modeling [Vogelius 81], [Babuska 91], [Madureira 99].

However, focusing our attention on geometrically more complex but thin solids, Armstrong and colleagues [Armstrong 94], [Donaghy 96] were the first to propose the use of medial axis transform (defined below) to resolve some of the geometric issues associated with the mid-element representation. The medial axis transform, or skeletal representation as it is referred to in this paper, is a natural generalization of the mid-element representation, and it consists of a skeleton and a radius function, where the skeleton follows the shape of the solid, while the radius function captures the local thickness. The skeleton of the dovetail is illustrated in Figure 2-4. Observe the similarities and differences between Figure 2-3 and Figure 2-4. The most important difference is that the skeletal representation is an unambiguous and complete geometric representation, whose mathematical properties are now well understood [Choi 97], [Sherbrooke 96], and its role in engineering analysis is well documented [Tam 91], [Armstrong 95], [Price 95], [Armstrong 98], [Monaghan 98], [Sheffer 98], [Armstrong 99], [Shim 01].

![Figure 2-4: The skeleton of the dovetail](image)

### 2.2 Skeletal Representation Based Generalization of the Kantorovich Method

The method proposed here combines the Kantorovich principle of two-stage reduction with the unique topologic and geometric properties of the skeletal representation, and has three essential features.

- First is the decomposition of a solid into its S-Voronoi decomposition (see Section 3).
- Second is the definition of generalized Kantorovich trial functions defined over the decomposition. By construction, the trial functions will not only satisfy the admissibility criterion, but will also be complete and satisfy essential boundary conditions.
- Third is the elimination of one of the space variables (essentially, the thickness parameter), by appropriate mathematical transformations.

In Section 3, we review the properties of skeletal representations. In Section 4, we describe the proposed method in detail using the Poisson’s equation as a vehicle. In Section 5, numerical experiments involving Poisson problems over 2-D polygonal solids are presented. In Section 6, we propose a strategy for inclusion of singularities, and Section 7 summarizes the main contributions of the paper.

### 3. SKELETAL REPRESENTATIONS

Skeletal representations (s-reps) are characterized by two entities, namely a skeleton (or medial axis) and a radius function. The two entities are defined through the concept of a maximal ball [Sherbrooke 96]:

- A closed ball $B(p,r) \subset \mathbb{R}^n$ is the set of points $q$ such that $\|p-q\| \leq r$.
A closed ball $B(p, r) \subset \mathbb{R}^n$ is maximal with respect to $\Omega$ if it is contained in $\Omega$, but not in any other closed ball contained in $\Omega$.

- **Skeleton** of $\Omega$ is the locus of the centers of all maximal balls of $\Omega$, plus the limit points of the locus.

- **Radius function** at a point on the skeleton is the radius of associated maximal ball.

The s-rep of a 2-D L-bracket is illustrated in Figure 3-1. A mathematical analysis of s-reps can be found in [Choi 97], [Calabi 68], [Sherbrooke 96].

![Figure 3-1: S-rep of an L-bracket.](image)

We shall assume here that the skeletal representation of a 2-D solid can be computed using, for example, the techniques proposed in [Meshkat 87], [Srinivasan 87], [Ramanathan 02]. Techniques for 3-D computation of a s-rep may be found in [Sapidis 91], [Hoffman 94], [Turkiyyah 97], [Etzion 99], [Etzion 02].

### 3.1 S-Voronoi Decomposition

The theory developed in this paper is restricted to geometrically complex thin solids whose skeletal branches are of dimension 'n-1' and terminate at the boundary. Polygons and polyhedrons, for example, exhibit this property [Blum 78], [Nackman 82]. Such solids possess a convenient S-Voronoi decomposition discussed below.

On the other hand solids such as the one illustrated in Figure 3-2 are not considered here since one of the skeletal branches terminates in the interior of the domain. We expect to extend the theory to such solids in a forthcoming paper.

![Figure 3-2: Interior skeletal point.](image)

Focusing our attention on thin 2-D solids, let $(\xi_i(s), \eta_i(s))$, $1 \leq i \leq N$ be the $N$ skeletal branches of a solid, where $s$ is the arc length parameter ranging from 0 to $l_i$, the length of the skeletal branch. Further, let $R_i(s)$, $1 \leq i \leq N$ be the associated radius functions. Given a triple $(\xi_i(s), \eta_i(s), R_i(s))$ we define the following.

Define $\alpha_i(s)$ to be the angle made by the tangent at $(\xi_i(s), \eta_i(s))$ to the global $x$ axis, i.e.,

$$
\cos \alpha_i, \sin \alpha_i = \frac{d\xi_i}{ds}, \frac{d\eta_i}{ds} = (\xi_i, \eta_i)'
$$

Further, let $\theta_i(s)$ be the angle between the tangent vector, and the vector to the nearest boundary point. One can show that [Blum 78]:

$$
\theta_i(s) = -\cos^{-1}\left(\frac{dR_i}{ds}\right)
$$

Figure 3-3 illustrates a skeletal branch that is a bisector of two boundary segments, and the definition of $\alpha$ and $\theta$.

![Figure 3-3: Geometry of a skeletal curve.](image)

One can now associate two sets $\Omega^+_i$ and $\Omega^-_i$ with each skeletal branch, respectively, of a directed skeletal branch, as illustrated in Figure 3-3. Equations (3-1) and (3-2) are transformations from $\Psi_i = \{(s, \kappa)|0 \leq s \leq l_i, 0 \leq \kappa < 1\}$ to $\Omega^+_i$.

If none of the skeletal branches terminate in the interior, then one can show that the solid can be expressed via the following S-Voronoi decomposition:

$$
\Omega = \bigcup_i \Omega^+_i + \Omega^-_i
$$

On the other hand, if there are internal terminal points for a skeleton, then the decomposition has additional terms:

$$
\Omega = \bigcup_i \Omega^+_i + \Omega^-_i + \bigcup_k \Omega^0_k
$$

We use the term S-Voronoi decomposition to distinguish it from the standard Voronoi decomposition [Srinivasan 87], the latter being a coarser version of the former. For example, Figure 3-4 illustrates the difference between the two for a rectangle.

(a) S-Voronoi decomposition

![Figure 3-4: S-Voronoi versus Voronoi decomposition.](image)

(b) Voronoi decomposition
Observe in Figure 3-4a that the S-Voronoi decomposition consists of 10 sub-domains, 2 sub-domains per skeletal branch. On the other hand, Figure 3-5b consists of 4 sub-domains, one per boundary segment.

Figure 3-5a illustrates the S-Voronoi decomposition of the dovetail consisting of 26 sub-domains. Figure 3-5b is a detailed view of Figure 3-5a about the left reentrant corner.

![Figure 3-5: S-Voronoi decomposition of dovetail.](image)

For transformations given by Equations (3-1) & (3-2), one may define standard Jacobians matrices [Bronshtein 85]. For each of the 3 types, the functions appearing in Equation (3-8) simplify significantly, as summarized in Table 3-1 below.

<table>
<thead>
<tr>
<th>Boundary Segments</th>
<th>Skeletal Branch</th>
<th>$\alpha_{i,s}$</th>
<th>$\theta_{i,s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line-line</td>
<td>Line-segment</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Point-point</td>
<td>Line-segment</td>
<td>0</td>
<td>$\sin \theta/R(s)$</td>
</tr>
<tr>
<td>Point-line</td>
<td>Parabola</td>
<td>$\sin \theta/2R(s)$</td>
<td>$\sin \theta/2R(s)$</td>
</tr>
</tbody>
</table>

**Table 3-1: Types of skeletal branches.**

Finally, one can show that the Jacobian transformation defined per Equation (3-6) is invertible, i.e., the determinant is non-zero, in the interior of a domain. Thus, we have:

$$\begin{bmatrix} U_{x,s} \\ U_{y,s} \end{bmatrix}_i = \left( J_i^\pm \right)^{-1} \begin{bmatrix} U_{x,k} \\ U_{y,k} \end{bmatrix}_i$$

**4. PROPOSED METHOD**

In this section, we propose a generalized Kantorovich method for a variational formulation, using the variational statement of the Poisson’s equation as a vehicle:

$$\text{Minimize } \Pi(U) = \frac{1}{2} \int_0^H \left( \int U_{x,s}^2 + U_{y,s}^2 - 2kU \right) dy dx$$

Subject to: $U = \tilde{u}(\Gamma)$ for $(x,y) \in \Omega$

The proposed method consists of the following steps.

**Step 1:** The first step is to compute the skeletal representation of the solid, and to express the solid as an S-Voronoi decomposition as in Equation (3-3). We assume that this can be carried out using one of technique proposed in [Meshkat 87] or [Etizon 99]. Due to the decomposition, the above minimization problem simplifies to:

$$\text{Minimize } \sum_i \left[ \Pi_i^\pm \left( U^\pm_i \right) \right]$$

where each term of the integral is of the form:

$$\Pi_i^\pm (U_i^\pm) = \frac{1}{2} \int \left[ \frac{1}{\alpha_i^\pm} \int \left( U_{x,s}^2 + U_{y,s}^2 \right) dy dx \right] - \int k U_{x,s} dy dx$$

**Step 2:** The next step is to exploit the Jacobian transformation described in Section 3, to make a variable change $(x,y) \rightarrow (s,k)$ in Equation (4-2):

$$\Pi_i^\pm (U) = \left[ \int \left( J_i^\pm \right)^{-1} U_i^\pm dy dx \right] \int J_i^\pm k U_i^\pm ds$$

We then define a set of generalized Kantorovich trial functions $U_i^\pm(s,k)$ that satisfy:
Admissibility criterion
Completeness criterion
Conformance criterion

Essential boundary conditions. These requirements are identical to the ones imposed on finite element trial functions to ensure convergence [Strang 73], and are discussed below.

Admissibility: For the Poisson problem, the admissibility criterion states that the trial functions must be at least $C^0$ continuous across the boundaries of adjacent sub-domains. In Figure 3-5, sub-domain 1 is adjacent to sub-domains 2, 3 and 4. The trial functions defined in region 1 must therefore be at least $C^0$ continuous with the trial functions in 2, 3 and 4.

Observe that there are two types of adjacency. The first type (type-1) involves regions that share a common skeletal branch; the adjacency between regions 1 and 2 is of this type. Since a skeletal branch corresponds to $k = 0$ in Equations (3-1) and (3-2) type-1 adjacency requires that

$$U^+_i(s,0) = U^+_i(s,0).$$

On the other hand, in a type-2 adjacency, neighboring regions (example: 1 and 3) do not share a common skeletal branch but a common branch-point. This corresponds to $s = 0$ or $s = l_i$ in Equations (3-1) and (3-2), leading to compatibility conditions that describe how unknown functions defined on one skeletal branch are related to unknown functions over neighboring skeletal branches.

Completeness: Next consider the completeness criterion. Recall that a polynomial $g(k)$ is complete up to order $m$, if $g(k)$ contains $k^0, k^1, ..., k^m$. We impose a similar condition with respect to the thickness variable $\kappa$ on

$$U^+_i(s,\kappa).$$

Conformance: In the posed problem, the requirement for conforming trial functions is the same as the admissibility, i.e., the trial functions must be at least $C^0$ continuous across the boundaries of adjacent domains.

Boundary Conditions: Finally, $U^+_i(s,\kappa)$ must satisfy essential boundary conditions on $\Omega^+_i$. Let $w^+_i(s)$ is the Dirichlet boundary condition on the boundary curve $\Gamma^+_i(s)$. The trial functions must therefore satisfy

$$U^+_i(s,1) = w^+_i(s).$$

**Trial Functions of type $C^0$:** Consider the following trial functions, where $\{u_i(s)\}$ are unknown functions:

$$U^+_i(s,\kappa) = \begin{cases} u_i(s)(1-\kappa) + \hat{w}_i(s)\kappa^2 \end{cases} \text{ in } \Omega^+_i$$

$$U^-_i(s,\kappa) = \begin{cases} u_i(s)(1-\kappa) + \hat{w}_i(s)\kappa^2 \end{cases} \text{ in } \Omega^-_i$$

One can verify that $U^+_i(s,\kappa)$ satisfy all the above requirements (additional compatibility conditions must be imposed on $w_i(s)$ at branch points).

**Trial Functions of type $C^1$:** While the above trial functions meet the necessary requirements, better convergence can be expected if higher order continuity is imposed. For example, one can show that the following trial functions satisfy $C^1$ continuity:

$$U^+_i(s,\kappa) = \begin{cases} u_i(s)(1-\kappa^2) + \hat{w}_i(s)\kappa \end{cases} \text{ in } \Omega^+_i$$

$$U^-_i(s,\kappa) = \begin{cases} u_i(s)(1-\kappa^2) + \hat{w}_i(s)\kappa \end{cases} \text{ in } \Omega^-_i$$

where $\{u_i(s),q_i(s)\}$ are unknown functions (compatibility conditions must be imposed on $\{u_i(s),q_i(s)\}$ at branchpoints in order to have $C^1$ continuity for type-2 adjacency). Henceforth, we assume that the trial functions for the Poisson’s equation are of type $C^1$ and are given by Equations (4-5a) and (4-5b).

**Step 3:** The next task is to substitute Equations (4-5a) and (4-5b) in Equation (4-3) and carry out a symbolic integration over the thickness variable $\kappa$. As stated earlier, the task is vastly simplified since $U^+_i(s,\kappa)$ involves low order polynomials. This task was executed using Mathematica\textsuperscript{M}, a symbolic software package.

Once $\kappa$ is eliminated, the problem reduces to solving for $\{u_i(s),q_i(s)\}_{i=1}^N$ in:

$$\text{Minimize } \Pi(U) = \sum_i \left[ \Pi^+_i(u_i(q_i),u_i(q_i),q_i(q_i)) \right]$$

subject to compatibility conditions at branch points.

**Step 4:** The final step is to apply a finite element procedure to minimize Equation (4-6) in an approximate sense using finite element approximation is standard [Shames 85]. We do not discuss the details here, except to note that we employ a Hermite approximation of $u_i(s)$ and a linear approximation of $q_i(s)$.

5. NUMERICAL EXPERIMENTS

In the previous section, we enforced all the required conditions to ensure convergence of the proposed method. In order to study the accuracy of the proposed method, we consider a few field problems whose exact solutions are known. In addition, a few problems with no known closed-form solutions are also considered, and the results are compared to finite element solutions.

Recall that the Kantorovich method consists of two distinct approximation stages (Figure 1-2):

- **Stage 1:** Approximation via Kantorovich trial functions
- **Stage 2:** Approximation via finite element trial functions.

Each stage introduces an approximation error, referred to as stage-1 and stage-2 error. Depending on the problem, one
or both errors may be present. We measure the total numerical error using a pair of normalized $L_\infty$ errors:

$$E_u = \frac{\max|u - \hat{u}|}{\max|u|} \quad ; \quad E_q = \frac{\max|q - \hat{q}|}{\max|q|}$$

**Experiment 1**

The first numerical experiment involves solving the Laplace equation $\nabla^2 U(x, y) = 0$ over 2 convex solids: (a) a rectangle of length 1 and height 0.1, (b) an equilateral triangle of side 1. Dirichlet boundary conditions are specified and their values are such that the exact solution is a quadratic field $U(x, y) = (x^2 - y^2) + 0.2(x + y)$.

Note that for convex solids, the geometric transformation between $(s, k)$ space and $(x, y)$ space, given by Equations (3-1) and (3-2), is linear. Thus the quadratic Kantorovich trial functions of Equation (4-5) and (4-5) are sufficient to capture the field exactly, i.e., one would expect the stage-1 error to be theoretically zero.

Further, in the finite element approximation, we have employed a quadratic approximation of $\hat{u}(s)$ and a linear approximation of capture $\hat{q}(s)$. Thus stage-2 error is also expected to be theoretically zero. Both these expectations are confirmed in Table 5-1.

**Table 5-1: Normalized errors for Experiment 1.**

<table>
<thead>
<tr>
<th>$U(x, y)$</th>
<th>$E_u$</th>
<th>$E_q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\left{ \begin{array}{c} (x^2 - y^2) + 0.2(x + y) \ -5 \end{array} \right.$</td>
<td>$10^{-12}$</td>
<td>$10^{-11}$</td>
</tr>
</tbody>
</table>

**Experiment 2**

The next set of numerical experiments is similar to the first except that the exact solution is a Laplacian field $U(x, y) = (x^3 - 3xy^2)$ over a rectangle of length 1 and height 0.1. We would expect to see stage-1 inaccuracy since the exact field is cubic while the approximating Kantorovich functions are quadratic. Stage-2 inaccuracy is expected to diminish with increasing number of elements. The normalized errors are summarized in Table 5-2. As expected, the errors diminish with increased number of elements, but never reach zero due to the presence of stage-1 error.

**Table 5-2: Normalized errors for Experiment 2.**

<table>
<thead>
<tr>
<th>#Finite Elements</th>
<th>$E_u$</th>
<th>$E_q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$5e^{-4}$</td>
<td>0.0211</td>
</tr>
<tr>
<td>13</td>
<td>$2.73e^{-5}$</td>
<td>0.0022</td>
</tr>
<tr>
<td>23</td>
<td>$7.46e^{-6}$</td>
<td>$6e^{-4}$</td>
</tr>
<tr>
<td>80</td>
<td>$6e^{-6}$</td>
<td>$1.05e^{-4}$</td>
</tr>
<tr>
<td>200</td>
<td>$6e^{-6}$</td>
<td>$0.75e^{-4}$</td>
</tr>
</tbody>
</table>

**Experiment 3**

The third set of numerical experiment involves solving the Poisson’s equation $\nabla^2 U(x, y) = -2$ over two non-convex solids (a) dovetail (Figure 2-4), and (b) a modified L-bracket (see Figure 5-2). Dirichlet boundary conditions are specified and their values are such that the exact solution is the Poisson field $U(x, y) = -(x^2 + y^2)/2$.

The computed solution $\hat{u}(s)$ over the skeleton of the triangle is illustrated in Figure 5-1. This experiment suggests the generality of the proposed technique. In contrast, a 'mid-element' based Kantorovich method is inconceivable for the triangle.

**Figure 5-1: Computed solution over the skeleton.**

**Table 5-3: Normalized errors for Experiment 3.**

<table>
<thead>
<tr>
<th>Field: $-(x^2 + y^2)/2$</th>
<th>$E_u$</th>
<th>$E_q$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dovetail</strong></td>
<td>$10^{-3}$</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td><strong>L-bracket</strong></td>
<td>$10^{-4}$</td>
<td>$10^{-2}$</td>
</tr>
</tbody>
</table>

**Figure 5-2: A modified L-bracket.**
Experiment 4

The final set of numerical experiments involves computing torsional stiffness for various 2-D cross-sections. Closed-form solutions typically do not exist for such problems, barring a few exceptions. We assume here that Saint Venant’s torsional assumptions hold true [Pilkey 02], [Chou 92]. For solids without holes, the problem reduces to solving

\[ \nabla^2 \varphi = -2 \text{ in } \Omega \]

\[ \varphi = 0 \text{ on } \partial \Omega, \]

where \( \varphi \) is the Prandtl’s function … then computing the torsional stiffness given by \( J = 2 \int \varphi d\Omega \).

Note that this experiment goes beyond just computing the two unknown functions \( \hat{u}(s) \) and \( \hat{q}(s) \) over the skeletal functions in that an integral of the 2-D solution over the entire domain must be computed.

We first consider the simple case of a thin rectangle whose torsional stiffness is known in closed-form. Figure 5-4 illustrates the computed Prandtl’s function \( \varphi \) over the skeleton of a rectangle of dimensions \( L = 1, H = 0.05 \).

Further examples appear in Table 5-5. St. Venant’s estimate of the torsional stiffness is implicitly based on a mid-element representation; it leads to fairly accurate estimate stiffness for solids such as the rectangle and I-beam, but is inaccurate, as one would expect, for geometrically complex solids such as the dovetail.

<table>
<thead>
<tr>
<th>2-D FEM Estimate</th>
<th>#Finite Elements</th>
<th>St. Venant’s Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Beam</td>
<td>2.4208e-9</td>
<td>2.4009e-9 (99.1%)</td>
</tr>
<tr>
<td>Dovetail</td>
<td>1.40112e-4</td>
<td>1.3832e-4 (98.7%)</td>
</tr>
</tbody>
</table>

Table 5-5: Torsional stiffness estimates for irregular geometry.

6. INCLUSION OF SINGULARITIES

In Sections 4 and 5 we used polynomials of \( \kappa \) as trial functions – Equations (4-5a) and (4-5b) – to approximate an unknown field \( U(x, \kappa) \). Polynomials are sufficient if the field is sufficiently smooth or if only a global property of the field is desired. However, elliptic fields often exhibit singularities at the boundary that must be captured to predict, for example, failure of a stressed member.

The most common source of a singularity is the reflex or reentrant corner where the internal angle between two adjacent boundary edges exceeds 180°. Figure 6-1 illustrates an example of a reentrant corner with an internal angle of \( \alpha \).

Figure 6-1: Singularity at a reentrant corner.

The nature of the singularity near a reentrant corner depends on the boundary conditions on the adjacent edges, and can be determined \textit{a priori} up to an unknown constant. For example, let a field \( U(x, y) \) satisfy the Poisson’s equation over the domain in Figure 6-1, and let zero Dirichlet boundary conditions be specified over the entire domain. Then the first term of the singularity near the
The reentrant corner of Figure 6-1 in polar coordinates is given by [Strang 73]:

\[ U(r, \theta) = K r^\alpha \sin \left( \frac{\theta \pi}{\alpha} \right) \]

where \( K \) is an unknown constant to be determined.

Unfortunately, polynomial trial functions cannot capture such singularities accurately. We propose here a simple strategy by augmenting polynomial functions with explicit singularity functions; this is similar to how singularities are handled in dassic finite element analysis [Strang 73]. The proposed strategy is as follows:

1. First create an artificial ‘singularity region’ of radius \( \epsilon \) centered about the reentrant corner, as illustrated in Figure 6-2.
2. Then represent the field in the \( \epsilon \)-region by a singularity function, and elsewhere, by polynomial trial functions as before.
3. Finally enforce \( C^0 \) or \( C^1 \) continuity, as appropriate, along the common boundary of the \( \epsilon \)-region.

**Figure 6-2: Isolation of the \( \epsilon \)-region**

At the present time, we have not implemented the above strategy, and expect to do so in the future.

### 7. CONCLUSIONS

The preciseness and algorithmic nature of the proposed method leads to a high degree of automation and accuracy. Standard solid modeling, finite element and graph theoretical concepts are sufficient, i.e., ‘special’ modeling techniques used in mid-element based techniques are not required. The proposed method permits use of singularity functions if desired.

On the other hand, there are two significant challenges associated with the proposed method: (1) it requires the computation of a skeletal representation; while this is known to be a hard problem, especially in 3D, recent research, example, [Etzion 02], is promising, and (2) the skeleton is ‘sensitive’ to small changes or representational inaccuracies in the boundary [Rezayat 96], and may require ‘smoothening’ or de-featuring [Donaghy 96].

### 8. REFERENCES


Invited Speaker
Jami Shah  
Arizona State University

ABSTRACT:

Evolution of geometric feature recognition techniques through four generations

Research in automatic feature recognition from 3D CAD models spans three decades. This talk will give a historical perspective of feature recognition methods. It will discuss the simple rule based systems of the 1970s, graph based systems of the 80s, volume decomposition and hint based systems of the 90s, and current hybrid systems. A very brief summary of the foundations of each approach and its limitations will be presented. The talk will also show snapshots of four generations of feature recognition systems developed in our lab. It will compare current feature recognition approaches from several criteria: robustness, computational complexity, range of features recognized, and extensibility. If time permits, work in progress in automatic recognition will be discussed.
Session 3A
Structured Meshing
AUTOMATIC NESTED REFINEMENT – A TECHNIQUE FOR THE GENERATION OF HIGH QUALITY MULTI-BLOCK STRUCTURED GRIDS FOR MULTI-SCALE PROBLEMS USING GRIDPRO®

Krishnakumar Rajagopalan¹, Peter. R. Eiseman²

¹Program Development Company, White Plains, NY., U.S.A. krishna@gridpro.com
²Program Development Company, White Plains, NY., U.S.A. eiseman@gridpro.com

ABSTRACT

A technique for the generation of conformal adaptive refinement of hex meshes is presented. Automatic Nested Refinement is a technique for generating recursively nested topology automatically. It can be applied inside GridPro’s topological paradigm to generate block structured grids which can resolve tiny features in the problem while providing for a smooth and consistent way to transition to a larger scale. The selection of self-similar, “fractal-like” topological templates makes the selection of number of levels easy, while making the technique feasible for infinite levels of adaptive refinement. The technique has been programmed and integrated into the GUI of GridPro, making it very accessible and easy to use. This method illustrates a way of generating structured grids in an unstructured way, made possible because of the topological paradigm of GridPro. Because the technique uses the topological paradigm, it inherits all the advantages the paradigm offers, including the ability to handle very complex geometries, parametric variation of surfaces, and the ease of use, speed and quality of GridPro. The technique has been illustrated using a variety of applications. This method has proved to be a fast, efficient, automatic and reliable means to perform physical simulations that have a disparity in scale.

Keywords: mesh generation, multi-scale problems, multi-block grids, nested refinement, fractals, conformal adaptive hex meshing, GridPro®

1. INTRODUCTION

The term multi-scale problems are usually applied for computational analysis of geometries having widely varying length scales. For example, full-field oil reservoir simulations are usually multi-scale problems. For full field simulations of oil reservoirs, the scale of the field is usually of the order of miles, while the scales of pipes and other such flow regions are of the order of inches. Computational analyses of multi-scale problems are usually hard because of the need to generate a computational mesh which can resolve both the higher end and lower end of the scale of the geometry, and also taking care that the transition is smooth and consistent. At present, there are a variety of methods to generate meshes for such problems. They generally involve hybrid or completely unstructured methods, or structured methods with hanging nodes and other techniques.

Multi-scale problems usually require a different approach to grid generation. To accurately resolve the solution of the field near the smallest spatial scales, the size of the discretized spatial cells need to be of the order of these smallest scales. If the same size cells are used in the large-scale field too, then there will be too many cells at places where it is not needed, and this can cause the computational time of the numerical simulation to be too big for any practical purposes. The field near the small-scale geometries needs to be resolved with small cells, while the rest of the flow field should be discretized with cells of large size. These kinds of problems require special kinds of grids - grids which can transition from a small size near the small features to large cell sizes in the general field.

Multi-scale problems are common in reservoir simulations. But these problems can also occur in other applications such as the analysis of very thin turbine blades, the analysis of flow and heat transfer about/inside thin pipes in biological flows etc. Also, as the need to produce more complete and reliable analysis grows, problems which are not usually multi-scale can become so. For example, the flow past an aircraft or a car is not usually a multi-scale problem. The study of fluid flow around a small appendage such as an antenna is usually considered redundant. But cases can occur where such analysis might be useful and needed, and in such cases, a reliable method to produce grids for such cases needs to be available. Other examples, like the analysis of the flow past a thin structure like a strand of hair, or the analysis of flow over riblets, etc can also be considered as multi-scale problems. Sometimes, such a multi-scale grid may be necessary to reduce the aspect ratio of the cells in certain locations. The modern CFD (computational fluid dynamics) algorithms are generally quite tolerant to high aspect ratios.
But there can be cases when low aspect ratio cells are desired in certain key locations.

In this paper we show a technique by which one can easily generate conformal multi-block grids for multi-scale problems. The basic idea was to stack up elementary topological elements in a certain way so as to handle scale geometries in grids. A recursive way of stacking up a single element was preferred because such a technique could be programmed and hence provide for an easy and automated way to handle multi-scale problems. Such a recursive or 'nested' structure is called a nested refinement in that region.

A program called Nest was developed which takes in a certain input and gives a nested topology as output. The user then loads in this nested topology (as a file), and links it to his existing topology in a few mouse clicks. Nest operates in an abstract topological level only, and does not need to know anything about the actual surfaces. The grid generation engine in GridPro takes care of topology conforming to the actual surfaces. To make the utility more accessible, a button was added in the GridPro GUI which leads to a dialog box which runs the program to create the nested topology.

2. GRIDPRO’S TOPOLOGICAL PARADIGM

GridPro’s topology paradigm is a very powerful and unique technique. This paradigm reduces the problem of generating multi-block grids to that of generating a set of loosely positioned topology corners and their connectivity. The advantages of such a technique are its ease of use and automation – because one does not have to worry about positioning the corners exactly. Also, topology can be used as a template and can be quite independent of the surfaces themselves. This gives rise to an important advantage that topology need be built only once for a grid generation problem. Another main advantage is the automation which can be achieved on the topology paradigm. Nest is one example of how this kind of automation can be achieved.

Figure 1 shows a simple example to illustrate how the topology paradigm works in GridPro. Both topology 1 and topology 2 produces almost identical grid point positions. The iterative grid optimization algorithms in GridPro are very robust and ensure the highest quality of the grid for a particular topology. Another example to illustrate the flexibility of topology is shown for a simple case below. The robust optimization algorithms can untwist a folded block as shown in figure 3.

![Figure 2](image1.png)  
**Figure 2.** Topology and Corresponding GridPro Grid

![Figure 3](image2.png)  
**Figure 3.** Topology corners displaced, but it produces the same grid

![Figure 4](image3.png)  
**Figure 4.** Grid Optimization in GridPro. Grid for iteration step - (a) 100 (b) 500 (c) 3000

![Figure 1](image4.png)  
**Figure 1. Loose positioning of topology corners**
Figure 4 illustrates the grid point movement algorithms by showing pictures of a grid at different iteration steps for the grid generation engine. Again, we see that the folded grid cells in (a) have been unfolded and neatly placed in (c). This illustrates the robustness of the GridPro’s grid generation engine.

Now that we have a robust topology engine behind us, we have several possible options. One option is to develop schemes to automatically generate topology. Another is to develop an interactive topology generator. An interactive topology generator called the AZ manager comes with GridPro which has numerous features for automating small parts of topology creation. This provides a kind of semi-automatic approach to topology creation. The fully automatic approach to topology creation is a present research topic in Program Development Company. For more details on topology creation and other aspects of grid generation in GridPro, please refer to the GridPro TIL manual [1] and the GridPro tutorials [2].

3. THE NEED FOR NESTED REFINEMENT

Consider the geometry as shown in figure 5(a) or figure 8. There are many ways we can go about generating a grid for such geometries. But suppose we set quality criteria on the grid. – Set the worst aspect ratio of the grid to be less than 10, and the grid should be smooth etc. A simple way is to just build a structured grid with a lot of grid points to capture the small features in the geometry. If we take this strategy, then the grid in figure 5(a) will have about a million cells. In figure 8, the radius of the tiny tube is about 0.005, and the box dimensions are of the order 10*10*10. This means that a simplistic strategy like the one described above will yield a grid having more than 10 billion cells! Even for such simple geometries, the problem starts becoming intractable using simple methods.

Traditionally, because of the difficulties in generating tractable structured grids for multi-scale problems, triangular/tetrahedral or hybrid meshes, meshes with hanging nodes etc have been the only grids used for such problems. Automatic nested refinement offers to fill in for a structured method which can do multi-scale problems in a reliable and automatic way, offering the advantages of an unstructured approach but a locally structured grid at the same time.

4. AUTOMATIC NESTED REFINEMENT

The basic idea behind nested refinement is simple and has been around for sometime. There are many ways of generating topology to get such grids in GridPro. Some references can be quoted from the GridPro TIL manual [1], where a nested refinement structure has been used to illustrate the idea of components in the topology input language (TIL). Nested refinement can be looked at as a generalization of a structure called the clamp[3].
Figures 5a, 5b, 5c, 5d illustrate how nested refinement can be applied in a topological sense and how this reflects in a grid for the geometry shown. The geometry consists of a regular two dimensional box with tiny sinusoidal waves covering the bottom. In this case, there are 50 sinusoidal waves covering the bottom. For a good resolution, one wave needs to be captured with about 40 grid cells. This means we need about 2000 grid cells in the bottom surface. In the grid shown, there are more than 3000 grid cells in the bottom, while there are only 12 in the top. There are 5 levels of nested refinement in this case. The construction of a self-similar structure makes it possible to produce a grid which expands smoothly out into a low resolution region. Also, the optimization algorithms in GridPro make it possible to get a high quality grid throughout.

This concept can be easily extended into three dimensions. Nest does this by simply extruding a two dimensional topology. Nest can be applied consecutively in two transverse directions to get a true three dimensional nested refinement.

5. IMPLEMENTATION OF NEST

Nest can be accessed from the AZ manager, the Graphical User Interface of GridPro. Clicking on the nest button will pop up a dialog box like the one shown in figure 6.

In GridPro, arbitrary groups of topology corners can be identified as a group. At the time of this writing, there can be 32 such separate groups of corners. The first line in the dialog
box in figure 6 specifies the group of corners which identifies the region of the grid where a high resolution is required. Similarly, lines 2 and 3, called the low density group number and direction group numbers respectively, specifies the group of corners which identifies the low density region and a special group called the direction group. The direction group is necessary because the nested refinement is done in one of two possible directions at a time, and the direction group of corners identifies the direction.

The number of levels is a crucial number which needs to be chosen with care by the user. This is the number of times the self-similar structure is sub-divided. For now, only 5 groups have been given in the dialog box. But the user can nest up to arbitrarily many levels of nested refinement using the program from the command line. In the limit of infinitely many recursions, it will converge to a fractal structure.

The ratios specified alongside the number give the overall cell size/resolution ratio which will be achieved between the high and low density regions. For example, in the case of the wavy wall shown in figure 5, the number of cells required in the top is 12 and the number of cells we need in the bottom is around 3000. This threshold ratio is close to 200, and hence, the number of levels can be chosen to be 5, which will give an average ratio of 243. The ratios go as $1:3^n$, where $n$ is the number of levels of nested refinement. This can be mathematically proved [1].

Figure 7(a). Oahu, Hawaii. The complete grid.

In figure 7(a), the island is the tiny white dot in the middle. As we can see the size of the cells far away from island is of the order of the size of the island itself.

Figure 7(b) shows a closer view of the grid where one can note how nested refinement helps in reducing the propagation of many grid lines by redirecting many of them back into the island.

Figure 7(c) shows the grid very close to the island. This is a structured grid which lies inside a cocoon of nested refinement. Zooming in closer the section shown by a rectangle figure 7(c), we note a smooth high resolution grid capturing the coastline of Oahu.

Figure 7(b). Oahu, Hawaii. Zooming in closer…

6. APPLICATION OF NEST TO SOME GEOMETRIES

6.1 Island of Oahu, Hawaii

Nest has been used to obtain a grid around the Hawaiian island of Oahu. This is a typical example of a multi-scale problem. The coast line of the island is crinkly and this typically requires a large number of cells close to the island. But the cell size far away from the island (in the far field – the ocean) can be the size of the island itself. The following pictures show the nested grid around Oahu.
6.2 Curved Wavy Tube in a Box

A thin curved tube in a box whose radius fluctuates by a small amount is another example of a multi-scale problem. Figure 8 illustrates such a geometry. The arrows indicate a flow direction, and the problem posed can be that of calculating the drag experienced by the tiny tube by such a flow.
Figure 10 shows two cross-sectional sheets perpendicular to each other. This illustrates the resolution of the grid near the wavy tube. Nest achieves the high resolution near the tube and the gentle transition to a lower resolution as one moves away from the tube.

Figure 10 Two cross sectional sheets

Figure 11 shows the results from a simulation performed using Fluent®. Further details of the simulation can be obtained by referring to [1].

7. CONCLUSION

A technique called ‘automatic nested refinement’ was implemented in the framework of GridPro for addressing multi-scale problems in the context of multi-block structured grids. It offers automation and great flexibility in handling scale differences for grid generation. Future work will involve developing topological structures which will provide for more optimized adaptation and more automation. One means to do this is to use three dimensional templates in a recursive way to do such adaptation [4], [8]. Verification and validation of the use of nested refinement by comparing performance of nested grids with other hybrid and unstructured grid methods will also be an important future direction.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the helpful interactions with Prof. Houston Wood, Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, Virginia.

REFERENCES

[1] Automatic Nested Refinement --- A Technique for the generation of high quality Multi-Block structured grids for Multi-Scale problems using GridPro® - Master’s Thesis for Krishnakumar Rajagopalan presented to the University of Virginia, May 2003


MULTIPLE STATIONARY AND MOVING BOUNDARY HANDLING IN CARTESIAN GRIDS

Kerem Pekkan

Georgia Institute of Technology, Atlanta, GA., U.S.A. kerem.pekkan@bme.gatech.edu

ABSTRACT

A Cartesian grid generation methodology is developed for unsteady control volume computational fluid dynamic (CFD) solvers. Arbitrary combinations and numbers of moving and stationary boundaries are allowed to exist in the two-dimensional Cartesian grid template. Specific definitions for the possible cases, CFD solver requirements and moving geometry handling algorithms of Cartesian grids are described. Applications are selected from bio-fluid dynamics and aerospace propulsion to demonstrate the capability of the method.

Keywords: Cartesian grids, moving boundaries, computational fluid dynamics

1. INTRODUCTION

Moving boundary problems of nature[1][2] and technology[3][4][5] is an attractive research area. Besides the trivial computational difficulties of interface movement, additional complexity is introduced due to surfaces that are irregular and not aligned with the orthogonal coordinate planes. Moreover in some problems multiple stationary and moving surface combinations can exist.

In moving boundary problems, in addition to the structured[6] and unstructured boundary confirming grids[7], embedded Chimera like moving zonal procedures[8] and level-set methods[9], Cartesian or cut-cell techniques can be used to define the solution domain. Cartesian grid approach is an efficient, versatile and rapid alternative. Irregular cut-cells and uniform rectangular volumes that are located near the boundaries and in the far field, respectively. As will be demonstrated Cartesian grid intersections depend on the local topological character of the boundary curves, therefore making the Cartesian approach problem independent and suitable for broad research interests.

Unless a structured boundary layer grid (obtained by normal offsetting of the boundary) and optional tetrahedral transition elements are introduced, Cartesian grids may not be in the best possible quality for the given application but still can be preferred due to their high turnaround time and wide ranges of applicable geometry. For boundaries with many sharp corners it is another alternative to unstructured (tetrahedral) and multi block structured grids.

When Cartesian grids are used, even in a simple two-dimensional problem, triangular, quadrilateral and pentagonal elements may coexist. Since the geometric possibilities are close to unlimited a systematic approach is necessary. Structured programming, CAD algorithms and approaches from constructive solid geometry are employed for the realization of this highly geometric task.

Figure 1. Solution algorithm and time loop.
Fig. 1 shows the main components of a moving boundary CFD solver that employs Cartesian grids. Details of the grid generation routines will be described next, followed by selected applications.

2. DEFINITIONS

The Cartesian grid terminology for stationary boundaries is quite complete in the literature[13][14]. In this section an extension will be made to cover domains that involve multiple stationary and moving boundaries.

In Fig. 2, a rectangular “template” containing stationary (solid) and moving boundaries, having five and seven “square cells” along x- and y- directions is plotted. Given the length and width of the template, integer number of square grids can be generated inside. Template boundary sides are termed as EAST, WEST, SOUTH and NORTH, where inflow/outflow, reflective, transmissive, injecting, moving wall boundary conditions can be specified. Also for the side of any cell inside the template, a boundary condition can be assigned when needed for specific applications.

In all problems, solid and moving walls are bordering the flow field of interest. “Curves” that form the boundaries of solid/moving walls are specified as line segments, in an order so that when the curve parameter increases, solid bodies are always on the left.

In a general problem there may be moving and solid boundaries connected in series, as shown in Fig. 2. These series of curves are termed as “streams.” In streams intersection points of moving and solid curves are labeled as “kink points”. A stream can be composed of a single moving or solid curve. Streams are allowed to loop, forming voids or closed bodies, with coinciding start and end points.

A boundary curve, passing through a template cell, intersects its sides in two points. Tracing the curve with the solid region being on the left-hand side, first intersection is denoted as “point 1a” and second as “point 2a.” Cell intersection points are stored with respect to coordinates relative to the cell.

If number of intersection points in a cell exceeds two, cell types that are not covered by the solvers’ cell-type-domain may appear. These cells need special treatment and named as “degenerate cells” in the Cartesian literature [13]. In that case, extra intersection points will be labeled and stored as “point 1b” and “point 2b,” to be used in degenerate cell handling routines, §3.7 A simple degenerate cell example is given in Fig. 2.

3. CARTESIAN GRID GENERATION ROUTINES

Cartesian Grid Solvers does not require a separate grid
generator program, while for arbitrary moving complex geometries; generation of the template grids and finding local intersections is a complicated geometric task.

Unlike boundary-conforming grids where global re-meshing, at each time step, alters the positions of grid points, Cartesian grid points are stationary. Thus for Cartesian control volumes that are not cut by a moving boundary, a geometric conservation law or Jacobian transformations is not needed.

3.1 Modeling Solid/Moving Wall Boundaries

Curves that form the boundaries of solid or moving walls are approximated with line segments. A line segment is the lowest object of the stream structure. It is defined by the parametric equation of straight line. Parametric representation of line segments is found useful especially in finding cell intersections. For this type of representation position vectors of the start and end points of the straight line are needed. A single curve parameter defines the position of any point on the line segment. Segment center velocity and its magnitude are also stored to allow for variable offsetting along a moving curve types.

The curve structure used in the code composed of the line segments, number of line segments that make up the curve and type of curve, which may be moving or solid. Line segment sequence of each solid/moving curve is arranged so that solid bodies are always on left. The geometric information of the input curves are kept in the following structure:

```c
struct curve {
  struct line seg[N_SEGMT];
  int type; /* SOLID or MOVE */
  int n_segmt; /* end index of seg[] */
};
```

3.2 Cell Size and Number of Cartesian Grid Lines

Since the domain is rectangular and the cells are square, number of grid lines in x- and y- directions is dependent on each other. First the minimum number of grids in each direction is calculated, which is the coarsest possible grid for the given template dimensions. Then, if an extra refinement is required, number of grids is increased in both directions with the same ratio. If the template side dimensions are not whole numbers, then they should be expressed in rational form, i.e. numerator over denominator. In that case, to calculate cell size and template grid line locations accurately, integer arithmetic needs to be performed.

3.3 Finding Streams

A general geometry is composed of arbitrary number of solid and moving curves. At the start of the problem, i.e. at initial state, some part of the solid boundaries may be covered with moving curves and may not saw the flow region. For a regressing material this situation is sketched in Fig. 3. As the solution proceeds, these solid boundaries will expose out and start affecting the flow. To obtain the transient solution without any restart, at each time step the flow boundaries should be identified. Curves defining flow boundaries or wetted areas are called “streams” and the corresponding process is called “stream generation.”

In Fig 3, initially there is one moving curve (A-B) and one solid curve (1-c-2). Solid curve is composed of two line segments and the initial stream is made up of the moving curve (A-B) only. As the boundary moves, at a later time step the number of moving curves increases to two. The stream is now made up of three curves (a-b, b-c-c-d, d-e). In a general problem the number of streams may be more than one. Algorithms developed in this study allow multiple streams, cover situations involving merging and break-up of moving boundaries and can also generate loop streams.

If there are no moving walls and all of the geometry that defines fluid boundaries is solid, then each solid wall is assigned as a new stream. If there are moving walls, new streams are generated by tracing moving and solid curves alternatively. In this trace, “A stream can form a closed loop or start and end at a template boundary” is the basic rule. Until this rule is satisfied, each trimmed/extended moving curve is traced first in its start direction and then towards its end. During these traces, intersections with other curves will be detected. Each new detected curve during this trace is kept in the order as a member of the generated stream. Boundaries that form loops can also be detected by this procedure.

3.4 Cartesian Cell Intersections

The basic geometry is illustrated in Fig 1. For each line segment, of each stream, intersections with vertical and horizontal grid lines are searched. If an intersection is found, intersection point and its type (moving or solid) are stored in the cell structure relative to cell coordinates. The coordinates should be specified relative to square cells because of the accuracy considerations. This algorithm is different than the one proposed in [1]. In that study boundary curves were traced and cell coordinates are
defined as integer variables taking discrete values to overcome the accuracy problem. The intersection routine that is developed here considers the sense of each line segment and differentiates different cases. Including specific orientations of segments, like cases parallel to Cartesian grid lines. Two intersections are allowed and typical for each Cartesian cell. If more than two intersections are found, their positions are stored for degenerate cell considerations.

For geometries involving moving and solid boundaries, there are five basic cell types. These are: full flow, solid, cut-solid, cut-move and kink cells, Fig. 4. Cut cells contain a single curve segment, which may be moving or stationary. Kink cells are cut cells where a solid curve ends and a moving curve starts or vice versa. The same convention also holds for segments inside cells: the solid part is on left, in the direction of curve parameter increase.

### 3.5 Cell Structure

In the solution domain each square formed by the grid lines defines a control volume, named as cell. Number of cells are equal to the number of grids. For each cell, besides flow variables the following information is also stored:

- Basic cell type (Fig. 4)
- Cell sub-type (Fig. 5)
- Position vector of first and second intersection points.
- Intersection types of first and second intersections. (Which depends whether the cell is intersected at that point by a solid or a moving curve.)
- Position vector of first and second degenerate intersection points and their types.
- Position vector of the kink point.
- List number that the cell is combined.
- Cell area.
- Position of cell center.
- The boundary condition specified for any of the cell sides.

### 3.6 Cell Sub-Types

The intersection type (cut segment crossing whether in or out of the Cartesian cell), and the cell side each intersection point is located determine the cell sub-type. As a solver convention, these sub-types are grouped and presented in Fig. 5. Depending on the conditions given in Fig 5, the solver differentiates 48 different cell sub-types. Although only cut-moving/solid (CUT-M/S) cells are drawn, for each cut cell sub-type there is also a corresponding KINK cell with intersection points at the same locations. In that case the kink point, point-3 in Fig. 4, is located at an arbitrary position inside the Cartesian cell.

### 3.7 Degenerate Cells

In the solution domain, if arbitrary intersections with the input geometry are allowed, some cells that are not recognized by the Cartesian solver may appear. These cells are named as “degenerate cells” in the Cartesian literature. (A simple example is a cell with more than two intersections, Fig. 2)

<table>
<thead>
<tr>
<th>PROBLEM TYPE</th>
<th>NUMBER of UNIT GEOMETRIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>32</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
</tr>
<tr>
<td>P3</td>
<td>64</td>
</tr>
<tr>
<td>P4 diagonal</td>
<td>16</td>
</tr>
<tr>
<td>P4</td>
<td>88</td>
</tr>
</tbody>
</table>

Table 1. Number of possible unit geometries for each degenerate cell problem type.
By increasing grid size or slightly shifting the input geometry some of the problematic cases can be overcome. However such remedies work only for bodies that are not moving. For applications involving continuously changing shapes and offsetting, these problematic geometries must be identified and suitably approximated.

Depending on the number of intersections in the degenerate Cartesian cell, geometrically possible problem types can be grouped in to four. These problem topologies will be labeled as P1, P2, P3 and P4 cells (With one, two, three and four intersections in a Cartesian cell respectively.) For each problem type, the rotations and symmetries of the basic geometry should be considered, together with the type (moving or solid), of the intersecting boundary. The total number of unit operations that is taken into account for each problem type is given in Table 1.

Figure 6. P2 and P4 type degenerate cells: In all figures dashed lines represent the final approximated boundary after degenerate cell corrections. Both line segments are of same type (Both MOVE and both SOLIDS). For the definitions of cell intersection points: 1a, 2a, 1b and 2b refer to Fig. 2.

Figure 7. Diagonal P4 problem - one of the four sub-types. The four cases that are shown on the left are same type of curve intersections. Different type intersections are plotted on right.

Figure 8. P1 problem: P1 problem is possible if the segments that make a sharp corner are of the same type and one of the segments coincide with the template grid lines. There are two basic geometry types. (And each type has two variations, a and b) For the basic geometry type 1a (top left figure), the ambiguity in cell intersections is corrected by: Mark (cell c) and (cell s) as FULL FLOW or SOLID. (cell b, d, e, g)

To disclose the scope of the work some examples of the primary geometries and rules that are used in the code are given in the Figs. 6, 7 and 8. For degenerate cells with intersections of different curve types (solid/move), approximated geometry is sharp and defines a kink type of cell. Without considering these topologies a general moving body Cartesian solver is not possible.

3.8 Marking Solid Cells

Procedures describe in the previous sections specify the geometric properties for each cut cell in the solution domain. The remaining cells are either solid or full flow. For solid cells, no flow solution is needed. Therefore they must be distinguished from the fluid flow cells before starting the solution.

The procedure that is used is similar to the one used in [1]. All the Cartesian cells are traced first horizontally and then vertically. During each trace, cell type does not changed until, a cut cell is reached, and at that cell, the marking type is reversed and switched to either FULLFLOW or SOLID. The tracing proceeds with this marking type afterwards. By taking into account the detected cut cell sub-type, initial cell type assumption and previous type marks are corrected. The algorithm is tested in various geometries and found to be working flawlessly in all cases considered so far.

The solid cell marking procedure of [1] does not take the cut cell sub-type into account. For this reason an extra trace, either in horizontal or vertical direction is needed. Even with this extra trace, ambiguous geometries are still possible. In this study since in marking type switches,
For all the combined cells in the template, cell states are area-averaged, area and cell center of the combined cells are found. A different list number is assigned to each combination.

In Fig. 10, Cartesian grid Information for various template locations of the letters ‘S’ and ‘A’ are plotted. Close-up regions demonstrate some critical locations and treatment of degenerate cells. The numbers shown on Cartesian cell centers, represent the combination list that the cell belongs. F, C and S stand for FULLFLOW, CUT and SOLID respectively. Due to the boundary movement, cell information is regenerated at each time step.

### 3.9 Cell Combinations

Intersections of arbitrary line segments may produce tiny cells, which minimize the time step size. If a fast Cartesian solver is the aim, these tiny cells must be combined and treated as a single control volume. The segment mid-point normal rule [1] is practiced here. Which will be referred as the best combination.

![Figure 9. Cell (i, j) is combined with the first of the two alternatives.](image)

Around confined regions and for cut cells with neighboring template boundaries, the best possible combination may not exist or the planned combination may produce an undesired size increase that decreases the local spatial accuracy. In order to acquire a consistent control volume size as much as possible, throughout the template, maximum three alternatives of the four neighboring cells are returned to the code in the order of the best combination possibility. In cases when the best combination cell does not exist other alternatives are considered in order, Fig. 9.

![Figure 10. Cell type and combination information. Lower left corner of letter “S” and top right of letter “A”](image)
4. MOVING WALL EXAMPLES

Geometry handling and moving wall-offsetting functions will be demonstrated via three examples. Geometric problems encountered during normal offsetting involve edge separation and degenerative close loops. These are related to the local curvature and offsetting distance. For the following examples wall offset velocity is constant.

As the result of detailed considerations of degenerate cells, some simple merging and break up situations can be detected, without any extra merge/break-up routines. Break-up region is realized by marking some degenerate cells as full flow. Fig. 12 demonstrates an offsetting sequence leading to break-up. Template is square of side 8 mm. Offset velocity is 2.5mm/s and grid size is 46x46.

Wall offsetting accuracy can be assessed by comparing exact flow area vs. perimeter plots. For an expanding and contracting circular body, Fig 13, such plots are generated for two grid sizes. A difference of ~10% is observed for the selected coarse grid, Fig 14. The circular body is initially represented by a 20-segment polygon and wall speed was constant. Increasing the grid size 1/3 resulted closer values.

Geometry handling and moving wall-offsetting functions will be demonstrated via three examples. Geometric problems encountered during normal offsetting involve edge separation and degenerative close loops. These are related to the local curvature and offsetting distance. For the following examples wall offset velocity is constant.

As the result of detailed considerations of degenerate cells, some simple merging and break up situations can be detected, without any extra merge/break-up routines. Break-up region is realized by marking some degenerate cells as full flow. Fig. 12 demonstrates an offsetting sequence leading to break-up. Template is square of side 8 mm. Offset velocity is 2.5mm/s and grid size is 46x46.

Wall offsetting accuracy can be assessed by comparing exact flow area vs. perimeter plots. For an expanding and contracting circular body, Fig 13, such plots are generated for two grid sizes. A difference of ~10% is observed for the selected coarse grid, Fig 14. The circular body is initially represented by a 20-segment polygon and wall speed was constant. Increasing the grid size 1/3 resulted closer values.

Fig. 11 is a regressing diameter pipe with an arbitrary shape. This geometry is selected because of its high convex curvatures, which may cause problems in geometry dependent offsetting codes.

Fig. 12. Break-up of an arbitrary solid body. Demonstrating the degenerate cell handling procedures and stream formation. The initial geometry is drawn using thick lines. Not all the zones are plotted. During break-up, time step is modified.

Fig. 13. Contracting circle in 100x100 grid. Wall Speed 5 units/sec, Template size 4 units square, Time step 0.001 sec. (Part of the mesh is shown)
REFERENCES

CONSTRUCTING ANISOTROPIC GEOMETRIC METRICS
USING OCTREES AND SKELETONS

Ko-Foa Tchon  Mohammed Khachan  François Guibault  Ricardo Camarero

Centre de recherche en calcul appliqué (CERCA)
5160, boul. Décarie, bureau 400, Montréal (Québec) H3X 2H9, Canada.
[tchon|khachan|francois|ricardo]@cerca.umontreal.ca

ABSTRACT

A three-dimensional anisotropic metric for geometry-based mesh adaptation is constructed from a triangulated domain definition. First, a Cartesian background octree is refined according to not only boundary curvature but also a local separation criterion from digital topology theory. This octree is then used to extract the domain skeleton through a medial axis transform. Finally, an efficient anisotropic metric is computed on the octree using the curvature tensor estimated from the boundary triangulation and the local domain thickness information embedded in the skeleton. Applications to geometric adaptation of overlay meshes used in grid-based methods for unstructured hexahedral mesh generation are also presented.

Keywords: geometric adaptation, anisotropic metric, octree skeleton, boundary curvature, domain thickness.

1. INTRODUCTION

Accuracy of finite element and finite volume methods is strongly dependent on the quality of the domain discretization and, more precisely, its mesh. Control of the size, stretching and orientation of the mesh elements is thus crucial. User experience can guide the generation of the mesh to manually adapt it to the problem at hand. Higher vertex densities can be requested in expected high load regions or boundary layers, for example. Such an a priori approach is, however, tedious and error prone. Automatic methods based on a posteriori error estimators have received extensive attention over the years and proved the effectiveness of solution-based mesh adaptation. See [1] and the references cited therein, among others. The Object-Oriented Remeshing Toolkit (OORT) developed at CERCA implements such methods [2]. However, if no solution is available, when generating an initial mesh for example, alternative methods based on domain geometry must be used.

The numerous unstructured mesh generation methods presented in the literature propose many different geometric adaptation approaches. However, like their solution-based counterparts, these algorithms always need to first map the characteristics of the target mesh elements at every point of the domain. Early advancing front methods relied on user specified sample points manually triangulated to form a coarse simplicial background mesh [3]. Target mesh properties are then computed at any point of the domain by locating the host background element and linearly interpolating the sample vertex values. Automated alternatives have then been developed using unconstrained Delaunay triangulations of the vertices of pre-meshed domain boundaries [4]. The target mesh spacing is then interpolated in the domain from boundary specified parameters. Furthermore, the discretization of the boundary itself can be automated using curvature, angle and proximity criteria, see [5] for example. However, such so-called empty Delaunay meshes are very coarse and can result in unwanted abrupt variations of the target mesh properties. To alleviate this side effect, an alternative interpolation scheme based on natural neighbors has been proposed [6]. Even smoother maps can be generated by diffusing target mesh parameters in uniform Cartesian background grids using point and line sources and a Poisson equation [7]. The resulting mesh gradation is very smooth and the uniform structure of the background grid facilitates host location for target parameter interpolation. A uniform grid cannot, however, capture very complex target maps with extreme length scale variations. Quadtrees, in two dimensions, and octrees, in three dimensions, are better suited for such maps because they enable local refinement while retaining implicit recursive structures facilitating host location. The use of quadtreess and octrees for unstructured simplicial mesh generation has been pioneered two decades ago [8] and a review can be found in [9]. These methods recursively divide the domain bounding box until the boundary features are adequately resolved and store the result in a tree structure. Allowing only a difference of one refinement level between neighboring cells results in smooth gradation. To generate a valid mesh, the tree cells are then usually split into simplicial elements and the boundary is recovered. However, since the size distribution of the terminal cells is well adapted to the domain geometry by construction, the final tree structure can also be used almost directly as a target map for other meshing algorithms such as the ad-
vancing front method [10–13]. Quadtrees and octrees can also be used solely as support mediums for more elaborate sizing functions. Their refinement is then not directly based on the domain geometry but rather on the adequate capture of the sizing function gradients [14].

The above list of geometry-based mesh sizing control strategies is far from exhaustive and their combination would give infinite possibilities. Two main ideas emerge however. First of all, target mesh specifications may take many forms but storage in a background mesh, instead of on the fly recomputation for example, is the most flexible approach. It decouples the control map from both the adaptation algorithm as well as the target mesh type, structured, unstructured or hybrid for example. This approach is thus potentially compatible with solution-based adaptation algorithms. The second common idea is that geometric adaptation should be based on the local curvature of the domain. Curvature-based sizing is commonly used for curvilinear and surface meshes and has a solid theoretical foundation [15]. It is, however, insufficient to simply diffuse such a sizing throughout a three-dimensional domain. An additional adaptation criterion based on the local thickness of the domain must be introduced to take into account regions with narrow gaps for example. Designing such a criterion is not trivial. At present, most attempts use heuristics based on proximity between boundary vertices, segments and facets and strongly depend on the boundary mesh itself. The present work proposes to use digital topology theory to extract local thickness information from the domain skeleton on a Cartesian background octree. To resolve possible small gaps in the domain, this octree is first refined according to not only boundary curvature but also a topologic separation criterion. Furthermore, to enable anisotropic adaptation, the octree is only used as a support for a Riemannian metric extracted from the domain boundary curvature tensor and the local thickness information retained by the skeleton. The resulting algorithm has been implemented in a package called GeoMetric and applied to overlay mesh adaptation for grid-based unstructured hexahedral mesh generation methods.

2. SOME DIGITAL TOPOLOGY

Pioneered by Azriel Rosenfeld [16], digital topology is mainly used in image processing and provides discrete analogs to Euclidean topology. It is built on the notion of connectivity of adjacent pixels in two dimensions and voxels in three dimensions. Consider, for example, a two-dimensional grid that partitions space in square pixels. Connectivity in this grid is based on two types of adjacency: two pixels are 4-adjacent if they share an edge and 8-adjacent if they share a vertex. Note that 4-adjacency implies 8-adjacency because two pixels sharing an edge also share vertices. Similarly, a three-dimensional digital grid partitions space in cubic voxels. Two voxels are 6-adjacent if they share a face and 26-adjacent if they share a vertex. Furthermore, two pixels (resp. voxels) are α-connected if there is a path of α-adjacent pixels (resp. voxels) between them. The set of α-adjacent neighbors of a pixel or voxel i is called its α-neighborhood and noted $N_\alpha(i)$ or simply $N_\alpha$ (Figs. 1 and 2). Using these definitions, digital analogs to curves, surfaces and skeletons are presented hereafter for uniform grids before being extended to quadrees and octrees.

![Figure 1: Two-dimensional neighborhoods.](image1)

(a) 4-neighborhood  (b) 8-neighborhood

![Figure 2: Three-dimensional neighborhoods.](image2)

(a) 6-neighborhood  (b) 26-neighborhood

2.1. Digital Curves and Surfaces

One of the fundamental topological property of Euclidean space is the Jordan theorem stating that a simple, i.e., non-self-intersecting, closed curve in two dimensions, or surface in three dimensions, partitions it in exactly two components: an interior and an exterior. Digital curves and surfaces obey the discrete version of the theorem [17]. Consider a two-dimensional binary grid where each pixel is either black or white. Let us call $N_\alpha$ the black or object pixels of a neighborhood $N_\alpha$ and $N_{\alpha'}$ its white or complement pixels. A 4-connected path of black pixels is a digital curve if and only if each of its pixels is simple, i.e., it verifies the following properties:

1. its $N_{\alpha'}$ neighbors are divided in exactly two 8-connected components, i.e., the interior and the exterior;
2. its $N_\alpha$ neighbors are 8-adjacent to these interior and exterior components.

In Fig. 3, the $N_\alpha$ neighborhood of the simple curve pixel $i$ contains pixels 1 to 8. Pixels 1, 2, 5 and 8 are black and form $N_\alpha$. Pixels 3, 4, 6 and 7 are white complements and represent $N_{\alpha'}$. This complement is indeed divided in two 8-connected components satisfying thereby Property 1. Pixel 4, 6 and 7 constitute one component, the interior for example, while the other component, i.e., the exterior, is composed only of pixel 3. The $N_\alpha$ neighborhood of pixel $i$ contains only two black pixels, 2 and 5. Both pixels are 8-adjacent to the interior and the exterior satisfying thereby Property 2. Figure 4 shows an example of a digital curve.

Similarly, a 6-connected path of black voxels in a three-dimensional binary grid is a digital surface if and only if each of its voxels is simple, i.e., it verifies the following properties:

1. its $N_{\alpha'}$ neighbors are divided in exactly two 26-connected components, i.e., the interior and the exterior;
2. its $N_\alpha$ neighbors are 26-adjacent to these interior and exterior components.

This concept of digital surface is essential to determine the grid resolution necessary to digitally represent a topological equivalent of a given domain geometry and will be used as a criterion for the background octree refinement.
The MA T skeleton is then formed by all the pixels or voxels that can be extended to such meshes [19]. Two polygonal cells are topological if they share an edge. This is equivalent to 4-adjacency in a two-dimensional regular grid. Similarly, two polyhedral cells are face-adjacent if they share a face. This is equivalent to 6-adjacency in a three-dimensional regular grid. Furthermore, two polygonal or polyhedral cells are vertex-adjacent if they share a vertex. This is equivalent to 8-adjacency in a two-dimensional regular grid and to 26-adjacency in a three-dimensional regular grid. Edge-adjacency and face-adjacency imply vertex-adjacency. Let $N_a$ with the subscripts $e$, $f$ and $v$ note the edge, face, and vertex-neighborhoods respectively. Using this notation, a polygonal cell of a two-dimensional edge-connected curve is simple if it verifies the following properties:

1. its $N_{ae}^v$ neighbors are divided in exactly two vertex-connected components, i.e., the interior and the exterior;
2. its $N_{af}^v$ neighbors are vertex-adjacent to these interior and exterior components.

In Fig. 6, the $N_a$ neighborhood of the simple quadree curve cell 1 contains cells 1 to 7 but not cells 8 and 9. Cells 2, 3, 5 and 6 are black and form $N_{ae}^b$. Cells 1, 4 and 7 are white and represent $N_{ae}^w$. This white complement is indeed divided in two vertex-connected components satisfying thereby Property 1. Cell 1 forms one component, the interior for example, while cells 4 and 7 constitute the other component, i.e., the exterior. The $N_{af}^b$ neighborhood of cell 1 contains only two black cells, 2 and 6. Both cells are vertex-adjacent to the interior and the exterior satisfying thereby Property 2.

Similarly, a polyhedral cell of a face-connected surface is simple if it verifies the following properties:

1. its $N_{af}^w$ neighbors are divided in exactly two vertex-connected components, i.e., the interior and the exterior;
2. its $N_{af}^b$ neighbors are vertex-adjacent to these interior and exterior components.

Figure 3: Simple curve pixel.

Figure 4: Simple closed 4-connected curve with two 8-connected components in the complement.

2.2. Digital Skeletons

The skeleton concept was introduced in continuous space by Blum as a shape descriptor [18]. Its discrete counterpart, the digital skeleton, is now used as a compact representation of binary shapes in image processing and pattern recognition. Conceptually, skeletonization transforms a two-dimensional object into its one-dimensional median line and a three-dimensional object into a two-dimensional median surface. Practically, digital skeletons are thin subsets of a binary shape that reflect its connectivity. Topological thinning algorithms are commonly used to obtain skeletons. Those algorithms erode layer by layer a digital object by turning off all pixels that can be removed without altering the topology of the original object. Thinning algorithms tend, however, to produce excessive erosion and have to be constrained. Alternatively, skeletons can also be generated using a Medial Axis Transform (MAT). Let $\delta$ be the distance to the boundary of any point inside an object. This distance transform $\delta$ measured in grid cells, can be computed for each pixel or voxel $i$ as follows:

1. Initialize $\delta_i^0$ to 1 for all interior and boundary pixels or voxels and to 0 all exterior ones;
2. Set $\delta_i^n = \delta_i^n + \min_{j \in N_a(i)} \delta_j^{n-1}$;
3. Iterate step 2 until $\delta_i^n = \delta_i^{n-1}$.

The MAT skeleton is then formed by all the pixels or voxels such that $\delta_i^n \geq \max_{j \in N_a(i)} \delta_j^n$. Figure 5 shows a square shape before and after skeletonization using a distance transform. The medial axis, or skeleton, is the locus of the centers of the maximal balls contained by an object and can, therefore, be used to extract boundary proximity and local domain thickness information.

2.3. Extension to Quadtrees and Octrees

Quadtrees and octrees can be considered as irregular polygonal and polyhedral meshes and the above definitions can be extended to such meshes [19]. Two polygonal cells are edge-adjacent if they share an edge. This is equivalent to 4-adjacency in a two-dimensional regular grid. Similarly, two polyhedral cells are face-adjacent if they share a face. This is equivalent to 6-adjacency in a three-dimensional regular grid. Furthermore, two polygonal or polyhedral cells are vertex-adjacent if they share a vertex. This is equivalent to 8-adjacency in a two-dimensional regular grid and to 26-adjacency in a three-dimensional regular grid. Edge-adjacency and face-adjacency imply vertex-adjacency. Let $N_a$ with the subscripts $e$, $f$ and $v$ note the edge, face, and vertex-neighborhoods respectively. Using this notation, a polygonal cell of a two-dimensional edge-connected curve is simple if it verifies the following properties:

1. its $N_{ae}^v$ neighbors are divided in exactly two vertex-connected components, i.e., the interior and the exterior;
2. its $N_{af}^v$ neighbors are vertex-adjacent to these interior and exterior components.

In Fig. 6, the $N_a$ neighborhood of the simple quadree curve cell 1 contains cells 1 to 7 but not cells 8 and 9. Cells 2, 3, 5 and 6 are black and form $N_{ae}^b$. Cells 1, 4 and 7 are white and represent $N_{ae}^w$. This white complement is indeed divided in two vertex-connected components satisfying thereby Property 1. Cell 1 forms one component, the interior for example, while cells 4 and 7 constitute the other component, i.e., the exterior. The $N_{af}^b$ neighborhood of cell 1 contains only two black cells, 2 and 6. Both cells are vertex-adjacent to the interior and the exterior satisfying thereby Property 2.

Similarly, a polyhedral cell of a face-connected surface is simple if it verifies the following properties:

1. its $N_{af}^w$ neighbors are divided in exactly two vertex-connected components, i.e., the interior and the exterior;
2. its $N_{af}^b$ neighbors are vertex-adjacent to these interior and exterior components.

Figure 5: Skeletonization of a square shape using a distance transform with a 4-neighborhood.

Figure 6: Simple quadtree curve cell.
Finally, the medial axis transform can also be extended to quadtrees [20] and octrees by modifying the distance transform to take into account variable cell sizes:

1. Initialize $\delta_i^0$ to half the cell size for all interior and boundary cells and to 0 all exterior ones;
2. Set $\delta_i^n = \delta_i^0 + \min_{j \in N_i^e(i)} \left( \delta_j^0 + \delta_j^{-1} \right)$;
3. Iterate step 2 until $\delta_i^n = \delta_i^{-1}$.

Let the largest box associated with each cell be the square, for a quadtree, or the cube, for an octree, of size $2\delta_i$. Let the largest box associated with each cell be the square, for a quadtree, or the cube, for an octree, of size $2\delta_i$ centered at the cell. A maximal cell is then a cell whose largest box is not completely contained by the largest box of any other cell, i.e., $\delta_i^n > \max_{j \in N_i^e(i)} \left( \delta_j^n - \delta_j - \delta_j \right)$, and the skeleton is the set of all maximal cells.

3. OCTREE GENERATION AND SKELETONIZATION

The skeletonization process described above is meaningful only if the octree is fine enough to resolve the significant features of the domain geometry. To ensure such a resolution, both local curvature and thickness refinement criteria are used. After presenting the required geometry definition, this section describes these criteria as well as the refinement process itself and the skeletonization of the resulting octree.

3.1. Domain Geometry Definition

The required input for the Cartesian background octree generation is a domain geometry definition. This definition must enable us to perform boundary intersection and inside-outside tests for the octree cells as well as closest point and local curvature interrogation. For the present project, triangulated boundary representations, from STereo Lithography (STL) files for example, were used. Since such triangulations simply serve as a support for geometric information, they do not have to be of high quality. They can be too fine but should not be too coarse or essential details will be lost. Ultimately, the user decides the level of details to be taken into account. Triangulations can also be dirty, i.e., not watertight. Dirt size should, however, be inferior to the size of the neighboring cells to make it invisible to the octree. One way to insure that is to make dirt size inferior to the size of the smallest possible cell. To accelerate intersection tests, the triangles are stored in an Alternating Digital Tree (ADT) [21] and, to improve accuracy and robustness, adaptive precision arithmetic is used [22]. Furthermore, Simulation of Simplicity (SoS) copes with degenerate intersection configurations such as barely touching entities [23]. Finally, curvature information can be given by the user along with the triangle vertices or it can be estimated directly from the triangulation [24].

Figure 7 shows the triangulation of an intricate mechanical part, a water jacket, downloaded from AVL [25]. The duplicate vertices of the STL file were first merged and the resulting triangulation was partitioned along sharp feature lines. The curvature tensor was then estimated separately for each patch. This geometry will be used throughout the present paper to illustrate the different steps of the algorithm.

3.2. Octree Refinement

The Cartesian background octree is generated using recursive non-conformal refinement of the input geometry bounding box. To resolve the significant features of the domain, a curvature-based criterion is first used. As in [26], each triangle of the geometry definition is associated with a target cell size based on the maximal curvature estimated anywhere on the facet. The higher the curvature, the smaller the target size. Octree cells are then simply refined until their sizes is smaller than the target sizes of the triangles they intersect. Note that only interference with the triangle bounding boxes are checked instead of actual intersection. The resulting octree will be slightly finer than strictly necessary but it greatly accelerates the refinement process.

A curvature criterion is, however, insufficient to ensure an adequate skeletonization. To find the medial axis, at least some cells must indeed be located strictly inside the domain. For example, small and almost planar gaps will not be adequately resolved using curvature-based refinement only. To avoid this problem, the octree could be refined until the boundary of the domain is discretized by a digital surface. However, in practice, such a refinement proved excessive for the present application. Digital surfaces as defined in Section 2 indeed introduce two criteria. The first one is a separation criterion requiring that the octree can be partitioned around each intersecting cell into an inside and an outside. The second criterion requires that the digital surface is thin enough for each intersecting cell to see this interior and exterior. This last criterion is mainly useful to triangulate digital surfaces using marching-cube algorithms. Applying this criterion tends to drive the refinement of octree cells intersecting non-axis aligned boundary surfaces up to the minimum size allowed. However, for our purposes, a thickness of two octree cells can be allowed as long as the surface still separates the inside from the outside of the domain. The second criterion has thus been dropped and the separation criterion slightly modified. Let the superscripts $b$ and $w$ note the boundary intersecting and non-intersecting cells respectively. Experimentally, it proved sufficient to require that the extended neighborhood $\bigcup_{j \in N_i^e(i)} N_i^w(j)$ of
Each boundary intersecting cell $i$ is partitioned in exactly two vertex-connected components, i.e., an interior and an exterior. This modified criterion results in a thicker discretized surface in exchange for a substantial reduction of the octree size. The actual savings depend on the domain geometry but, for the water jacket case, a reduction of the octree size by a factor of two was achieved.

Using these two refinement criteria, octree generation proceeds as follows:

1. Create a root cell encompassing the whole domain and flag it as intersecting the boundary;
2. Iterate to refine according to curvature:
   (a) Mark the set of cells interfering with boundary triangles whose target curvature-based size is inferior to their own current size;
3. Identify interior and exterior cells;
4. Iterate to refine according to separation criterion:
   (a) Mark the set of boundary intersecting cells invalidating the separation criterion;
   (b) Add to this set the $A^0_v$ neighborhood of the cells marked in step 4a;
(c) Refine the cells of the resulting set;
(d) Balance the tree to allow a difference of only one level of refinement between face-adjacent cells;
(e) Update intersection, interior and exterior flags.

Note that propagation to immediate neighbors and balancing are used to diffuse and smooth out the refinement. Furthermore, octree cells cannot be refined beyond a minimum size $h_{\text{min}}$. This may preclude adequate resolution of some geometric features, blunt some corners and fill some gaps. Adverse effects are, however, minimized if $h_{\text{min}}$ corresponds to the minimum mesh element size allowed during the actual adaptation process.

Figures 8(a) and 9(a) show the boundary intersecting cells of the octree generated for the water jacket geometry without and with the separation criteria. Those are relatively moderate size octrees counting 204777 and 217769 cells respectively. Practical geometries could however be much more complex and need bigger octrees. To accommodate such applications, explicit mesh-like data structures storing the vertices, edges and faces of the cells have been avoided in favor of an implicit tree structure storing only the parent and children for each cell. The size and position of the cells are then computed from the octree root. Such a data structure can, however, be very taxing during neighbor searches performed to verify the separation criterion. That is why binary coordinates were added to each cell to accelerate tree traversal and cell localization [27]. The resulting data structure is fast and compact.

3.3. Octree Skeletonization

Using the medial axis transform, skeletonization of the resulting octree is rather straightforward but may result in unwanted branches. Corners in the domain can indeed produce terminal skeleton branches going all the way to its boundary (Fig. 5). Those terminal branches are not desirable because the radius of the maximal balls tends to zero as we approach the boundary and do not always indicate an adequate local mesh size. For example, if the corner angle is very small then effectively the local mesh size should be small, i.e., the minimum allowable. However, if the angle is around 90 degrees or more then the size of the maximal balls is not a good indicator of the necessary local mesh size. Furthermore, skeletonization is sensitive to noise from the domain discretization by the octree, i.e., its stair-step boundary. It can produce very small branches terminating at noisy border cells. Those noise induced branches behave like corners with very wide angles close to 180 degrees.

A skeleton simplification to prune those unwanted branches is thus needed. The same strategy was used to prune corner and noise induced terminal branches. Following [28], the separation angle filters unwanted skeleton cells. By definition, this angle is formed by the vectors connecting an actual medial axis point to its closest boundary points (Fig. 10). It is approximated on skeleton cell edges by the angle formed by the vectors to the closest boundary point of each end vertex. For each skeleton cell, it is then taken as the minimum of its edge separation angles. The separation angle is big for branches resulting from sharp corners and small for blunt ones. Cells with a separation angle smaller than a given threshold are pruned from the skeleton. Best results were obtained with a threshold of 120 degrees.

Note also that the present medial axis transform is based on a chessboard distance measure on the octree and may thus suffer from digitization bias [30, 31]. The induced error is, however, acceptable for our purposes and is partly compensated by the pruning algorithm that uses Euclidean distances. Figure 9(b) presents the final skeleton for the water jacket geometry. When compared to the skeleton extracted from an octree refined using a curvature criterion only (Fig. 8(b)), the necessity of the separation criterion is clear. Without it, some important branches of the skeleton are missing. Note also that these skeletons are rather fat, and sometimes disconnected. Because the octree is refined only at the domain boundary, the center of the domain is coarsely discretized and the resulting skeleton is only a rough approximation. It is, however, sufficient because only an approximation of the local thickness of the domain is needed. The skeleton is fat in thick regions of the domain and svelte in thin ones. The relative error made on the size of the maximal boxes reflecting the local thickness of the domain is thus more or less constant and quite acceptable for our purposes.

4. Metric Extraction

By construction, the size distribution of the final octree cells adequately resolves the domain because it takes into account both boundary curvature and local thickness. This information is, however, isotropic and not optimal, i.e., it only gives the most constraining limit. Consider, for example, a long and narrow gap misaligned with the root cell. The octree leaf cells in the neighborhood of this gap reflect the thickness of the gap and not its length. Meshing such a gap with elements restricted to the size of these cells would be wasteful. This octree is, however, the ideal support medium for an anisotropic geometric metric map because it is already adapted to its expected variations. After a brief summary on Riemannian control metrics, this section presents the extraction of more efficient anisotropic geometry-based sizing information for such a map from the domain boundary triangulation and the octree skeleton.

4.1. Riemannian Metric

To control adaptation, an anisotropic control map must be used to prescribe not only the size but also the stretching and orientation of the mesh elements to be built. These specifications can be given as the metric of the transformation that
maps a perfect mesh element into a unit cube for hexahedral meshes or a unit equilateral tetrahedron for simplicial meshes. In three dimensions, this Riemannian metric is defined at every point of the domain by a symmetric positive-definite $3 \times 3$ matrix $\mathbf{M}$. This matrix can be factored as the product of a rotation matrix $\mathbf{R}$ and a diagonal scaling matrix $\Lambda$:

$$
\mathbf{M} = \mathbf{R} \Lambda \mathbf{R}^{-1} = \mathbf{R} \left( \begin{array}{ccc} h_1^2 & 0 & 0 \\ 0 & h_2^2 & 0 \\ 0 & 0 & h_3^2 \end{array} \right) \mathbf{R}^{-1} \quad (1)
$$

where $h_1$, $h_2$ and $h_3$ are the target element sizes along the three axes of the local coordinate system defined by $\mathbf{R}$. Such a size specification map can be given analytically or constructed from a posteriori error analysis, from the geometrical properties of the domain, as in the present paper, or from any other user defined inputs. An isotropic size specification map reduces to an identity matrix multiplied by $\tilde{h}^2$ where $\tilde{h}$ is the desired element size. Whatever it’s origin, the control metric contains information on the prescribed size, stretching and orientation of the mesh to be built as an anisotropic metric field. See [32] as well as [33] and [1] for a more complete discussion on metrics.

Using metrics promotes decoupling of the actual adaptation algorithm from the target mesh specifications. Algorithm traditionally used for solution-based adaptation through global remeshing or local mesh modifications can then be used for geometric adaptation.

### 4.2. Extracting Thickness and Curvature Data

The adaptation metric has to take into account both the local thickness and curvature of the domain. Thickness information can be extracted from the octree skeleton cells using the distance transform $\delta$. Curvature information, on the other hand, can be extracted from octree cells intersecting the triangulated domain boundary. To construct the geometric metric, these two types of information must be combined and diffused in the whole domain.

For each octree cell, $\delta$ is an approximation of the distance to the closest domain boundary. By definition, it is also an approximation of the radius of the maximal ball centered on a skeleton cell and the local thickness $\tau$ of the domain is equal to twice this radius. However, at non-skeleton cells, $\tau$ is not related to the local value of $\delta$ but rather to the thickness associated to the closest skeleton cell. Computing $\tau$ at those non-skeleton cells could thus be reduced to searching for the closest skeleton cell. To get a smoother distribution, $\tau$ can also be simply diffused from the skeleton cells to the rest of the octree using a Laplacian operator:

$$
\tau^{n+1}_i = \tau^n_i + \frac{\sum_{j \in N_j(i)} (\tau^n_j - \tau^n_i) / l_{ij}}{\sum_{j \in N_j(i)} 1 / l_{ij}} \quad (2)
$$

where $n$ is an iteration counter and $l_{ij}$ is the Euclidean distance from the center of cell $i$ to the center of cell $j$. This latter approach was used. Dirichlet conditions were imposed at skeleton cells where $\tau$ was set to twice the value of the distance transform $\delta$. To simplify the metric interpolation and minimize memory requirements to store the background octree, the metric, and therefore $\tau$, was considered constant over each octree cell. The octree resolution proved experimentally sufficient since the generation process locally refined cells in expected high gradient regions.

After its diffusion, the local thickness $\tau$ can be combined with the curvature tensor at each boundary intersecting cell to give the following metric:

$$
\mathbf{M} = \left( \begin{array}{ccc} h_\tau^2 & 0 & 0 \\ 0 & h_{\kappa_1}^2 & 0 \\ 0 & 0 & h_{\kappa_2}^2 \end{array} \right) \left( \begin{array}{ccc} \tilde{n} & \tilde{t}_1 & \tilde{t}_2 \end{array} \right)^T \quad (3)
$$

where $\tilde{n}$ is the unit normal to the boundary while $\tilde{t}_1$ and $\tilde{t}_2$ are the unit tangents in the direction of the principal curvatures $\kappa_1$ and $\kappa_2$ evaluated at the boundary point closest to the center of the intersecting cell. The directional target sizes $h_\tau$, $h_{\kappa_1}$ and $h_{\kappa_2}$ are computed as functions of $\tau$, $\kappa_1$ and $\kappa_2$. To diffuse this metric tensor throughout the domain, a term by term Laplacian operator is used with intersecting cell values acting as Dirichlet conditions:

$$
\mathbf{M}^{n+1}_i = \mathbf{M}^n_i + \frac{\sum_{j \in N_j(i)} (\mathbf{M}^n_j - \mathbf{M}^n_i) / l_{ij}}{\sum_{j \in N_j(i)} 1 / l_{ij}} \quad (4)
$$

where the notation of Eq. (2) is used. Finally note that, at cells located outside the domain, the metric tensor is set to the identity matrix times the squared inverse of the prescribed size at infinity, usually chosen as the size of the domain bounding box.

### 5. APPLICATION

The application we propose to explore is geometric adaptation for superposition or grid-based hexahedral mesh generation methods [35–42]. Such methods overlay an initial mesh, usually Cartesian, on the domain geometry and keep only its interior elements. The boundary of the resulting mesh has then to be fitted to the domain through cutting, projection or isomorphism. These methods are robust but often generate poor quality elements at the boundary of the domain because of the misalignment of the initial mesh [43,44]. Furthermore, the body fitting step of grid-based methods can be performed reliably only if local mesh density is sufficient to capture the features of the domain geometry [26]. The present metric construction algorithm combined with an appropriate adaptation tool is ideally suited to generate the initial mesh required by such methods.

Relocation algorithms are commonly used for solution-based adaptation and essentially smooth the mesh in the target anisotropic metric space. The particular algorithm used here is based on a spring analogy that considers the mesh as a network of vertices linked by springs. The optimal position of each vertex $i$ is computed iteratively by the following length equidistribution formula:

$$
\vec{x}^{n+1}_i = \vec{x}^n_i + \omega \sum_{j} k_{ij}^n (\vec{x}^n_j - \vec{x}^n_i) / \sum_{j} k_{ij}^n \quad (5)
$$

where $n$ is an iteration counter, $\omega$ is a relaxation factor, $j$ denotes all vertices sharing an edge with vertex $i$ and the spring rigidity constant $k_{ij}$ is the metric length of edge $ij$. The adaptation metric proved experimentally sufficient since the generation process locally refined cells in expected high gradient regions.
divided by its Euclidean length. The metric length of an edge $i,j$ is given by:

$$ l_{ij} = \int_0^1 \sqrt{(\vec{x}_j - \vec{x}_i)^T M(\vec{x}_i)(\vec{x}_j - \vec{x}_i)} \, dt \quad (6) $$

where $\vec{x}_i = \vec{x}_i + t (\vec{x}_j - \vec{x}_i)$. This metric length is integrated numerically using a simple trapezoidal rule with $M$ being interpolated on the background octree. See [45] for more details on relocation methods.

Figure 11 presents an initial structured hexahedral mesh for the water jacket geometry with and without geometric adaptation. This cubic overlay grid counts $100 \times 100 \times 100$ hexahedra. Without adaptation, only 23071 elements are located inside the geometry and are retained for the body-fitting step of the grid-based method. Fine details cannot be captured adequately by such a Cartesian mesh unless the resolution is drastically increased. Global refinement would be very wasteful while local refinement is usually isotropic and resort to non-conformal transition elements. However, with adaptation by point relocation and a geometric metric, those details are easily resolved. The adapted overlay mesh counts 116430 elements inside the geometry and those elements are better aligned with the boundary. The present geometric adaptation strategy should thus make the overlay meshes less sensitive to misalignment problems typical of grid-based methods. The anisotropy introduced by the geometric metric is also much more efficient than the usual Cartesian refinement, global or even local, for long and narrow regions. Finally, although the general shape of the domain can already be recognized, the actual boundary of the model has yet to be recovered. The present adaptation process should, however, greatly facilitate the body-fitting process.

Figures 12 and 13 present further adaptation examples for geometries found on the Internet [29, 34]. In addition to the adapted overlay mesh to be used by a grid-based method, the triangulated domain, the intersecting cells of the background octree and the corresponding skeleton are also presented. Note how much more efficient is the adapted anisotropic mesh compared to the isotropically refined octree.

As illustrated by these examples, surprisingly good results can be obtained using only point relocation to adapt the over-
lay mesh. However, for geometries with very large length scale variations, local mesh refinement may be needed. To avoid hanging-nodes, a conformal all-hexahedral refinement method based a pillowing or shrink and connect strategy can be used [46]. Consider, for example, the toy dinosaur [34] presented in Fig. 14. Again, the triangulated domain, the intersecting cells of the background octree and the corresponding skeleton are presented in addition to the adapted overlay mesh. This adapted mesh was generated from a $5 \times 5 \times 5$ cubic grid encompassing the domain that was coarsely refined using the shrink and connect strategy. Each resulting hexahedron was then diced in 8 and the point relocation algorithm smoothed the resulting mesh in the target geometric metric. The final mesh counts 48954 elements located inside the geometry. Figure 15 plots a cut of the complete overlay mesh before the removal of non-interior elements and shows how the geometric metric drives the combined point relocation and local refinement algorithm to effectively carve the model geometry out of the initial cube. Although the elements located outside the geometry are very distorted, the geometric metric, and the smoothing process used to generate it, gives very good elements inside the geometry itself without any inverted cell.

Finally note that, for all the examples presented in this section, the following size functions were used: $h_r = \pi^2/3$ and $h_{s_1,s_2} = \pi^2/8 |s_1 \cdot s_2|$. Furthermore, a relaxation factor $\omega$ of 0.5 was necessary in Eq. 5 to stabilize the point relocation algorithm.

### 6. CONCLUSION

A new method to construct geometry-based adaptation metrics from triangulated domains was introduced in the present paper. These anisotropic metrics are computed using local domain curvature, estimated from its triangulated boundaries, as well as thickness. Digital topology theory is used to extract this thickness from the domain skeleton on a Cartesian background octree used as a support medium for the anisotropic metric. Applications illustrated the effectiveness of this approach for hexahedral mesh adaptation.

The present geometric metric can, however, be used on any other mesh type, tetrahedral or hybrid for example, as long as the adaptation algorithm uses metrics to specify its target. It could also be combined with other specifications based on user experience. For example, in computational fluid dynamics applications, the metric could be modified to take into account boundary layers around solid walls. Although, it could never replace a solution-based metric computed using a posteriori error estimators, such a metric is, however, invaluable to generate and adapt initial meshes when no solution is yet available.

### 7. ACKNOWLEDGMENTS

The authors would like to thank NSERC for its financial support. Many thanks also to Julien Dompierre for his insight on metrics and mesh adaptation.
REFERENCES


Figure 15: Toy dinosaur — Cut of the whole adapted overlay mesh.


Session 3B
Adaptivity
FULLY AUTOMATIC ADAPTIVE MESH REFINEMENT INTEGRATED INTO THE SOLUTION PROCESS

Joseph R. Tristano, Zhijan Chen, D. Alfred Hancq, Wa Kwok

ANSYS Incorporated, Canonsburg, PA U.S.A.

{joe.tristano,james.chen,al.hancq,wa.kwok}@ansys.com

ABSTRACT

Finite element analysts and designers need to feel confident in the results of their analyses before sending a product to prototype or production. Mesh discretization can greatly influence the desired results. In this paper, we present a framework for adaptive mesh refinement to obtain FEA results with a desired accuracy. The process involves adaptively refining the mesh based on solution error norms until the result desired converges to a certain accuracy. The adaptive refinement/meshing process must be fully automatic and very robust. We present an exhaustive method to create a fully automatic and integrated process that takes advantage of many of the mesh refinement and mesh optimization algorithms found in literature. The results of the process provide the user with the desired accuracy in the smallest number of iterations possible.

Keywords: h-adaptivity, adaptive mesh refinement, adaptive finite element

1. INTRODUCTION

Finite element analysts and designers need to feel confident in the results of their analyses before sending a product to prototyping or production in order to save their company’s time and money. This is especially true for users who are not experts in FEA. The discretization of the model intended for analysis can greatly influence the outcome of their desired results. In order to feel confident with the desired results, we have developed a fully automatic adaptive mesh refinement process that is integrated into the solution process. The mesh is refined based upon error norms and other information output by the solver. In order to make the process reliable and robust, multiple methods for refining the mesh have been implemented. The process also needs to be efficient, i.e., the process needs to mark enough nodes for refinement such that the iterations are not wasted in the solver. The importance of this work lies in the method to integrate the tools and procedures found in literature into a fully embedded automatic and robust adaptive solution process.

The remainder of this paper will first discuss the overall framework of the process. We will then discuss the previous work in the area on which the building blocks for the process were built. We will then go into detail of the sub-processes within the process such as methodology for selecting nodes and elements for refinement and the refinement process itself. We will finally show some examples and discuss areas for future work. This paper will focus on refinement of tetrahedral meshes. However, the method can easily be extended to other element types.

Figure 1 Overview of Adaptive Solution Process
1.1 Process Overview

The adaptive solution process is an iterative process illustrated Figure 1.

It is clearly seen that this adaptive solution process does not lend itself to failure. Various techniques are used to protect the refine process from producing poorly shaped elements. It is imperative that the mesh generated from the refinement process is of adequate quality.

The process is as follows. The initial solve takes place. The solver driver then determines which nodes must be refined and which nodes must be excluded from refinement (to prevent refinement of singularities). Depending on the amount of nodes marked, two paths may be taken. The first path is the template based tetrahedron refinement based on Staten’s work. The second path uses a Pseudo Global Refinement (PGR) method that: deletes the tetrahedron, refines the surface mesh and performs some optimization of the surface mesh, re-meshes with tetrahedron and refines any marked interior nodes using templates. Extensive tetrahedron optimization is performed using conventional methods of B.Joe [1] for each method of refinement. If the conventional optimization methods fail, more specialized quadratic tetrahedron optimization methods are called (Kwok et al. [2]). This process iterates until the solution has converged or the number of adaptive iterations has been met.

If either method fails, various styles of the PGR algorithm work to refine the model to create a high quality tetrahedron mesh.

The processes communicate via data passed through COM Interfaces. These interfaces allow us to maintain each procedure of the process independently. The interfaces are implemented in several components (DLL’s). The solver component solves the FE model. The driver component determines what elements should be refined based on analysis type specific criterion. This component communicates between the solver and mesh refiner and determines when the adaptive loop should start. This component also charts the progress of adaptivity for the user, which enables the user to track the convergence of the result per refinement iteration along with displaying the number of nodes and elements for each iteration. The mesh refiner asks the solver for a mesh object and the nodes and elements to be refined and excluded from refinement. This component is self-encapsulated and determines whether templates or PGR refinement should be executed. If the quality of the output mesh is not adequate, more refinement and mesh optimization are performed to increase the quality to acceptable levels. This componentization also enables quick prototyping and implementation of new refinement algorithms, solvers, and error estimators.

1.2 Previous Work

The literature contains many works regarding error estimates [3],[4] and refinement methods[5],[6], however, the methods used to fully automate and integrate these tools into a robust fail-safe algorithm have been trivialized. When dealing with complex real world geometry and quadratic tetrahedron, this process is never trivial.

1.2.1 Adaptive Refinement

Diaz-Morcillo [7] presented an adaptive mesh refinement process for electromagnetic problems. Since the elements are usually linear and the geometry is usually simple for this class of problems, the framework for the algorithm is allowed to be quite simple: solve, estimate error, refine. Jones [8] also describes a method very similar to Diaz’s. His work also concentrates more on the refinement algorithm’s than creating a fail-safe process that can run on real world geometry.

1.2.2 Template Based Refinement

We have chosen to use Staten’s [6] (Figure 2) template based refinement over existing methods of refinement. This method was chosen over Delaunay[5] refinement because it has shown the ability to maintain overall mesh quality after successive refinements.

![Figure 2 Staten’s Template Based Refinement](image)

2. ADAPTIVE REFINEMENT SUBPROCSESES

2.1 Methodology for Choosing Candidates for Adaptive Refinement

Adaptive H refinement is supported for structural, thermal, and modal analysis. The result quantity $\hat{\phi}$, the expected accuracy $E$ (expressed as a percentage), and the region R on the geometry that is being subjected (scoped) to adaptive analysis may be selected. The user-specified accuracy is achieved when convergence is satisfied as follows:

$$100 \left( \frac{\phi_{i+1} - \phi_i}{\frac{1}{2}(\phi_{i+1} + \phi_i)} \right) < E, \ i = 1,2,3,...n \ (in \ R) \quad (1)$$

Elements will be flagged for refinement on some basis measure. The criterion for which elements and nodes are selected for adaptive refinement depends on analysis type and on what results quantities are requested. In addition, depending on analysis type and the requested result type of $\phi$, nodes may be flagged for refinement to augment the element list. This is done to help ensure that true convergence is achieved on the result $\phi$. 
2.1.1 Element Selection Methodology, Structural and Thermal Analysis

A basis measure must be employed to identify poorly shaped elements that are causing analysis inaccuracies. The procedure to select the elements for refinement, aside from the error calculation itself, is the same for both structural and thermal analysis and will be discussed first. For structural analysis, a displacement based error measure developed by Zienkiewicz-Zhu[3] is used as this basis. Essentially an elemental stress error is calculated from the difference between the average nodal stress and the element nodal stress. This stress error is then integrated over the element volume to calculate an energy error that is then used as the structural error basis. This calculation is performed over the element set and can be seen mathematically by [9]:

\[ \{\Delta \sigma \}' = \sigma'_{\text{av}} - \sigma'_{\text{e}} \]

where:

\[ \{\Delta \sigma \}' = \text{stress error vector at noden of element} \]

\[ \sigma'_{\text{av}} = \text{averaged stress vector at noden} = \frac{\sum_{N} \sigma'_{i}}{N} \]

\[ N = \text{number of elements connected to noden} \]

\[ \{\sigma'_{i}\} = \text{stress vector of noden of element} \]

Then for each element:

\[ e_{\text{e}} = \frac{1}{2} \int_{\text{vol}} \{\Delta \sigma \}'^T \{D\}^{-1} \{\Delta \sigma \} d(\text{vol}) \]

where:

\[ e_{\text{e}} = \text{energy error for element} \]

\[ \text{vol} = \text{element volume} \]

\[ \{D\} = \text{stress–strain matrix of the element} \]

\[ \{\Delta \sigma \}' = \text{stress error vector} \] (2)

A similar calculation for temperature-based problems using elemental heat flux was given by Huang and Lewis[4] and is used as a basis for thermal energy error.

After all the element errors are collected, a cutoff range technique is employed where the elements with the largest error are flagged for refinement. A critical value is calculated and elements with an error above this value will be flagged for refinement according to the equation:

\[ e_{\text{req}} = e_{\text{max}} - \beta (e_{\text{max}} - e_{\text{min}}) \]

where:

\[ e_{\text{req}} = \text{minimum element error required to be flagged for refinement} \]

\[ e_{\text{max}} = \text{maximum element error in element set} \]

\[ e_{\text{min}} = \text{minimum element error in element set} \]

\[ \beta = \text{cutoff factor, } \beta \in \mathbb{R}^+ \text{ default of .75} \] (3)

As can be seen from the equation above, a cutoff factor of zero would cause only elements with the highest error to be flagged while a factor of one would flag all elements for refinement. At first glance, the default \( \beta \) of .75 would seem to allow a disproportionately large number of elements to be flagged but experience has shown this value to be valid. This is due to a typically large gradient in the distribution of element error throughout the element set. Although highly dependent on the loading and geometry, typical percentages of elements flagged for refinement range from 5 to 20 percent.

2.1.2 Element Selection Methodology, Modal Analysis

In selecting elements for modal refinement, a different basis mechanism as well as different cutoff technique is employed. The basis function is simply largest element volume. The element set is sorted on that basis and then a cutoff element technique is employed so that a set percentage of the largest elements are flagged for refinement. Typically, 25 percent of the elements will be flagged for refinement but this value is controllable by the user.

2.1.3 Node Augmentation Selection methodology

Although the element selection methods described above perform well in obtaining an accurate global solution, node augmentation based on the result may be required to ensure that a local converged result is achieved. This is applicable to results where local refined meshes are required to achieve an accurate solution. Consider the case where \( \phi \) is based on equivalent stress and a stress concentration exists. Consider also that in that region the element error is low enough compared to another region such that the other region in the model is flagged for refinement. Thus, as a result, the change in \( \phi \) will be likely small enough to satisfy the requested accuracy \( E \), although true convergence has not been reached. To aid in this, a list of nodes on the region \( R \) will be flagged for refinement that have values near \( \phi \) and thus ensures that a false convergence will not occur. This is node augmentation is only applicable to certain type of results where this false convergence can occur and includes:

- Structural stresses and strains
- Structural post tools based on stress and/or strain
- Structural contact results
- Thermal heat fluxes

Nodes are selected for refinement based on a cutoff range technique. However, since convergence may be applied to either the minimum or maximum of \( \phi \), and in addition, \( \phi \) may be positive or negative, logic is required to handle each permutation. Each node in the set will test the following logic statement and will be flagged for refinement if the statement is true:

\[ \phi < 0 \land (\alpha \phi > \phi) \]

\[ \phi \geq 0 \land (\alpha \phi < \phi) \]

where:

\[ \phi = \text{result quantity to be converged upon} \]

\[ \phi_n = \text{result quantity at node } n \]

\[ \alpha = \text{cutoff factor, } \alpha \in \mathbb{R}^+, \text{ default of .8} \] (4)
Care must be taken when compiling this node list. Artificial singularities may exist in the model due to the applied loads that could cause a divergent refinement series. A list of boundary conditions known to cause these singularities is compiled and any nodes in the region R that exist on these boundaries are excluded as candidates for node augmentation. Note that geometric singularities may exist in model that the program cannot detect and thus care must be taken by the user to ensure that a proper region R is selected for the result quantity where convergence is of interest.

2.2 Refinement Process

2.2.1 Refinement Driver

After the nodes and elements are marked for refinement, the refinement driver loads the refine component. Data, such as the previous mesh, marked and excluded nodes and elements, and the BRep are passed into the refine component via a COM Interface (Figure 3).

```csharp
interface IDGAnrRefine : IDGBase
{
  [property("method DefineMesh")],
  HRESULT DefineMesh([In] IDGBase& pP, IDGBase& pV, IDGBase& pF);
}
```

Figure 3 COM Interface for Refinement Component

2.2.2 Node Marking and Element Improvement

The mesh is converted into a local data structure and the appropriate nodes are marked. During the node marking leg, the algorithm may determine that some quadratic tetrahedron may invert during the refinement process. At this time a few things may happen: Mid-side nodes on element edges interior to the topological body are perturbed, poor boundary elements may be deleted, nodes may be unmarked.

When bending interior edges, we examine the diagonal of the tetrahedron that is chosen to be refined that will invert the children tetrahedron. The mid-side node is perturbed until the children elements will have an acceptable shape.

Figure 4 Bending Interior Edges

There may be cases where there is not enough room on the interior of the model to bend an interior element edge to create an acceptable quality element (Figure 4). In this case, we determine if the tetrahedron is a poor element that lies on the boundary of the topological body. If this poor element lies on the boundary, a few operations may be performed to either modify the tetrahedron or simply delete it. Some of these operators include:

- Splitting the longest edge to create better-shaped elements
- Swapping element edges to create a better configuration
- Moving interior corner nodes to improve resulting element shapes
- Deleting the element if it is a kite or cap on the boundary
- Modifying a mid-side node of the elements on the boundary.

If all of these methods fail to produce a valid configuration, the nodes marked on the poor element will be unmarked for refinement. Although the driver component wanted the nodes to be refined, the unmarking of nodes has been proven to be an acceptable practice via empirical studies.

2.2.2 Template Based Refinement

The template base refinement follows Staten’s work with slight modifications. There are cases where after some refinement that the diagonal to refine the tetrahedron upon will create invalid elements. In this case, we again try to bend interior mid-side nodes to create an acceptable quality element.

2.2.3 Pseudo Global Refinement (PGR)

In the cases where template based refinement fails, we try a new method called Pseudo Global Refinement (PGR). The PGR method is an innovative technique that utilizes template based surface mesh refinement and local remeshing techniques to perform mesh refinement during an adaptive solution process.
The PGR method is comprised of the following steps.

1. Remember marked interior node locations ("memory nodes") and delete tetrahedron
2. Split boundary edges that have very high transitions as to improve the quality of the resultant mesh
3. Improve quality of surface mesh via triangle quality swapping
4. Refine marked nodes on surface using template refinement and clean surface mesh
5. Fill body with tetrahedron
6. Refine Interior Nodes closest to “memory nodes”
7. Optimize Mesh

2.2.3.1 PGR Surface Mesh Aspect Ratio Adjustment

To obtain a successful refinement using the PGR method, the surface mesh is massaged to prevent poor element quality later in the process (Figure 5). This preventative method reduces high transitions in the mesh by looping through marked nodes and determining if any edge, $E_i$ at node $M$, with length, $L_{i \text{ at node } M}$, exceeds the ratio $\beta$ when compared to the smallest edge length at the node, $L_{\text{min at node } M}$. If this edge ratio exceeds $\beta$, the edge is split to reduce the high transition. In practice, we have found that $\beta=10.0$ is a suitable number.

![Figure 5 Splitting High Transition Edges](image)

2.2.3.2 PGR Surface Mesh Optimization

We perform topological optimization of the mesh along with quality optimization of the mesh. The quality optimization of the mesh is required for quadratic elements. An element may have acceptable quality when it is linear but adding mid-side nodes to the element may invalidate it (Figure 6). Therefore, the quality swap is extremely important. A quality swap is one in which a swap is performed if the quality of the elements will improve in the new configuration. This step provides a high quality surface mesh to send to the template based surface mesh refinement.

![Figure 6 Acceptable Linear Element Becomes Invalid with the Addition of Mid-Side Nodes](image)

2.2.3.3 PGR Template Surface Mesh Refinement

PGR Template triangle refinement uses Staten’s method [6] of refinement to refine the marked nodes (Figure 7). This refinement is then followed by topological optimization of the triangles [10] and mixed Laplacian/Optimization smoothing[11] of the refined surface mesh.

![Figure 7 Template Triangle Refinement](image)

2.3 Tetrahedron Optimization Methods

After the topological body bound by the refined surface mesh is filled with tetrahedron, mesh optimization is performed on the mesh. B. Joe’s [1] tetrahedron optimization techniques have been implemented for the initial optimization leg of the code. For cases where the mesh cannot be improved by Joe’s techniques, more advanced and specialized tetrahedron optimization operators are used. These methods consist of the same methods used to improve tetrahedron quality during the node marking leg of the process.

Improving poor boundary element tetrahedron by reposition their mid-side nodes is one of the most common specialized tetrahedron operators called in the process (Figure 8). The operator is quite simple. The smallest or inverted angle is found on the boundary face. The unconstrained edge is determined, i.e., the edge that does not lie on a topological edge is found. The direction of the edge, $\vec{E}$, is crossed with the surface normal, $\nabla_{\text{surface}}$, to determine the direction, $\vec{F}_{\text{move}}$, in which the mid-side node of the moving edge should move. This process moves the node a small $\delta$ based on the distance between the mid-side nodes of the adjacent edges until the quality of the tetrahedron adjacent to the edge are acceptable.
3. EXAMPLES AND FUTURE WORK

3.1 Example

3.1.1 Pressure Vessel

The following example illustrates our process on a moderate complexity CAD geometry of a pressure vessel. The intersections of the fillets in the model contain many sliver surfaces, which have been defeatured via “Mesh Based Defeaturing” [12]. The loading condition is shown in Figure 8. The initial mesh (Figure 10) was intentionally set to the coarsest setting possible to show the robustness of the refinement process. It is clearly seen that the initial mesh contains many high aspect ratio tetrahedron. The model was set to adapt on the maximum normal stress of the whole model with a convergence criterion of 7% allowable change in the maximum normal stress (Figure 11). After four iterations, the model converged to .5% change in maximum normal stress (Figure 12, Table 1). The large increase in the number of nodes is due to the massaging of the surface mesh during PGR. PGR allows the complex fillets to be refined while maintaining the overall quality of the mesh.
3.1.2 Plate with Hole

The next example illustrates an academic problem of a plate with a hole. The interesting item with this model is the scoping of results to the top surface (region R as discussed above). As stated above, the scoping control limits the marking of nodes to nodes lying on the scoped surface (Figure 17). The model converges to 1.88 % change in maximum equivalent stress in four iterations (Figure 16, Table 2).

<table>
<thead>
<tr>
<th>Solution Number</th>
<th>Normal Stress (Pa)</th>
<th>Change (%)</th>
<th>Nodes</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.3980E+07</td>
<td></td>
<td>2080</td>
<td>1056</td>
</tr>
<tr>
<td>2</td>
<td>1.9378E+07</td>
<td>32.062</td>
<td>3500</td>
<td>1843</td>
</tr>
<tr>
<td>3</td>
<td>2.0886E+07</td>
<td>7.8</td>
<td>35255</td>
<td>22418</td>
</tr>
<tr>
<td>4</td>
<td>2.0994E+07</td>
<td>0.51585</td>
<td>123451</td>
<td>83988</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution Number</th>
<th>Equivalent Stress (psi)</th>
<th>Change (%)</th>
<th>Nodes</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1088.4</td>
<td></td>
<td>438</td>
<td>174</td>
</tr>
<tr>
<td>2</td>
<td>1430.5</td>
<td>27.161</td>
<td>2388</td>
<td>1183</td>
</tr>
<tr>
<td>3</td>
<td>1502.9</td>
<td>4.9363</td>
<td>12494</td>
<td>7543</td>
</tr>
<tr>
<td>4</td>
<td>1531.6</td>
<td>1.8895</td>
<td>35341</td>
<td>22925</td>
</tr>
</tbody>
</table>
This process is quite robust. However, as with any process, there is always room for improvement. Future work in this area may consist of:

- Determining better heuristics for when to choose PGR over Template refinement
- Using an optimization method to place the mid-side nodes when bending interior edges or moving the mid-side node of a poor surface tetrahedron
- Developing new specialized tetrahedron cleanup operators
- Improve refinement for thin models when the aspect ratio of the opposite side of the model is much larger than the side being refined (Figure 18)

Figure 18 Cross-section of thin model with high aspect ratio tetrahedron

REFERENCES

A NEW TYPE OF SIZE FUNCTION RESPECTING PREMESher ENTITIES

Jin Zhu

Fluent, Inc. 1007 Church Street, Evanston, IL, U.S.A. jz@fluent.com

ABSTRACT

This paper describes the creation of a new type of size function – the mesh size function that honors the existing mesh on premeshed geometry entities and radiates the mesh sizes from the premeshed source entities to the attached entities – from the technology of using background overlay grids. The creation of faceted meshes from premeshed source entities (i.e. edges or faces) is presented in a more general way, which allows the use of existing procedure of size function implementations. The introduction of the mesh size function has greatly enhanced the capabilities of the three types of size functions that were already available (including a fixed size function, a curvature size function and a proximity size function) and provided nice solutions to the situations where the old size functions did not work desirably. Meshing results of the new size function with controlled mesh sizes are given.

Keywords: mesh generation, size control, size functions, background grid.

1. INTRODUCTION

As everyone knows, the mesh size control is very critical to mesh quality and to the successful field simulations using the generated mesh. The mesh sizes need to catch local small geometry features, and then are smoothly transitioned into the nearby areas of the geometry unless they reach the given size limit. Various methods have been used by different researchers to set up size functions to automatically detect the geometric features and put appropriate mesh size at desired locations, thus eliminate the need of manually locate the local features of the geometry and mesh these entities by desired sizes [1-7]. The background overlay grid size functions that were developed in our previous work illustrated satisfactory performance in mesh size control [8]. However, sometimes creating a mesh that is radiated in a controlled manner from some premeshed boundaries of the domain can also be an efficient way of obtaining desired mesh transition and gradation. The mesh on premeshed boundaries can come from manual operations as desired, but more often it comes from the meshing results of other size functions, or even from imported geometry. In the following, we will list some problems that would be encountered during the meshing processes by using the size function capabilities that had existed, and demonstrate the necessity of creating a new mesh size function.

In the first place, let’s take the case of meshing a 2D airfoil as an example, as displayed in Figure 1. The initial mesh size of a fixed size function is defined as a constant value specified by the user. This works for many cases where uniform sizes at the location of the sources are desired. However, for other cases such as the one in Figure 1, it requires that non-uniform mesh sizes be used at the location of the sources. It is not acceptable to define a constant size along all the airfoil edges. It is necessary for this problem to cluster meshes at the leading edge, at an approximate shock location along the upper surface, and at the trailing edge. Using the current size function implementation, it is required to define at least 6 size functions to cluster elements at the desired locations and then grow the elements away from the airfoil surfaces. One size function uses vertex a as source, and other 5 size functions use edges bc, de, fg, hij and kl, respectively, as sources. These edges can be part of the airfoil surfaces. This is not a very convenient way and takes some trial and error to get desirable mesh clustering at the airfoil surfaces.

A much more convenient and better approach will be to mesh the edges (that represent the surfaces of the airfoil) separately and cluster the edge nodes using the standard edge meshing bunching functionality. Then, have a size function to use those edges as sources and the existing (and varying) mesh sizes at those edges as the initial mesh size. The mesh elements are then allowed to grow using the user specified...
ratio and size limit. Note that this concept can be extended into 3D meshing, in which case the size function will take the existing mesh on source faces as the initial size.

For another example, the geometry in Figure 2(a) contains two volumes, exterior volume volume.1 and interior volume volume.2. Four size functions are created (growth rate = 1.2, size limit = 2, cells-per-gap = 3 for proximity size function and angle = 25 for curvature size function) and attached to the geometry as follows:

- Proximity size function sfunc.1:
  source: volume.1
  attachment: volume.1
- Curvature size function sfunc.2:
  source: all faces of volume.1
  attachment: volume.1
- Proximity size function sfunc.3:
  source: volume.2
  attachment: volume.2
- Curvature size function sfunc.4:
  source: all faces of volume.2
  attachment: volume.2

Suppose volume.1 is meshed first and volume.2 second. Then the common face between volume.1 and volume.2 is meshed according to the size functions attached to both volumes. Since the size functions attached to volume.1 give smaller mesh sizes than the size functions attached to volume.2, so the mesh on the common face is dominated by sfunc.1 and sfunc.2 (actually by proximity size function sfunc.1), instead of sfunc.3 and sfunc.4. However, when meshing volume.2, the mesh size is purely controlled by the two size functions attached to volume.2, which will conflict with the meshes generated on the common face, thus causes size jump inside volume.2 or even generates un-usable meshes. (See Figure 2 (b))

Since size functions attached to the upper topology will also affect its lower topologies, same problem will occur even if volume.2 is meshed first. Here the key issue is that the mesh size on the common face is controlled by four size functions from two sharing volumes, whereas the mesh size in each volume is controlled only by two size functions associated with it.

The mesh size on the common face of above model in Figure 2 is nearly constant. For the similar geometry shown in Figure 3 where the common face has varying sizes due to the changing curvature and gap distance from volume.2, same mismatching of mesh sizes will be encountered.

As one can expect that it is hard to determine which size functions are the dominant ones that give the smallest mesh size on the entities to be meshed. To avoid a poor mesh being generated, a workaround for the above scenario is to attach the size functions in a crossing way, that is, also attach size functions sfunc.1 and sfunc.2 to volume.2 and, similarly,
For some mesh schemes, the mesh sizes determined by size function have to be adjusted so that the scheme can work. For example in a mapped face, the mesh sizes on opposite paired edges have to be increased or decreased so that their mesh intervals match each other. In Figure 4, there are two faces, face.1 and face.2, connected through common edge $ab$. The edge at the left-most side is used as source edge of a fixed size function, and a start size of 0.1 and growth rate of 1.2 are specified. The fixed size function is attached to both the left face face.1 and right face face.2. When face.1 is meshed with the map scheme, the mesh sizes on the common edge of the two faces are decreased (i.e. smaller mesh size than computed by the fixed size function) in order to match the mesh intervals on the opposite paired edge which is also the source edge of the fixed size function. Later, when face.2 that is adjacent to face.1 is meshed with the triangle/pave scheme, the mesh size obtained from the defined size function will be very different from the existing mesh on the common edge, causing big size jump near the common edge.

Figure 5 is similar to Figure 4, but now the source entity is the upper-left vertex, therefore the mesh distributions on the common edge $ab$ are non-uniform. Suppose face.1 on the left side is first map meshed. When the right face is meshed with either the map scheme (Figure 5 (a)) or the quad/pave scheme (Figure 5 (b)), great size variation can be observed.

From above illustrations it can be seen that the face or volume meshed first will have great (and usually adverse) impact on the mesh quality of the face or volume across the common boundary that is meshed later and whose attached size functions can not match or smoothly transition the mesh sizes on the premeshed common boundary. It is difficult, if not impossible, to handle the mesh size conflict across the common boundary by using existing size functions, nor by specifying additional sources to them, because existing size functions can only measure the mesh sizes of their sources based on curvature, proximity and fixed sizes, but they can not evaluate their sources by means of existing meshes on the sources that may have arbitrary and variable mesh size distributions that are non-predictable before hand.

Suppose we can define a new size function for the case in Figure 4 and Figure 5 in such a way that it can use the common edge as source entity and be attached to the face.2. This size function is valid only when its source entity has existing mesh when being evaluated. Then this new size function will dominate its attached face and give smaller sizes than previously defined fixed size function in its affected area, so that the mesh size from the existing mesh on the source edge will be grown into the neighboring face.2 and the mesh on face.2 will be improved significantly. This fact suggests that we should create a new type of size function to address this awkward situation to ensure smooth transition across the common edge of different mesh domains. More than just growing the mesh size from boundary into the interior of mesh domain, the definition of this new size function will help to resolve the mesh size conflict that may occur across the common boundaries of adjacent mesh domains.

Sometimes, imported face or edge meshes that were generated from outside the meshing product need to be
preserved and used in the creation of a mesh for a geometry model. Figure 6 demonstrates such a case where face “A” has imported mesh that needs to be taken into account in the generation of volume mesh. It is required that the new mesh grows smoothly from the imported mesh into the rest of the domain of the geometry. None of the previously implemented size functions in our meshing product can satisfy this kind of requirement, thus the mesh size function to be presented in this paper is indispensable for this purpose.

Figure 6  Mesh creation from imported mesh

The goal of this work was to create a new type of size function, named as mesh size function, which respects the existing mesh on premeshed source entities, controls the mesh size growth from premeshed source entities to the attached entities and at the same time, like other size functions we already had, provides very rapid evaluators that would be general for any meshing algorithm.

This paper describes how this new type of size function is implemented using a background overlay grid. The work will be presented by comparing this new size function with old ones, and their differences being emphasized. Application examples of this new size function are given.

2. DEFINITIONS OF SIZE FUNCTIONS

As in our previous implementations, the new mesh size function is also based on a distance-controlled radiation. The parameters that are common to all size functions are used for the new mesh size function too. They are:

- Source entities: Edges or faces that have existing meshes are used as geometric entities. When the mesh size function is defined, the source entities may not have meshes, but they should have meshes available in order to be valid in meshing the attached entities.

- Attached entities: The attached entities on which the mesh size function will have influence include edge, face or volume. For mesh size function, usually the attached entity is different from the source.

- Growth rate: This parameter controls the geometric pace with which the mesh on the premeshed source entities is grown into affected areas.

- Size limit: This is the maximum mesh size. When the grown size at the given location exceeds the size limit, this limit is used instead.

In our previous implementation, all the size functions require a specific parameter, respectively, to define the mesh sizes on the source entities for initialization purpose. In the definition of the mesh size function, the existing meshes on the source entities are directly used as starting sizes, so only the common parameters list above will be enough for its definition.

3. SIZE FUNCTION INITIALIZATION

In preparation for the generation of the background grids, all types of size functions must be initialized differently. This initialization establishes the desired sizes everywhere on the sources. For old size functions, it is needed to generate a reasonable faceted representation of the source entities and then an ideal mesh size is computed for each piece of facet and stored in it.

For the mesh size function, however, we directly use the meshes on the source edge or source face as input. For a meshed edge source, each element of the edge mesh is converted into an edge segment and the length of the segment represents the local mesh size on that edge. An edge segment holder is used to store all the edge mesh segments associated with the premeshed geometry edge. For a meshed face source, we convert each triangle element of the face into a facet and pass the size information of that triangle element to the facet. If the source face has quadrilateral elements, each quad element is split into two triangle elements each of which is converted into a facet of the source face. The mesh size of a face facet is computed as the averaged length of the three sides of its original triangular element from which it is converted. A face facet holder is used to store all the face mesh facets associated with the premeshed geometry face.

When evaluating the mesh size at a point in the space, the point is first projected to a selected edge segment (for edge source case) or face facet (for face source case). If the projection is valid, the mesh size stored in that edge segment or face facet is taken as the start size and then grown to the given point, according to specified growth rate of the mesh size function.

4. BACKGROUND GRID GENERATION

4.1 Improved procedures of establishing mesh sizes at nodes of background grid

As a result of the size function initialization, the desired size on all sources is known. The next step is to establish the complete background grid, realized by the refining process. This procedure was described in detail in our previous work [8] and will not be listed here. The only difference is that at each corner node, the background grid will also use the mesh size radiated from the mesh size functions and compare it
with all other mesh sizes obtained from old size functions, when applied together.

However, when establishing values at the background grid nodes, an improvement can be made regarding the approach of growing the mesh size from the source entity to a given point. Previously, we used an interactive procedure to determine the spacing at a given point as influenced by a particular source. Using the prescribed geometric growth factor, 'g', we essentially "march" to the desired point from the source, applying the growth factor at each interval and summing the result. Rather than iterate in this fashion, the desired mesh size can be obtained by analytically summing the terms of the geometric series given as

\[ S_n = S_0 \cdot g^n \quad (S_0 \text{ is the spacing at the source, } n \geq 0) \]

The distance from the source entity to the given point is the sum of mesh sizes at incremental intervals except for the first mesh size on the source, and then the proper terms of the series can be listed as:

\[ R_n = R_{n-1} \cdot g \quad (n > 0) \]
\[ R_0 = 0 \quad (n = 0) \]

Or in full expression:

\[ R_0 = 0, \quad R_1 = S_0 \cdot g, \quad R_2 = S_0 \cdot g^2, \ldots, \quad R_n = S_0 \cdot g^n \]

Knowing the Euclidean distance (R) from the source to the node in question, we can sum all the items in the series until \( R_n \), so that we can then directly solve for the exponent as follows:

\[ R_n = S_0 \cdot \left( g^n - 1 \right) / \left( g - 1 \right) \cdot S_0 \]
\[ g^n = R \cdot \left( g - 1 \right) / S_0 + g \]
\[ n = \ln \left( R \cdot \left( g - 1 \right) / S_0 + g \right) / \ln \left( g \right) \]

Finally we take the integer part of the obtained n value.

\[ n = \left( \text{int} \right) n \]

which can then be used to immediately evaluate the spacing at the node without the need for iteration. The desired point will locate within the region between two subsequent distances \( R_n \) and \( R_{n+1} \) from source that are measured at incremental steps \( n \) and \( n+1 \), respectively. Then the following condition can be satisfied:

\[ R_n \leq R \leq R_{n+1} \]

This would simplify the evaluation of \( S_n \) and speed up the calculation.

A linear interpolation between the two bounding distances is accomplished by this equation

\[ \gamma = (R - R_n) / (R_{n+1} - R_n) \]

Here \( 0 \leq \gamma \leq 1 \). The actual size, \( S_P \), at the given point, \( P \), is computed as:

\[ S_P = (1 - \gamma) \cdot S_n + \gamma \cdot S_{n+1} \]

The final size is the smallest one of the defined size limit and the all computed sizes (if a corner point is affected by several size functions).

### 4.2 Improvement to projections to source entities under way

According to our statistics obtained from timing the profile of size function creation, it is found that the bottleneck of the size function speed is the projection of the corner nodes of background grid to the faceted source entities, which counts for about 90% of the total time. The remaining time is spent for other operations such as growing the initial mesh size on source entities to a given point along the distance, inserting newly computed mesh size into a sorted list, and getting mesh size at a shared point from the list. The most time used for projection is spent in searching the best facet to project the node. An investigation in improving the projection process is being under way which tries to project a list of nodes in one background grid to the best facets at the same time. This approach will significantly reduce the time in background grid generation once successfully realized.

![Flow chart of the size function applications](image-url)
4.3 Flow chart of the background grid size function approach

No matter what types of the size functions to use, except for the differences in initializations, the same procedures will be followed when apply these size functions to the meshing processes. The following chart in Figure 7 illustrates the general procedures we used in the meshing processes for all meshing schemes. After defined, the size functions are attached to the geometric entities via a specially designed data structure. Initialization of size functions is triggered if any of the attached entities or their lower topologies is being meshed. The background grid generation for the attached entities follows the initialization process, creating a specific set of background grid for each group of entities that have identical size functions attached. The established background grid serves as an evaluator providing mesh size information to the meshing process. After entering the meshing session, the mesh size at a given point is evaluated quickly through tri-linear interpolations in a background cell into which the point falls. By the returned mesh size value, the next mesh node is placed along certain direction.

5. EXAMPLES

A few examples are given below to show the application of the new mesh size function or its combination with other types of size functions in the meshing process. Smooth meshes that were unable to be generated before have been generated due to the introduction of the new mesh size function.

5.1 Updated meshing results for co-centric volumes

Figure 8 is the updated meshing results of the example in the beginning of this paper (see Figure 2). To prevent the size jump in the interior volume.2, a mesh size function is created that uses the common face as source and is attached to the interior volume. In order for the mesh size function to be useful to its attachment, the exterior volume.1 should be meshed first in this case, so that the common face inherits its meshes from the meshing process of exterior volume before interior volume.2 is meshed, thus the mesh size function using the common face as source can be valid for use in meshing volume.2.

Figure 9 gives the new meshing results corresponding to Figure 3. Similarly to Figure 8, the meshes of the interior volume are radiated nicely from the common face, although the meshes on common face have varying degrees of sizes than in the previous case.

5.2 Remeshed results for connected faces

Next, referring to Figure 4, we have defined a second mesh size function that uses the common edge of the two connected faces as source and attach it to the face on the right side (see Figure 10). Since the mesh size function has smaller size distributions everywhere in the right face than the original fixed size function and so will dominate the mesh size selection in the whole domain of the right face, the tri/pave algorithm will use the mesh size from the mesh size function to position nodes, forming smooth mesh transitions from the common edge and across the whole face on face.2. The adjustment to the mesh distributions on the common edge does not deteriorate the mesh quality on the right face any more.

Figure 8  Remeshed results from Figure 2 when mesh size function is applied

Figure 9  Remeshed results from Figure 3 when mesh size function is applied

Similar results to the above have been obtained in Figure 11 for the case displayed in Figure 5 where the initial size function start from the upper-left vertex, instead of the left-most edge as in Figure 4. No matter the right face is meshed with the map scheme (Figure 11 (a)) or the quad/pave scheme (Figure 11 (b), the meshes on the face are grown in such a way that you will not notice any sudden changes of mesh sizes near the common edge and it looks like the whole meshes are smoothly radiated from the same upper-left vertex without size jumping.
5.3 New meshing results for volumes with imported face mesh

We discussed the impossibility of generating a volume mesh that is required to radiate from the imported face mesh and concluded that there was no easy way of doing it with the old size function capabilities in our mesh sizing tool. However, with the realization of the mesh size function, this task becomes very easy. Simply specify the face “A” having imported mesh as the source face (see Figure 6) and specify the volume to be meshed as attachment entity of the mesh size function, and then start meshing the volume. The volume, which could be imported together with the meshed face or created from the imported face (e.g. by sweeping the given face along specified path) within Gambit product, will be meshed according to the user’s specification. Figure 12(a) shows the mesh on the boundary surface of the volume after the meshing process is finished, and Figure 12(b) is the internal mesh patterns of the volume. A growth rate of 1.2 is used in the definition and the size limit is large enough not to be reached within the domain of the model.

![Figure 10](image1.png)  
**Figure 10** Mesh distributions on right face when the common edge is used for mesh size function

(a) Mapped mesh on right face

(b) Quad/paved mesh of right face

![Figure 11](image2.png)  
**Figure 11** Mesh distributions on right face using a mesh size function from the common edge

![Figure 12](image3.png)  
**Figure 12** Meshing results of volume using imported face mesh as source

CONCLUSION

From the established method of size functions using the background overlay grids, a new mesh size function has been set up for controlling mesh sizes and radiation from premeshed geometric entities (i.e. edges and/or faces). The defined mesh size function has provided supplemental means to assist all the meshing tools where the size functions that were implemented previously sometimes could not meet the
special needs. The details on how to construct the new mesh size function has been described and its comparison with other size functions presented. The proposed mesh size function has been implemented in Gambit product, and its efficiency has been illustrated by successful meshing examples with satisfactory results.

REFERENCES


AUTOMATED ADAPTIVE FORMING SIMULATIONS
Jie Wan, Suleyman Kocak and Mark S. Shephard
Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY, U.S.A.
shephard@scorec.rpi.edu

ABSTRACT
In this study, an automated adaptive mesh control scheme, based on local mesh modifications, is developed for the finite element simulations of 3D metal forming processes. Error indicators are used to control the mesh discretization errors, and a consecutive $h$-adaptive procedure is conducted. The mesh size field used in the $h$-adaptive procedure is processed to control the geometric approximation errors on the evolving workpiece boundary mesh. During the simulations the workpiece mesh is adaptively enriched whenever the mesh is no longer acceptable. Industrial problems are investigated to demonstrate the capabilities of the developed scheme.

Keywords: adaptive mesh control, metal forming, mesh adaptation

1. INTRODUCTION
The manufacturing and testing process of aerospace components is expensive and difficult. Mathematical modeling tools have been developed to reduce/replace the historic trial and error process. As a mathematical tool, the finite element method has been widely used to simulate metal forming problems like forging, extrusion, rolling, etc., where the mesh of the workpiece evolves to represent the material flow. These processes often require the mesh of the deformed workpiece to be enriched whenever the mesh becomes unacceptable due to severe distortion or workpiece-die interference occurring during the incremental flow process. In these cases, it is necessary to replace the deformed mesh with an updated mesh, which is better-conditioned and consistent with the current configuration [1-5]. History dependent state variables also need to be accurately transferred from the old mesh to the new mesh [1,3].

Most procedures developed to update the deformed mesh of the workpiece apply automatic mesh generation techniques to generate a completely new mesh [1-9]. Typically, a remeshing process includes the following steps: 1) Update the boundary mesh representation; 2) Generate a new mesh; and 3) Transfer the history-dependent field variables from the old mesh to the new mesh. In remeshing process, the entire domain of the workpiece is remeshed, even though there might be only a limited number of elements that need to be modified. The workload in this process could be decreased significantly by applying local modification operators.

The emphasis of the developed procedures is the effective enrichment of the deformed mesh. The mesh enrichment process is carried out by applying local mesh modification operators. An adaptive procedure built on discretization error indicators is used for the calculation of the mesh size field. In addition the mesh size field is modified to help control geometric approximation errors in the next set of simulation steps. The simulation process is automated by combining the adaptive mesh control procedure with the forming analysis engine to enable continuous automated simulation.

This paper is organized as follows: An overview of adaptive mesh modification for evolving geometries is given in Section 2. The model topology update process is discussed in Section 3. Section 4 is dedicated to definition of the mesh size field. The controlled mesh modification process is explained in Section 5. Example problems from industrial applications are given in Section 6, and finally some conclusions are drawn in Section 7.
2. OVERVIEW OF ADAPTIVE MESH MODIFICATION FOR EVOLVING GEOMETRY PROBLEMS

In metal forming simulations, the workpiece undergoes large plastic deformation that results in major changes in the geometry of the model and the mesh. Mesh enrichments are usually needed due to two reasons (i) element shapes degrade or (ii) discretization errors become too large. In the updated mesh, it is the quality of elements that determines how many time steps can be taken before the next mesh enrichment is needed. For this reason, special care is given to the mesh enrichment process.

DEFORM™ [10] is the FEM engine used in this effort. DEFORM is tailored for large deformation modeling, and it can analyze various forming and heat transfer processes. The formulation of the engine is given in Kobayashi et al. [11]. The engine supports rigid, elastic and thermo-viscoplastic material models, and it has an extensive material database for many alloys.

Mesh enrichment process used in this study includes the following steps:

Update the model topology: During the simulation steps the geometry and topology (in terms of contact surfaces) of the workpiece model evolve. Therefore, a consistent model topology must be constructed before mesh modifications. In this study, the model is defined as a non-manifold topological model that reflects the contact configuration as it evolves during the simulation.

Determine the new mesh size field: Mesh modifications are conducted based on the mesh size field. In this study, the mesh size field is determined by an h-adaptive procedure guided by error indicators and geometric approximation control.

Apply mesh modification operations: Once the mesh size field is obtained, the mesh is modified by using local mesh modification operations. History dependent solution fields are also transferred incrementally during these operations.

3. MODEL TOPOLOGY UPDATE

In a forming simulation, the geometrical components consist of the workpiece geometry, die geometries and die motions. To properly perform an automated forming simulation, a topological description of the evolving workpiece boundary must be available to reflect the simulation status so that the workpiece boundary and the analysis attributes are appropriately maintained during the mesh updating.

Since a FEM engine will only track nodal contact information, a procedure is needed to update the topological representation of the workpiece boundary before each mesh enrichment step. The mesh model to be constructed needs to provide a consistent geometric interpretation of the contact boundary conditions between the workpiece mesh and the die surfaces, which are extracted from the solution information in terms of nodal contact. The radial edge topology structure [12,13] given in Figure 1 is used to express the topological model of the workpiece domain.

![Figure 1. The radial edge topology structure for model](image)

Given the mesh topology [14] and nodal contact conditions, mesh model construction steps can be summarized as follows with the aid of Figures 2-4, where ○ represents a mesh vertex in contact and □ represents a mesh vertex not in contact, and \( M_i^d[G_e^c] \) indicates a boundary mesh entity classified in contact and \( M_i^s[G_e^s] \) indicates a boundary mesh entity classified not in contact:

1. All boundary mesh vertices are marked according to the nodal contact conditions. The markings reflect if the mesh vertices are in contact with the specific die surfaces;
2. For each boundary mesh face, mark it in contact if all its bounding mesh vertices are in contact with the same die surface, or not in contact otherwise. The markings of boundary mesh faces are illustrated in Figure 2;
3. Consider next the boundary mesh edges. If such a mesh edge bounds only one contact mesh face, or does not bound any contact mesh faces but its two bounding mesh vertices are in contact with the same die surface, classify it on a model edge \( G_i^q \); otherwise classify it on a model face \( G_i^c \). Classifications of boundary mesh edges are illustrated in Figure 3, where the thickest lines indicate the mesh edges classified on model edges;
4. Consider each boundary mesh vertex. If it is marked in contact and does not bound any mesh edges classified in contact with a die surface, classify it on a model vertex \( G_i^0 \); otherwise count the number of its adjacent model edge classified mesh edges. In the latter case, if the number is 1 or greater than 2, classify it on a model vertex \( G_i^p \); or if the number equals to 2, classify it on a model edge \( G_i^1 \). In other cases, classify it on a model face \( G_i^3 \). Classifications of boundary mesh vertices are illustrated in Figure 4;
5. Model edges are defined by the appropriate collecting of mesh edges classified on model edges that connect together;

6. Model faces are defined by collecting of mesh faces classified in contact with specific die surfaces or on the free surfaces that are bounded by loops of model edges.

When the essential boundary conditions are applied, additional care is needed so that the model topology defined also reflects those boundary conditions.

The above process naturally results in a mesh model, in which all mesh entities \( M_{i}^{d} | G_{j}^{d} \) classified on a specific model entity \( G_{j}^{d} \), where \( d_{i} \leq d_{j} \leq 2 \), have the identical boundary conditions. This enables the automatic maintenance and delivery of contact (and other BC’s if defined) for the workpiece mesh during the local mesh modifications applied on the workpiece boundary. A mesh model constructed is shown in Figure 5, where a single shell consisting of 3 contact model faces and 1 free model face is defined (the third contact model face is at the bottom). All mesh entities, \( M_{i}^{d} \) where \( 0 \leq d \leq 2 \), classified on any contact model face and its closure, are in contact with the same die surface.

Figure 5: A mesh model for deformed object

4. MESH SIZE FIELD SPECIFICATION

Mesh discretization errors arise from the use of a finite dimensional solution space defined in terms of polynomial basis functions over a set of mesh entities. When the order of the polynomial basis function approximation is fixed, these errors directly depend on the mesh size and vanish as the mesh size approaches zero. Accordingly, these errors can be controlled by employing an \( h \)-adaptive analysis process when the mesh size is controlled through the domain. In this study, a mesh size field is built using error indicators and processed to control geometric approximation errors.

4.1 Error Indicators

A recovery-based \textit{a posteriori} procedure developed by Zienkiewicz and Zhu (ZZ) [15] is used. The recovered solution is obtained based on the projection of effective strain field and is adopted to replace the exact solution in obtaining the element error indicators. The error \( \| e_{\tau} \|_{e} \) for an element \( \tau \) is given in \( L_{2} \) error norm by

\[
\| e_{\tau} \|_{e} = \left( \int_{\Omega_{\tau}} (e_{\tau} - \epsilon_{h})^2 \, d\Omega \right)^{\frac{1}{2}}
\]

(1)

where \( e \) and \( \epsilon_{h} \) are respectively the exact value and the finite element approximation of effective strain. Since \( e \) is unknown, an “improved” solution \( \epsilon^{*} \) is obtained by the recovery procedure and used as an indicator of the elemental error \( \| e_{\tau} \|_{e} : \)

\[
\| e_{\tau} \|_{e} = \| e^{*} \|_{e} = \left( \int_{\Omega} (\epsilon^{*} - \epsilon_{h})^2 \, d\Omega \right)^{\frac{1}{2}}
\]

(2)

For a given domain \( \Omega^{*} \), the error norm is estimated as:

\[
\| e_{\tau} \|_{e} = \left( \sum_{\tau \in i} \| e_{\tau} \|_{e} \right)^{\frac{1}{2}}
\]

(3)

where \( n \) is the total number of the elements within the domain \( \Omega^{*} \). Furthermore, the error can be interpreted by taking the relative percentage error \( \eta \) defined as:
where \( \| \varepsilon \| \) is the corresponding norm of the recovered solution over the domain \( \Omega' \).

### 4.2 An H-Adaptive Procedure

Based on the element error indicators and convergence rate of the finite element method, the mesh adaptivity can be readily performed to satisfy certain mesh optimality criteria. A general requirement is to ensure a minimum percentage error in certain (energy or \( L_2 \)) norm to be achieved so that

\[
\eta \leq \hat{\eta}
\]

is met, where \( \hat{\eta} \) is the maximum allowed relative percentage error.

The optimal mesh will have the same contribution from the estimated element errors based on the following equation [16]:

\[
h_{t}^{\text{new}} = h_{t}^{\text{old}} \times r_{t}
\]

where \( h_{t}^{\text{old}} \) and \( h_{t}^{\text{new}} \) denote the current characteristic size of an element \( t \) and the expected characteristic size of the new elements inside the subdomain covered by element \( t \), respectively. The element size factor \( r_{t} \) is computed from the estimated element errors based on the following equation:

\[
r_{t} = \left\| \varepsilon \right\|^{2} \left( \frac{\hat{\eta}^{2}}{\sum_{i=1}^{d} \left\| \varepsilon \right\|^{2d}} \right)^{rac{1}{2d}}
\]

where \( d \) and \( p \) are respectively the dimension and order of the elements. In this study, \( d = 3 \) and \( p = 1 \).

Figure 6 shows an input mesh and an adapted mesh. As can be seen from the figure, a finer mesh is obtained where the strain gradients are high.

### 4.3 Processing the Mesh Size Field to Control Geometric Approximations

The mesh size field is adjusted to satisfy the further requirements over the workpiece boundary mesh to reduce geometric interference (penetration) between the workpiece mesh and the die surfaces, and to control mesh gradation around small model edges and faces that are subject to substantial deformation in future steps.

**Geometric interference (GI)**

The workpiece boundary can become overlapped with the die geometries during the forming analysis [6,8,10,17]. It has been observed that an important source of geometric approximation errors due to workpiece-die overlap leads to inaccurate representation of the die-workpiece interaction and therefore results in underestimation of required forming load and overestimation of required filling material [10]. Various methods have been devised to control the overlap in the remeshing-based procedures [8,10,17]. The die-workpiece interference is generally adopted to measure the physical overlap between the workpiece and die geometries. It can be defined as \( \delta = Q_u \cap Q_d \), where \( Q_u \) and \( Q_d \) represent the domains of the workpiece and dies, respectively. In order to improve the simulation results, it is desired to control the geometric interference in the updated mesh.

One of the widely used measures for quantifying the amount of \( \delta \) is interference depth [8,10,17]. Generally, interference depth is defined as the minimum translational distance required to separate two overlapping objects. In forming simulations, interference depth is usually assessed as the distance from the middle point of a mesh edge with a contact node at each end to the die surface in the inner normal direction \( \varepsilon_{\text{inner}} \) of the workpiece boundary at the middle point [10,17] as illustrated in Figure 7a, where \( \bigcirc \) represents a mesh vertex in contact. Denoting the allowed interference amount as \( \delta_{p} \), if \( \delta \geq \delta_{p} \) at any portion, it is regarded that the die-workpiece interference has become exceeding (too large) and needs to be reduced.

GI calculation based on examination of single location would clearly be unsuitable in the situations illustrated in Figures 7b, c and d. The GI would be computed as zero in Figure 7b and much smaller than the actual amount in Figure 7c. For the case shown in Figure 7d where \( \Box \) represents a mesh vertex not in contact, which often occurs in the incoming region, since the mesh edge \( M_{j}^{\text{new}} \) is bounded by one free mesh vertex \( M_{j}^{0} \) and one contact mesh vertex \( M_{j}^{0} \), it is not considered at all. As a remedy,
in the developed procedure, interference depth is computed as the maximum value among the distances measured from multiple locations (points) along a boundary mesh edge \( M_i^j \) of the workpiece to a die surface in the inner normal direction \( \hat{n}_{\text{inner}} \) of the workpiece boundary, where \( 1 \leq d \leq 2 \) and \( M_i^j \) is bounded by 1 or 2 contact mesh vertices. This approach is illustrated in Figure 8, where 3 measuring points are taken and the computed GI is likely to be closer to the actual GI than that computed in Figure 7b and 7c. To calculate the workpiece-die distance at multiple points efficiently, a searching structure based on an octree [18] is adopted.

The bounding box of the mesh face is defined by two corners located at \( X_{\text{min}} = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) \) and \( X_{\text{max}} = (x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) \). For the mesh face to be contained in a terminal octant, its bounding box must reside at least partially inside the terminal octant, namely

\[
\begin{align*}
& x_{\text{min}} \leq x_0 \leq x_{\text{max}} \quad \text{and} \quad \gamma_{\text{min}} \leq y_0 \leq y_{\text{max}} \\
& z_{\text{min}} \leq z_0 \leq z_{\text{max}}
\end{align*}
\]

where, \( X_{o_{-\text{min}}} = (x_{o_{-\text{min}}}, y_{o_{-\text{min}}}, z_{o_{-\text{min}}}) \) and \( X_{o_{-\text{max}}} = (x_{o_{-\text{max}}}, y_{o_{-\text{max}}}, z_{o_{-\text{max}}}) \) represent the lower and upper corner points of the octant respectively, and \( tol \geq 0 \) is a tolerance defined according to the allowed GI, \( \delta_p \), in the simulation. In this study, we take \( tol = 1.1 \times \delta_p \) so that only the mesh faces contained in a specific terminal octant are considered to compute the workpiece-die distance at any point \( Q(x, y, z) \) that satisfies:

\[
\begin{align*}
& x_{o_{-\text{min}}} \leq x \leq x_{o_{-\text{max}}} \quad \text{and} \quad y_{o_{-\text{min}}} \leq y \leq y_{o_{-\text{max}}} \\
& z_{o_{-\text{min}}} \leq z \leq z_{o_{-\text{max}}}
\end{align*}
\]

Mesh edges on the contact surfaces exceeding the allowed GI are to be refined by altering the local mesh size field to capture the local geometric features on the die surfaces. Figure 9 shows an example application of GI control.

Mesh gradation control

When small but required model features are defined by the topology update procedure, the mesh size field around those small model features is adjusted to avoid introducing a number of poorly-shaped elements due to large mesh gradation. In the local mesh illustrated in Figure 10, there is a short model edge \( G^i_1 \) bounded by two model vertices \( G^i_0 \) and \( G^i_1 \). Denoting the length of \( G^i_1 \) as \( l_i \) and the requested mesh size around \( G^i_1 \) as \( s_i \). When \( s_i \gg l_i \), a poorly-shaped element \( M^j_0 \) is to be introduced as shown in Figure 10a. Therefore it is desired to adjust \( s_i \) close to \( l_i \) so that a smooth mesh gradation is resulted as shown in Figure 10b.
5. CONTROLLED MESH MODIFICATIONS

Adaptive mesh control is executed through the applications of local mesh modification operations.

5.1 Local Mesh Modification Operations

Before discussing the enrichment processes, the local mesh modification operations are introduced, which include:

- Splitting of mesh edges, faces and regions
- Collapsing of mesh edges, faces or regions
- Swapping of mesh edges and faces
- Geometry modification of mesh vertices, edges or faces

Splitting of mesh edges, faces and regions

The most commonly applied splitting procedure is edge splitting. In the case of simplex elements, it is straightforward to define a set of templates to account for the edge splitting in the higher order mesh entities the edge bounds [19].

Collapsing of mesh edges, faces or regions

There are a number of specific mesh entity collapsing operations that can be defined for the case of simplex elements. The most commonly applied collapsing operation is an edge collapse. The application of a requested collapsing operation is precluded if it would create a mesh invalidity [19]. Additionally, some application related geometric constraints may be applied to control the use of these operators. This will be discussed later.

Swapping of mesh edges and faces

Efforts to date on the development of mesh entity swapping procedures have been focused on simplex elements. In 3-D the swap operations are qualified and a variety of edge and face swapping operations have been defined. The application of a requested swapping operation is precluded if it would create a mesh invalidity [19].

Geometry modification of mesh vertices, edges or faces

The positions of mesh vertices and the shapes of mesh edges and faces can be altered. In all cases, there is a limit to the amount of geometric change possible before one or more connected higher order mesh entities become invalid.

5.2 Mesh Enrichment Process

Given a desired mesh size field and an expected element shape criteria for the elements, the mesh enrichments are executed by the controlled applications of local mesh modifications. Whenever either element size or shape is not satisfied, a mesh enrichment is determined and performed to eliminate the deficiency and achieve the best satisfaction of the requested size and shape.

5.2.1 Mesh entity refinement and coarsening

When an element does not satisfy the local mesh size, mesh entity refinement or coarsening, considering control of the resulting mesh entity shapes, is executed.

Mesh entity refinement is accomplished using splitting operations with edge splits being most popular due to their flexibility. It is always possible to introduce the desired mesh entity refinement in those cases where introduced mesh entities do not need to have their geometry modified. In the case where mesh entities are classified on a curved boundary of the model are split, it is necessary to modify the geometry of those mesh entities so they lie on the model boundary. These geometric modifications can invalidate the shapes of connected higher order mesh entities. In those cases additional mesh modification operators are applied to yield a valid mesh [20].

Many of the published mesh coarsening procedures are restricted to the reversal of previously executed refinement procedures. For the purposes of a more general mesh modification processes, a coarsening procedure based on mesh entity collapsing is more flexible. Edge-based collapsing procedures are well suited to most collapse needs. However, recent efforts indicate that other collapse operations are advantageous in specific circumstances. Since a specific collapsing operation may not be allowed due to its producing a mesh invalidity or large geometric approximation error, it is not always possible to execute a specific coarsening.

5.2.2 Element shape improvement

Element distortions in the workpiece mesh may cause degradation in the computed results or discontinue the simulation. Therefore, the distorted elements need to be detected and improved efficiently. The modified mean ratio, which is defined as \( \left( \frac{1}{V} \sum_{i} l_i \right) \), where \( V \) is the volume of the tetrahedron and \( l_i \) is the length of \( i \)-th mesh edge bounding the tetrahedron [21], can be used to evaluate the distortion level. The distortion factor is 1.0 for an equilateral element and a value of 0.0 indicates a zero volume element. Depending upon the acceptable distortions in the FE mesh, local modifications are made to improve the shape of the distorted elements.
Mesh shape improvement efforts include node-point repositioning procedures and the application of local mesh modifications. Emphasis here is on the controlled mesh modifications.

For a poorly-shaped tetrahedral mesh region, $M_i$, rank its six bounding mesh edges, $M_j$, where $1 \leq j \leq 6$, by the order of their lengths squared, $l_j^2$, as follows:

$$l_1^2 \leq l_2^2 \leq l_3^2 \leq l_4^2 \leq l_5^2 \leq l_6^2$$  (11)

When $l_j^2 << l_i^2$, the poor shape of $M_i$ is due to the existence of “short” mesh edge(s), otherwise it must contain large dihedral angles. Figure 11 shows examples of poorly shaped elements taking the various configurations. Figures 11a and 11b have the case of a short mesh edge, $M^j$. Figure 11c is the case of two large dihedral angles and Figure 11d is the case of three large dihedral angles. Accordingly, application of mesh modifications is guided by the two basic goals of:

- Removal of poorly-shaped mesh regions which contain “short” mesh edge(s) through elimination of those “short” mesh edges;
- Removal of the large dihedral angles based on operators keyed by the ordered mesh edge lengths and the adjacency information.

Handling mesh regions containing “short” mesh edge(s)

The options to eliminate a “short” mesh edge include collapsing it by removing either its bounding mesh vertex, an appropriate swap operation, or repositioning its bounding mesh vertices. The topological restrictions on edge collapsing are discussed in detail by Cougny and Shephard [19]. The geometric restrictions on edge collapse require that all updated mesh regions have positive volumes.

When elimination of the “short” mesh edge(s) is not allowed geometrically, alternative consideration is to remove either its bounding mesh vertex by collapsing a mesh edge that connects to it within the local mesh such that the edge is eliminated by merging with a longer edge. An example to overcome the geometric restrictions in 2D is illustrated in Figure 12, where a “short” mesh edge $M^i$ bounds two poorly-shaped elements (mesh faces), $M^j$ and $M^k$ (Figure 12a). The first attempt is naturally to collapse $M^i$ with the mesh vertex $M^0$ deleted as shown in Figure 12b. However this leads to definition of an invalid mesh face $M^3$, which are bounded by mesh vertices $M^2$, $M^4$ and $M^6$, and therefore is not allowed geometrically. As an alternative, one of the mesh edges connected to $M^i$ at $M^0$ can be collapsed with $M^0$ deleted. In Figure 12c, mesh edge $M^2$ is collapsed and in Figure 12d, mesh edge $M^3$ is collapsed. Both these operations are allowed geometrically. The modification by collapsing $M^1$ (Figure 12d) leads to the better local mesh quality and thus is selected to be applied.
Handling mesh regions containing large dihedral angle(s)

A poorly-shaped mesh region containing no “short” mesh edges must contain one to three large dihedral angles. Considering the possible effects on the local mesh, a simple deletion is preferred. Some of the cases when these elements arise happen when a flat mesh region has one or more mesh faces classified on a model face that is used by only one model region with the remaining mesh faces classified in that model region. In these cases, the mesh region can simply be deleted with the remaining mesh faces and model region classified mesh edges and vertices reclassified on the model face on which the deleted face(s) were classified.

When a mesh region containing large dihedral angle(s) can not be removed by a simple deletion, the situation needs to be analyzed based on the adjacency information to determine the most desired mesh modification to eliminate as many large dihedral angles as possible. Consider the case shown in Figure 11d, where mesh edges, $M^1_{k_{1}}$, $M^1_{k_{2}}$, and $M^1_{k_{3}}$, bounded by mesh vertex $M^0_{k_{4}}$, are the three shortest mesh edges within element $M^1_{h}$.

It is possible to eliminate all potential large dihedral angle(s) by collapsing one of these three mesh edges as shown in Figure 13a. If none of these mesh edges, $M^1_{k_{1}}$, $M^1_{k_{2}}$, and $M^1_{k_{3}}$, can be collapsed, an additional effort is to look at the shapes of the mesh faces bounded by them within the element, and to swap the longest mesh edge bounding a sliver mesh face if there exists one. The situation is illustrated in Figure 13b, where mesh face $M^2_{k_{4}}$ bounded by mesh edges $M^1_{k_{1}}$, $M^1_{k_{2}}$, and $M^1_{k_{3}}$, is sliver, and the mesh edge $M^1_{k_{3}}$ is considered to be appropriately swapped.

Treatment on new contact mesh vertices

A situation that requires specific consideration is when boundary mesh edges are split. In these cases, the new contact mesh vertices geometrically not on specific die surfaces and other new mesh vertices inside die surfaces need to be placed on the die surfaces as illustrated in Figure 14. Repositioning of the new mesh vertices is conducted through a vertex snapping procedure [20].

Free surface smoothing

It has been observed that in cases where the mesh used is not a good approximation to the smooth geometry of the workpiece free surfaces, the simulation accuracy can degrade [3]. Subdivision surface procedures can be adopted during refinement of the free surfaces to approximate the actual surface curvature. In the developed procedure, the position of the subdivision points along the boundary mesh edges are calculated in the form of interpolation functions through employing a modified butterfly scheme [22,23]. The interpolation templates to be applied are selected based on the mesh entity classification and the adjacency around the mesh edges in the local boundary mesh.

5.4 Local Transfer of History Dependent Solution Information

Operations like remeshing or local mesh modifications require the history-dependent solution fields be accurately transferred from the original mesh to the modified mesh so that the computation can be resumed and continued from the updated reference state. When the deformed mesh is updated through a local mesh modification operation, the solution fields for only the involved local mesh need to be transferred.

There are two methods used to transfer solution parameters. They involve using a) the existing interpolants or b) new interpolants defined to satisfy certain desired properties. Since only the solution information over the modified local mesh is transferred depending on individual local mesh modifications, the required calculations are local and efficient.
6. INDUSTRIAL APPLICATIONS

A back extrusion problem and a steering link problem are investigated to demonstrate the capabilities developed in this study. To show the improvements achieved, both of the problems are solved by using DEFORM only and the automated mesh modification procedure (which uses DEFORM as FEM engine) and the solutions are compared. Results obtained are presented and discussed below.

6.1 A Back Extrusion Problem

A back extrusion problem shown in Figure 16 is considered. The plastic behavior of the material is specified with a material flow stress function. A total stroke of 7.2 inches is defined, which corresponds to 180 steps with a stroke of 0.04 inch per step. The allowed geometric interference is 0.07 inch. The initial mesh of workpiece consists of 906 mesh vertices and 5433 mesh regions.

The problem is solved by 1) using DEFORM’s remeshing procedure and 2) using the automated mesh modification procedure. The simulation requires 13 DEFORM’s remeshing steps and 9 mesh enrichment steps, respectively (see Table 1).

![Figure 16. A Back Extrusion Problem](image)

Table 1: Remeshings and Mesh Enrichments used

<table>
<thead>
<tr>
<th>Remeshings</th>
<th>At Step</th>
<th>Number of Elements</th>
<th>Mesh Enrichments</th>
<th>At Step</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>10049</td>
<td>1</td>
<td>5</td>
<td>7041</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>10834</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>10763</td>
<td>2</td>
<td>19</td>
<td>10575</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>14171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>21158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>52</td>
<td>21605</td>
<td>3</td>
<td>50</td>
<td>10890</td>
</tr>
<tr>
<td>7</td>
<td>65</td>
<td>21754</td>
<td>4</td>
<td>72</td>
<td>13077</td>
</tr>
<tr>
<td>8</td>
<td>81</td>
<td>21727</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>92</td>
<td>23242</td>
<td>5</td>
<td>99</td>
<td>14246</td>
</tr>
<tr>
<td>10</td>
<td>112</td>
<td>24698</td>
<td>6</td>
<td>120</td>
<td>17952</td>
</tr>
<tr>
<td>11</td>
<td>134</td>
<td>24599</td>
<td>7</td>
<td>133</td>
<td>25219</td>
</tr>
<tr>
<td>12</td>
<td>154</td>
<td>25272</td>
<td>8</td>
<td>155</td>
<td>30325</td>
</tr>
<tr>
<td>13</td>
<td>166</td>
<td>24771</td>
<td>9</td>
<td>172</td>
<td>38136</td>
</tr>
</tbody>
</table>

In Figure 17 the maximum dihedral angles of the workpiece mesh are compared before and after the remeshings and the mesh enrichments, respectively. It can be seen the mesh quality is stably and largely improved through the mesh enrichments, particularly during the later stages of the simulation.

![Fig 17. Changes of the maximum dihedral angles before and after remeshings/mesh enrichments](image)

A temperature field updated by the local mesh modification dependent solution transfer operators is shown in Figure 18. As can be seen, solution transfer does not lose any significant information. As expected, the global transfer demonstrated a greater diffusion of peak values.

Four steps of the simulation process are given in Figure 19. From the figure, it can be seen that the contact surface between the moving die and the workpiece is well captured by controlling the workpiece-die geometric interference. Furthermore, it can be observed that high mesh density is obtained around high strain gradient regions, which are in the vicinity of moving die surface.

![Figure 18. Updated temperature field by local transfer operators](image)
6.2 A Steering Link Problem

A steering link problem shown in Figure 20 is considered. The plastic behavior of the material is specified with a material flow stress function. A total stroke of 41.7mm is defined, which corresponds to 278 steps with a stroke of 0.15mm per step. The allowed geometric interference is 1.0mm. The initial workpiece mesh consists of 6765 mesh vertices and 28885 mesh regions.

The problem is solved respectively by 1) DEFORM’s remeshing procedure and 2) the mesh modification procedure. The first remeshing/mesh enrichment is needed at STEP 112. It takes 35 DEFORM’s remeshings and 18 mesh modification steps.

**Improvements of element quality**

The quality of the workpiece mesh is improved stably by the mesh enrichments through the entire simulation process. Figure 21 shows the improvements to the maximum dihedral angles before and after the mesh enrichments.

**Mesh size control**

The workpiece mesh is adapted to control solution error and the geometric approximation. The maximum edge length increases from 11.98mm in the initial mesh to 24.49mm in the final mesh while the minimum edge length decreases from 0.92mm to 0.30mm. The effects of mesh adaptivity are shown in Figure 22.

The interior mesh density and effective strain of the final workpiece mesh are compared between the adaptive enrichment based and the remeshing based simulations as shown in Figure 23.

The simulation ends with a final mesh containing 24729 mesh vertices and 115483 mesh regions. The estimated relative error in effective strain decreases from 11.69% at STEP 112 to 6.53% at the end. The final mesh is compared with the one obtained by DEFORM’s remeshings in Figure 24.

7. CLOSING REMARKS

This paper has considered an automated adaptive mesh control procedure for metal forming simulations. The adaptive mesh control is executed through the controlled applications of mesh modification operators, which are based on an adaptively defined mesh size field. To properly maintain the workpiece boundary conditions, the model topology of the workpiece mesh is updated consistently with the solution information. The mesh size field is obtained by accounting for discretization errors through an $h$-adaptive procedure guided by error indicators and procedures to control the geometric approximations.

The adaptive mesh modifications are applied to two 3-D forming problems and compared with a procedure that employs complete remeshings. The results demonstrate the ability of adaptive mesh modification procedures to
properly deal with any of the complications that can arise during the simulation. Mesh modification based adaptive procedures have also been applied to transient flow simulations including ones with bodies in relative motion.

Figure 22. Mesh adapted consistently with the effective strain profile

Step 120

Step 160

Step 180

Step 200

Step 220

Step 240

Step 260

Completion

Figure 23. The final interior mesh density and effective strain profiles

(a) By mesh enrichments

(b) By remeshings

Figure 24. Final meshes of the workpiece

(a) By mesh enrichments

(b) By remeshings

(12046 mesh vertices and 51987 mesh regions)

REFERENCES


[10] DEFORM 3D™ User Manual, SFTC, Columbus, OH.


ANISOTROPIC MESH ADAPTATION FOR TRANSIENT FLOWS SIMULATIONS

Pascal J. FREY and Frédéric ALAUZET

† INRIA, Projet Gamma, Domaine de Voluceau, BP 105, 78153 Le Chesnay cedex, France
‡ Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, 75013 Paris, France

ABSTRACT

Unstructured mesh adaptation has already revealed very efficient for computing an accurate solution in a reasonable amount of time on current PC architectures. Two features are still missing in the adaptation scheme: (i) the creation of arbitrary anisotropic meshes and (ii) the capture of transient phenomena. Therefore, in this paper, we propose a global scheme suitable to compute steady-state as well as transient problems, based on anisotropic mesh adaptation. Several examples of numerical simulations in CFD are provided to emphasize the efficiency of the proposed approach.

Keywords: Mesh adaptation, anisotropy, metric, error estimate, CFD, transient problem

INTRODUCTION

Nowadays, in the context of numerical simulations, unstructured mesh adaptation is unanimously recognized as an efficient and clever method for improving the accuracy of the solution as well as for capturing the behavior of physical phenomena, even if it is not known a priori. Moreover, reducing the number of nodes (the number of degrees of freedom) allows to substantially reduce the CPU time. Hence, three-dimensional complex simulations are now commonly used in many engineering fields! It becomes even possible to compute large unsteady simulations on current workstations (no parallel architecture required) almost on a daily basis. A missing piece on the simulation checkboard remained the anisotropic adaptation for unsteady phenomena. This paper attempts to bridge this gap.

Problem statement

As pointed out, mesh adaptation is now widely used in complex three-dimensional numerical simulations, especially in CFD. The aim of mesh adaptation is to control the generation of a new mesh in a computational scheme, such that the computational error (the approximation error) estimated on this mesh is bounded by a given threshold value. If we consider that the approximation error is bounded by the interpolation error, the problems turns to generate a new mesh on which the interpolation error is equidistributed.

Here, we attempt to glue various software pieces together, in order to build a fully automatic anisotropic mesh adaptation scheme suitable for unsteady problems. This challenging application is based on the definition of a geometric error estimate of the interpolation error, on the construction of a proper discrete metric tensor and on governed anisotropic surface and volume meshing algorithms. All these pieces have one feature in common: they all involve an anisotropic metric. Anisotropic mesh generators require a metric to be defined at the mesh vertices in order to create adapted meshes. The discrete metric is defined at the mesh vertices in order to account for the variations of (possibly all) the variables of the problem.
So, the error estimate has to relate the interpolation (or approximation) error to element sizes and directions, using a ... metric. It turns out that defining a suitable metric is the key to success in mesh adaptation.

Related work

Over the last few years, a rather large number of papers have been published on mesh adaptation for numerical simulations (see [14] for a survey). However, only a few number of papers have addressed the problem of constructing three-dimensional anisotropic meshes. Actually, most of these papers mainly suggested to modify the point insertion procedure, common to all meshing algorithms, to account for boundary layers (i.e., close to the boundaries). Although some of these algorithms may be useful in such circumstances, all of them usually lack the kind of generality (or automation) required to capture, for instance, shocks waves in the regions away from the domain boundaries. Indeed, fully automatic tools are still crucially required in the context of mesh generation and error estimation. Once available, such tools would greatly facilitate the design and implementation of an automatic adaptation scheme, for any type of numerical simulation (the solver would be the only application dependent software piece).

Paper outline

In this paper, we briefly introduce an a posteriori geometric error estimate based on a discrete approximation of the second derivates of the variables of the problem that will be used to define a suitable metric for mesh adaptation, Section 1. In Section 2, we recall the main stages of the mesh adaptation scheme, i.e., surface and volume governed mesh generation. In Section 3, we present a new adaptation scheme based on a transient fixed point algorithm suitable for transient problems. Finally, in Section 4, three application examples of mesh adaptation are provided to illustrate the proposed approach, in the isotropic and anisotropic case.

1. METRIC RELATED ISSUES

In many engineering applications, it is often desirable to create adapted meshes presenting highly anisotropic features (stretched elements in arbitrary directions). This challenging problem can be (easily) resumed to that of constructing appropriate metric tensors in order to govern the meshing algorithms. The key idea is thus to define a convenient metric tensor (via a suitable error estimate) based on the discrete variables (solutions) of the problem. Such a discrete metric can be used to create unit meshes as will be seen in the next Section.

In this Section, we will first introduce the main principle of a geometric error estimate of the interpolation error. Then, we will explain how to construct a metric tensor for mesh adaptation.

1.1 A geometric error estimate

As the finite element solution \( u_i \) is not interpolant and as it is not possible to guarantee that \( u_i \) coincides with the exact solution \( u \) in at least one point of each element, it seems rather difficult to quantify the gap \( e_i = u - u_i \). However, it is possible to use an indirect approach to measure this gap [1]. For elliptic problems, it has been proved (Céa’s lemma, [12]) that the FE error is bounded by the interpolation error \( e_i : \| e_i \| \leq C \| u - \Pi_h u_i \| \), where \( \Pi_h u_i \) is the interpolation of \( u \) on the mesh \( H_i \). We assume that this relation still holds in the class of problems described here. Actually, studies based on the interpolation error show (practically) that the link between the interpolation error and the approximation error is even stronger than the bound given by Céa’s lemma. Hence, the interpolation error leads to a "good" error estimate [14]. The problem is to characterize the mesh on which the interpolation error is bounded by a given tolerance value or equidistributed.

It has been proved that the analysis of a “measure” of the interpolation error leads to define an anisotropic metric map which prescribes element sizes and directions [7]. To measure the interpolation error, we consider the discrete \( L_\infty \) norm of the error defined in tetrahedron \( K \) as:

\[
\| u - \Pi_h u_i \|_{\infty,K} \leq c \max_{x \in K} \max_{e \in E_K} (|e| |H_u(x)|) (1)
\]

where \( c \) is a constant dependent of the dimension, \( E_K \) is the set of element edges of \( K \) and \( |H_u| \) is the absolute value of the Hessian of the variable \( u \) (symmetric definite positive tensor). As the Hessian matrix is symmetric, it can be decomposed as: \( H_u = R \Lambda R^{-1} \) where \( R \) is the eigenvector matrix and \( \Lambda = \text{diag}(\lambda_i) \) is the eigenvalue matrix, the absolute value of the Hessian matrix is then
defined as follows:

\[ |H_u| = |\mathcal{R}| \Lambda|\mathcal{R}^{-1} \quad \text{with } |\Lambda| = \text{diag}(|\lambda_i|). \]

Notice that this error is related to the Hessian of the variable \( u \) and to the mesh edges, hence it provides directional thus anisotropic information. Controlling the mesh edges allows to control the interpolation error on the mesh elements.

From a practical point of view, the right-hand side term of Equation (1) is not useful as it involves the maximum of the metric field \( |H_u| \) that is usually not known. Nevertheless, it is possible to define a suitable metric tensor \( \tilde{\mathcal{M}} \) such that the interpolation error on a mesh element is given by:

\[ \varepsilon_K = c \max_{\vec{e} \in E_K} \langle \vec{e}, \tilde{\mathcal{M}}(K) \vec{e} \rangle. \]

In the mesh adaptation context, the error tolerance \( \varepsilon \) that must be equidistributed over the mesh is fixed and the mesh element have to be characterized via this constraint. For a given mesh element \( K \), we define the metric tensor \( \mathcal{M}(K) = c \varepsilon^{-1} \tilde{\mathcal{M}}(K) \), and all mesh edges must comply with the following equality:

\[ \langle \vec{e}, \mathcal{M}(K) \vec{e} \rangle = 1. \quad (2) \]

A mesh for which all edges comply with this relationship is a so-called unit mesh.

In other words, to equidistribute the interpolation error over the mesh, we have modified the scalar product that lies under the notion of distance used in mesh generation algorithms, based on the local metric \( \mathcal{M} \) that replace the usual Euclidean metric. This tensor \( \mathcal{M} \) must still be defined more precisely.

The specificity of this error estimate is related to the following features:

- the analysis is not asymptotical, \( (h \) does not tend towards zero),
- it is based on the hessian of the solution,
- it is intrinsically anisotropic
- it does not depend of the nature of the operator (therefore it can be used for any type of equation).

**Remark 1.1.** This error estimate is called "geometric" as the solution on a mesh can be seen as a Cartesian surface and we attempt to define a geometric metric in order to control the gap to the surface.

### 1.2 Metric construction

Let us denote by \( h_{min} \) (resp. \( h_{max} \)) the minimal (resp. maximal) mesh element size. According to the previous section, we define the metric tensor as: \( \mathcal{M} = \mathcal{R} \tilde{\mathcal{M}}^{-1} \), with:

\[ \lambda_i = \min \left( \max \left( \frac{c|\lambda_i|}{\varepsilon}, \frac{1}{h_{max}^2} \right), \frac{1}{h_{min}^2} \right). \]

Introducing a minimal (resp. maximal) element size is a way of avoiding unrealistic (unpracticable) metrics. It also allows in a computational (explicit) scheme to control the time step.

The Equation (1) leads to a global upper bound of the interpolation error. However, in order to combine various variables, each of them having a different meaning or a different nature, it becomes necessary to introduce a relative bound on the interpolation error, to have dimensionless variables, as follows:

\[ \frac{\| u - \Pi_h u \|_{\infty,K}}{\| u \|_{\infty,\Omega}} \leq c \max_{x \in K} \max_{\vec{e} \in E_K} \langle \vec{e}, |H_u(x)\vec{e}| \rangle \frac{\| u \|_{\infty,\Omega}}{\| u \|_{\infty,\Omega}}. \]

Then, all metrics can be combined together into a single metric tensor using a metric intersection scheme.

Moreover, in numerical simulations, solutions vary from several orders of magnitude (multi-scale phenomena, recirculations, shocks, etc.). It is thus difficult to capture the weakest phenomena via mesh adaptation, and even harder to do it when, for instance in CFD, shocks are located in the flow. A local error estimation can overcome this problem. Following the previous idea, the error estimate is also normalized using the local value of the gradient norm of the variable \( u \), weak phenomena can be captured even in presence of strong shocks. To this end, we introduce the following estimate:

\[ \frac{\| u - \Pi_h u \|_{\infty,K}}{\| u \| + h \| \nabla u \|_2} \leq c \max_{x \in K} \max_{\vec{e} \in E_K} \langle \vec{e}, \frac{|H_u(x)|}{|u| + h \| \nabla u \|_2} \vec{e} \rangle, \]
where $\bar{h}$ is the diameter of the element (its largest edge) in the background mesh (at the previous iteration).

### 1.3 Metric intersection

When several metrics are specified at the same vertex, a unique metric tensor must be defined taking into account all given metrics. To this end, a metric intersection procedure is used. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two metric tensors given at a vertex $P$. The metric tensor $\mathcal{M}_{1\cap 2}$ corresponding to the intersection of $\mathcal{M}_1$ and $\mathcal{M}_2$ must be such that the interpolation error for each variable is bounded by the given tolerance value. To this end, we use the simultaneous reduction of the quadratic forms associated with the two metrics [1] (cf. Figure 1).

The two metric tensors are represented by the associated ellipsoids $\mathcal{E}_{\mathcal{M}_i}$. The ellipsoid $\mathcal{E}_{\mathcal{M}}$ of maximal volume included in the (geometric) intersection of these two ellipsoids defines the desired metric tensor. Formally speaking, let us consider

$$M_d = \{ \mathcal{M} | \mathcal{M} \text{ metric tensor} \}$$

be the set of all metric tensors in $\mathbb{R}^d$ and let us define the ellipsoid associated with the metric $\mathcal{M}$ by:

$$\mathcal{E}_{\mathcal{M}} = \{ M | \sqrt{t P M P M} = 1 \}.$$

Hence, the metric $\mathcal{M}_{1\cap 2}$ is defined by the ellipsoid $\mathcal{E}_{\mathcal{M}_{1\cap 2}} = \sup_{M_i \in M_d} \mathcal{E}_{\mathcal{M}_i} \subset \mathcal{E}_{\mathcal{M}_1} \cap \mathcal{E}_{\mathcal{M}_2}$ or:

$$\sup_{M_i \in M_d} \left\{ M | \sqrt{t P M P M} = 1 \right\} \subset \mathcal{E}_{\mathcal{M}_1} \cap \mathcal{E}_{\mathcal{M}_2},$$

where the sup over the set of metrics represents the metric with the largest volume of its associated ellipsoid.

### 2. MESH ADAPTATION

As mentioned in the Introduction, the generation of an adapted mesh is based on the specification of a discrete anisotropic metric tensor at each mesh vertex of the current mesh. The aim is then to compute the edge lengths with respect to this metric. For the sake of simplicity, it is possible to define the metric tensor so as to prescribe a unit edge length. The standard Euclidean scalar product is then modified using a proper metric tensor field. At each vertex, a different expression of the metric $\mathcal{M}$ leads to a different expression of the scalar product. Let $P$ be a vertex and let $\mathcal{M}(P)$ be the metric at $P$. The desired edge $PX$ must have a length close to one w.r.t $\mathcal{M}(P)$:

$$l_{\mathcal{M}(P)}(P\bar{X}) = \sqrt{t PX M(P) PX} = 1.$$  

As the metric varies in the domain (is not constant in an element), we need to consider the metrics at the edge endpoints as well as all intermediate metrics along the edge. To this end, we introduce the average length of $PX$ as:

$$l_{\mathcal{M}}(P\bar{X}) = \frac{1}{\int_0^1 \sqrt{t P X \mathcal{M}(P + t P X) P X}},$$  

The desired adapted mesh is then a unit mesh, i.e., a mesh such that for each edge $\bar{e} \in E_K$, $l_{\mathcal{M}}(\bar{e}) \approx 1$.

In our approach, the generation of adapted meshes is a two-steps process. At first the surface mesh is adapted using local modifications [16], then the volume mesh is adapted using a constrained Delaunay algorithm extended to the anisotropic case [19]. Notice that most of the vertices of the previous meshes are kept in order to reduce errors (as much as possible) when interpolating the solutions from one mesh to the other.

#### 2.1 Surface mesh adaptation

Given a discrete surface (a piecewise linear approximation of the domain boundaries) and a discrete
metric field, the aim is to generate an adapted mesh with respect to this metric. To this end, the approach we use consists in modifying iteratively the initial surface mesh so as to complete a unit mesh. Obviously, as the mesh is intended for FE computations, the mesh gradient is also a major concern. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

As no CAD information is provided, an internal relaxation, node removal, node repositioning and degree repositioning ans local mesh optimization procedures. The ingredients to achieve this goal typically include mesh enrichment, mesh coarsening and local mesh optimization procedures. The local mesh modifications operators involved are: edge flipping, edge collapsing, edge splitting and node removal, node repositioning and degree relaxation.

The surface mesh modification algorithm is pretty straightforward, edge lengths are computed with respect to the metric $\tilde{\mathcal{M}}$ and edge too small are collapsed while edge too long are split into unit length segments. Edge flips and node repositioning operations are performed to improve the overall mesh quality (in terms of shape and size) [17].

### 2.2 Volume mesh adaptation

Once the surface mesh has been adapted, a unit volume mesh is generated with respect to the modified metric $\tilde{\mathcal{M}}$. In our approach a constrained Delaunay procedure is used to build first an empty Delaunay mesh (with no internal vertices). Then, based on an edge length analysis, internal nodes are added into the current mesh (most of them coming from the background mesh, at the previous iteration) using the Delaunay kernel, extended to the anisotropic case [19].

As pointed out, the classical distance evaluation is replaced using an evaluation related to the local anisotropic metric. Hence, let $A$ and $B$ be two points, the distance between $A$ and $B$, $d(A, B)$ is now replaced by $l_{\mathcal{M}}(A, B)$ as defined by Equation (3). Then, $O_K$ (the circumcenter of a tetrahedron) is computed as the solution of the system:

$$l_{\mathcal{M}}(O_K, P_i) = l_{\mathcal{M}}(O_K, P_j) \quad \forall i, j = 1, 4, i \neq j,$$

and $r_K$, the circumradius of $K$, is computed as:

$$r_K = l_{\mathcal{M}}(O_K, P_j).$$

Finally, the Delaunay measure:

$$\alpha_{\mathcal{M}}(P, K) = \frac{l_{\mathcal{M}}(O_K, P)}{l_{\mathcal{M}}(O_K, P_j)} < 1,$$

where $\alpha_{\mathcal{M}}(P, K)$ is used to define the cavity of $K$.

However, as we face a nonlinear system to compute $O_K$ and as the metric is discrete, it is then not so easy to compute the desired length. Therefore, approximations are needed to return to the Euclidean context. To this end, we fix the metric and various approximations can be used.

### 2.3 Unsteady adaptation scheme

The classical adaptation algorithm is usually composed of four successive steps: (i) solution computation, (ii) error estimation and metric construction, (iii) governed mesh adaptation and (iv) interpolation of the solutions. However, this approach has been proved to be inadapted when dealing with transient (time-dependent) problems [2], mainly because of the impossibility to predict the behavior or the evolution of such phenomena. For stationary problems, mesh adaptation aims to find a fixed point for the couple mesh-solution. The objective is to converge towards both the stationary solution of the problem and the adapted mesh (that equidistribute the interpolation error). In such context, the mesh is always "late" with respect to the phenomenon and thus, if the number of adaptations is too small, the solution is diffused. Moreover, the adaptation introduce a quite large error due to the interpolation of the solution from one mesh (the background mesh) to the other, that impacts the time derivatives and thus introduce an error in phase. To overcome these problems, we have proposed an extension of the classical adaptation scheme, specifically intended for transient problems [2].

The transient fixed point algorithm is composed of two steps, the main adaptation loop and an internal loop in which the transient fixed point problem
is solved (Fig. 5). At each iteration of the main loop, we consider a time period \([t, t + \Delta t]\) in which the solution evolves. From the solution at time \(t\), we compute the solution to time \(t + \Delta t\), and the computation is iterated until the desired accuracy is obtained for the solution at \(t + \Delta t\). Hence, the solution behavior is predicted in all regions of the domain where it evolves. A metric intersection procedure in time is introduced to adapt the mesh in these regions.

To generate an adapted mesh in the region where the solution advances in time, the metric must reflect this evolution. To this end, the metric must take into account the information given by the initial solution, the final solution and all intermediate solutions. Therefore, all metrics are intersected and the metric tensor at iteration \(i\) and the internal loop \(j\) is defined as:

\[
\mathcal{M}_{i,j} = \bigcap_{k=1}^{m} \mathcal{M}_{i,j}^{k},
\]

where \(\mathcal{M}_{i,j}^{k}\) is the \(k\)th intermediate metric given by the solution.

## 3. APPLICATION EXAMPLES

In this section, we present three examples of unstructured mesh adaptation intended to illustrate the various problems described previously. The first example aims at showing anisotropic surface and volume mesh adaptation for an analytical metric specification. The second example will focus on the error estimate for a steady-state Euler computation in two dimensions. Finally, the third example deals with a transient CFD problem on a complex geometry in three dimensions.

### 3.1 A “not-so-simple” analytical example

In this example, we will show the behavior of the anisotropic adaptation meshing procedure on an analytical function and, more precisely, the metric intersection algorithm. Let us consider the surface in \(\mathbb{R}^3\) defined on \([-1, 1]^3\):

\[
f_1(x, y, z) = \tanh\left((x + 1.3)^{20}(y - 0.3)^9z\right),
\]

the computational domain is a supertorus defined by the set of equations:

\[
\begin{align*}
x &= \cos^n(\theta)\left(r_0 + r_1\cos^n(\phi)\right), \\
y &= \sin^n(\theta)\left(r_0 + r_1\cos^n(\phi)\right), \\
z &= (r_0 + r_1)\sin^n(\phi),
\end{align*}
\]

where \(\theta\) and \(\phi\) vary in \([0, 2\pi]\) and \(r_0 + r_1\) (resp. \(r_0 - r_1\)) represents the external (resp. internal) radius of the torus (here \(n_1 = n_2 = 0.2\)).

The specificity of this case is that it combines two different metric fields, the first one defined by the function \(f_1\), the second one related to the geometry of the domain (the intrinsic surface properties). Here, the surface meshing algorithm proceeds by first analysing the surface given an initial (crude) surface triangulation (Fig. 2) in order to construct the geometric metric tensor\(^1\). The number of adaptations has been set \(a \text{ priori}\) to 8, the desired error bound is \(\varepsilon = 0.0084\). The Hessian of the variable (the surface curvature or the function) is computed a least-squares approximation [16]. Table 1 reports statistics about the initial triangulation and the sequence of adapted meshes, \(np, nc, nf\) representing the number of vertices, tetrahedra and triangles, respectively, \(\bar{\varepsilon}\) and \(\varepsilon_{\text{max}}\) represent the average and maximal error measured on the elements.

The distance to the hypersurface is computed on each tetrahedron \(K\) by considering the maximal value of the distances between the edge midpoints, the face midpoints and the barycenter of \(K\). The average (resp. maximal) value of the error should be close to the desired error \(\varepsilon\) on the final adapted mesh, thus validating the proposed error estimate.

<table>
<thead>
<tr>
<th>It</th>
<th>(np)</th>
<th>(nc)</th>
<th>(nf)</th>
<th>(\bar{\varepsilon})</th>
<th>(\varepsilon_{\text{max}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3,850</td>
<td>15,460</td>
<td>4,884</td>
<td>0.0438</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>72,090</td>
<td>416,124</td>
<td>14,548</td>
<td>0.0064</td>
<td>0.5673</td>
</tr>
<tr>
<td>4</td>
<td>29,651</td>
<td>154,763</td>
<td>17,076</td>
<td>0.0047</td>
<td>0.0555</td>
</tr>
<tr>
<td>5</td>
<td>22,782</td>
<td>114,637</td>
<td>16,190</td>
<td>0.0051</td>
<td>0.0454</td>
</tr>
<tr>
<td>8</td>
<td>21,063</td>
<td>105,268</td>
<td>15,474</td>
<td>0.0051</td>
<td>0.0345</td>
</tr>
</tbody>
</table>

**Table 1.** Statistics about the adapted meshes at iterations 0 (initial), 2, 4, 5 and 8 (final) for the analytical example.

\(^1\)Needless to say that the analytical function is not used to get the first and second derivatives of the surface at the mesh vertices, discrete approximations are computed instead [16].
Remark. In this example, the final adaptation, 99.6\% of the elements have an error below the given tolerance $\varepsilon$ and the average error $\bar{\varepsilon} = 0.0051$ is lesser than $\varepsilon$. Moreover, the maximal error $\varepsilon_{\max}$ is constantly decreasing over the iterations.

3.2 A 2D steady-state problem

This example related to a Euler computation at Mach 3 in a scramjet configuration is typical of numerical simulations in compressible fluids, involving highly anisotropic phenomena (shocks). The aim is to capture the behavior of the physical phenomenon and to emphasize the reduction of the number of degrees of freedom obtained thanks to the anisotropy [9]. The geometry of the computational domain is shown in Fig. 6.

Two series of mesh adaptation have been carried out on this example, isotropic and anisotropic, to be compared. However, the parameters were the same for both computations: 9 adaptations have been performed, each 400 time steps of the Euler solver. The density variable has been chosen to contain more than 5 times less vertices than the final isotropic mesh. Accordingly, the anisotropic approach required 14\(mn\) for the isotropic approach, thus reducing the time by a impressive factor of 9. Moreover, by stretching the elements along the discontinuities, the numerical diffusion due to the Riemann solver has been significantly reduced (Fig. 8).

Remark 3.1. The same test case was already proposed a few years ago. However, the anisotropic mesh adaptation was giving nearly isotropic elements along the shock waves [9]. The result is now improved using the geometric error estimate described in Section 1.

3.3 A 3D transient CFD problem

Finally, the last example will demonstrate the efficiency of the transient adaptation procedure, on a

<table>
<thead>
<tr>
<th>It.</th>
<th>$n_p^A$</th>
<th>$n_e^A$</th>
<th>$n_p^I$</th>
<th>$n_e^I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8,012</td>
<td>15,275</td>
<td>8,012</td>
<td>15,275</td>
</tr>
<tr>
<td>2</td>
<td>5,783</td>
<td>11,055</td>
<td>76,565</td>
<td>151,558</td>
</tr>
<tr>
<td>5</td>
<td>9,292</td>
<td>17,983</td>
<td>75,997</td>
<td>150,532</td>
</tr>
<tr>
<td>9</td>
<td>15,110</td>
<td>29,569</td>
<td>78,702</td>
<td>155,951</td>
</tr>
</tbody>
</table>

Table 2. Statistics about the initial and adapted meshes for the anisotropic and isotropic cases on the scramjet configuration.

An isotropic example\(^2\). The problem concerns a non linear wave propagation in a complex 3D geometry. This simulation can be seen as a generalisation of the Riemann problem (a shock tube) in higher dimension. More precisely, a Heaviside perturbation is introduced into a uniform field so as to simulate an explosion (a region of high pressure is introduced into the ambient atmosphere). The flow is approached using Euler equations in the conservative forms. A finite volume (for the flow computation) solver is used. Euler equations are solved using an explicit scheme, a four order Runge-Kutta is used for time integration.

In this example, the objective is to compute the solution at physical time $t = 0.1$ sec. The simulation is decomposed into 30 periods (adaptations). At each main iteration, 4 internal iterations are performed to solve the transient fixed point problem. The metric is defined based on the density variations. The adaptation parameters have been set to: $\varepsilon = 0.01$, $h_{\text{min}} = 0.3m$, $h_{\text{max}} = 10m$, for a computational domain size of $85m \times 85m \times 70m$. Table 3 reports statistics about the adapted (isotropic) meshes. Adapted meshes are presented in Figures 9 to 13.

<table>
<thead>
<tr>
<th>It.</th>
<th>$t$</th>
<th>$n_p$</th>
<th>$n_e$</th>
<th>$n_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.027</td>
<td>280,525</td>
<td>1,630,619</td>
<td>40,736</td>
</tr>
<tr>
<td>15</td>
<td>0.05</td>
<td>603,644</td>
<td>3,541,268</td>
<td>63,526</td>
</tr>
<tr>
<td>23</td>
<td>0.077</td>
<td>739,854</td>
<td>4,326,861</td>
<td>78,816</td>
</tr>
<tr>
<td>30</td>
<td>0.1</td>
<td>743,735</td>
<td>4,328,741</td>
<td>87,322</td>
</tr>
</tbody>
</table>

Table 3. Statistics about the adapted meshes for the transient problem.

\(^2\)Anisotropy is expected soon in 3D.
4. CONCLUSIONS

In this paper, we have presented a global scheme for mesh adaptation in the context of numerical simulations. The proposed approach involves an anisotropic geometric error estimate, surface and volume mesh adaptation algorithms based on discrete metric tensors. It differs from the classical adaptation scheme by integrating a inner loop corresponding to a transient fixed point algorithm.

REFERENCES


Figure 2. Initial and final adapted surface meshes for the analytical metric. Cut through the volume to show anisotropic tetrahedra within this domain.

Figure 3. Impact of the metric intersection scheme (initial and final solution during a period).

Figure 4. Comparison of the adapted solution with a reference solution (uniform mesh) in red. Green: 6 metric intersection, blue: 2 metric intersection, magenta: only the initial solution, cyan: only the final solution.
Figure 5. Global overview of the modified mesh adaptation scheme for transient simulations.

Figure 6. Initial and adapted meshes at the iterations 2 and 9 and corresponding isodensity distributions for the scramjet configuration.
Figure 7. Final anisotropic mesh for the scramjet configuration.

Figure 8. Anisotropic vs. isotropic mesh comparison for the final meshes (local zoom).
Figure 9. Geometric surface mesh (left-hand side) and initial computational surface mesh (right-hand side).

Figure 10. Isodensity surfaces at times $t = 0.027$ sec, $t = 0.05$ sec, $t = 0.077$ sec and $t = 0.1$ sec.
Figure 11. *Isotropic adapted surface meshes and isodensity distributions at times* \( t = 0.027 \text{ sec}, \ t = 0.05 \text{ sec}, \ t = 0.077 \text{ sec}\) and \( t = 0.1 \text{ sec}.\)

Figure 12. *Isotropic adapted volume meshes and cutting plane through the domain showing the isodensity distributions at times* \( t = 0.027 \text{ sec}, \ t = 0.05 \text{ sec}, \ t = 0.077 \text{ sec}\) and \( t = 0.1 \text{ sec}.\)
Figure 13. Two cutting planes through the adapted volume mesh at time $t = 0.077$ sec and corresponding isodensity distributions.
Session 4
Smoothing
MULTILEVEL ACCELERATED OPTIMIZATION FOR PROBLEMS IN GRID GENERATION

Markus Berndt$^1$  Mikhail Shashkov$^2$

$^1$Los Alamos National Laboratory, Los Alamos, NM, U.S.A. berndt@lanl.gov
$^2$Los Alamos National Laboratory, Los Alamos, NM, U.S.A. shashkov@lanl.gov

ABSTRACT

The quality of numerical simulations of processes that are modeled by partial differential equations strongly depends on the quality of the mesh that is used for their discretization. This quality is affected, for example, by mesh smoothness, or discretization error. To improve the mesh, a functional that is in general nonlinear must be minimized (for example, the $L^2$ approximation error on the mesh). This minimization is constrained by the validity of the mesh, since no mesh folding is allowed. Classical techniques, such as nonlinear CG, or Gauss-Seidel steepest descent, perform very poorly on this class of minimization problems. We introduce a new minimization technique that utilizes the underlying geometry of the problem. By coarsening the mesh successively, in a multilevel-like fashion, minimizing appropriate coarse grid quality measures, and interpolating finer meshes from coarser ones, a more rapid movement of fine mesh points results, and the overall convergence of the minimization procedure is accelerated.

Keywords: mesh generation, optimization, multilevel methods

1. INTRODUCTION

Grid generation often requires the minimization of a functional that describes mesh smoothness or approximation quality of a specific function. Optimization methods that are commonly used in this context are of the steepest descent type [1, 2]. These methods perform very poorly, if vertices, or clusters of vertices have to move large distances from an initial grid to reach a final optimal configuration.

Our multilevel approach effectively achieves an accelerated vertex movement by coarsening the grid and then solving an appropriate optimization problem on the coarsened grid. The coarse grid is then interpolated to the fine grid. On a coarser grid, vertices can move larger distances than on fine grids, without the mesh becoming invalid. By interpolating an improved coarse grid to a finer grid we effectively move clusters of fine grid vertices by moving only a single coarse grid vertex. We apply the idea of coarsening a fine grid, approximately solving an appropriate coarse grid optimization problem, and interpolation the resulting grid back to the fine level recursively. This describes our multilevel accelerated optimization procedure.

In this paper, we first give an overview of classical multigrid methods to motivate the applicability of multilevel-type ideas in the context of optimization problems that arise in the context of grid generation. We then introduce our target optimization problem and construct the multilevel components coarsening and interpolation. After defining the coarse grid optimization problems we define the multilevel optimization procedure, give a complexity analysis and a numerical example to underline the performance gain that can be achieved with this type of approach to optimization.

2. OVERVIEW OF MULTILEVEL METHODS

Multilevel methods are the most efficient methods for solving linear systems that typically arise in the
discretization of elliptic partial differential equations (PDEs). A good introduction to these methods is given in the Multigrid Tutorial [3]. For a more comprehensive overview, see [4]. Here, we give a short introduction of multilevel methods, to motivate their applicability to some grid generation problems.

One class of methods for solving linear systems is called relaxation methods. As an example, we introduce Jacobi relaxation. A given linear system

\[ A\mathbf{x} = b \]  

(1)

can be re-written, using the diagonal \( D \), the lower triangular part \( L \), and the upper triangular part \( U \) of \( A \) in the following way,

\[ D\mathbf{x} = b - L\mathbf{x} - U\mathbf{x} . \]  

(2)

Equation (2) gives rise to the iterative procedure commonly referred to as Jacobi relaxation,

\[ x_{n+1} = D^{-1}(b - Lx_n - Ux_n). \]  

(3)

For the class of linear systems that arise in the discretization of elliptic PDEs, it is easily seen that iteration (3) converges to the solution \( x^* \) of equation (1).

The obvious advantage of such an iteration is its simplicity and therefore its ease of implementation. However, the convergence of such relaxation methods is typically prohibitively slow. To illustrate the convergence behavior, we discretize

\[ u'' = 0, \quad \text{on } (0,1), \]  

(4)

with homogeneous Dirichlet boundary conditions \( u(0) = u(1) = 0 \), using finite differences on a regular grid. Note that in this example, an iterate equals the error.

Figure 1 shows an oscillatory initial guess (top), the solution after one Jacobi relaxation (middle), and the error after five Jacobi relaxations (bottom).

The figure illustrates that already after one Jacobi relaxation, the high frequency error components are gone (middle), and after five relaxations only a very low frequency error remains (bottom).

An important observation is that the notion of frequency is linked to the mesh size. The function in the bottom graph in Figure 1 can be adequately represented on a mesh consisting of five equally spaced grid points. On such a grid, this function can be interpreted as being of high frequency. Jacobi relaxation would dampen it in a couple of iterations. This observation motivates multilevel methods.

We first note that, if \( \mathbf{x} \) is any vector and \( r = b - A\mathbf{x} \) the residual, then solving for the correction \( \hat{\mathbf{x}} \)

\[ A\hat{\mathbf{x}} = r \]  

(5)

enables us to write down the solution \( x^* = \mathbf{x} + \hat{\mathbf{x}} \). Using a relaxation method on the finest grid, transferring the residual to a coarser grid, solving the correction equation there, and correcting the fine grid solution with an interpolation of the correction from the coarse grid, summarizes a two grid method. Of course, this two grid method can be used to solve the coarse grid linear system (if the coarse grid can be further coarsened). This constitutes the recursive definition of the multigrid method.

Figure 2 illustrates one iteration of the multilevel method. Relaxation is represented by circles, restriction and interpolation are represented by arrows that are pointing down or up, respectively, and the coarse grid solve is represented by a square. Because of the characteristic shape of this diagram, the iteration is
called V-cycle.

The most important property of the multilevel method is that it is typically of complexity $O(N)$ for the V-cycle, where $N$ is the number of unknowns. In other words, it is scalable, whereas most other known linear solvers are not.

3. OPTIMIZATION IN GRID GENERATION

Many problems in grid generation can be stated as a discrete nonlinear optimization problem, where the objective function has the form

$$ F(G) = \sum_{\text{vertices}} F_{\text{loc}}(G). \quad (6) $$

Here, $G$ is the grid. An example for this is the reference Jacobian objective functional that is used in mesh smoothing (see, for example, [1]).

Methods that are commonly used to solve these types of optimization problems include nonlinear conjugate gradient methods, Gauss-Seidel Newton, and Newton Gauss-Seidel. These methods have two common flaws. Their convergence is typically slow, and they do not scale well with the size of the mesh.

To exemplify the problems that all these methods exhibit, we briefly describe the Gauss-Seidel Newton method. Denote by $\{x^{(0)}_i\}_{i=1,...,n}$ the vertices, and by $E$ the edges in the initial grid $G^{(0)} = (\{x^{(0)}_i\}_{i=1,...,n}, E)$. The $k$-th iteration is as follows.

1. Loop over the vertices $i = 1, \ldots, N$ and solve for each $i$ the local minimization problem

   $$ x^{(k)}_i = \arg\min_x F(x^{(k-1)}_1, \ldots, x^{(k)}_{i-1}, x_i, x^{(k-1)}_{i+1}, \ldots, x^{(k-1)}_N) $$

   by Newton's method.

2. Set $G^k = (\{x^{(k)}_i\}_{i=1,...,n}, E)$.

Of course, in each Newton step, the new optimal position for vertex $i$ can only be chosen inside a set that preserves mesh validity. We refer to this set at the feasible set. The problem with this approach is that if the optimal grid is far away from the initial grid, the convergence will be very slow.

We propose an optimization algorithm that is based on ideas from multilevel methods. These methods build on the flaws that the aforementioned methods have, in much the same way, that multilevel methods build on the flaws of simple relaxation methods.

4. COARSENING AN UNSTRUCTURED TRIANGULAR GRID – THE RESTRICTION OPERATOR

The coarsening procedure that we present is very similar to Delaunay-Coarsening (DC), introduced in [5]. In an initial step in DC, the list of vertices is reordered in such a way that all boundary vertices come first. In a loop over the vertices in this list the current vertex is added to the list of coarse vertices and its neighbors are deleted from the list of vertices. Hence, initially the boundary is coarsened, and then the interior. As a slight modification of this algorithm, we consider such boundary vertices first that are necessary to properly resolve the shape of the domain. An example for such vertices are the four corner vertices of a square.

An alternative coarsening algorithm that could be employed in the context of our algorithm is presented in [6]. This algorithm is based on edge contraction and it works in two as well as in three dimensions. For the purposes of this paper, it was simpler to use a strategy that builds on readily available software for two dimensional Delaunay triangulation. We now formalize our coarsening procedure.

We will need the notion of distance for vertices in a grid. To that end, we first define a path $P_{i,j}$ connecting vertices $x_i$ and $x_j$ through a grid as a sequence of edges $\{e_{ij}\}_{i=1,...,K}$, where $e_{ij}$ and $e_{i,j+1}$ are adjacent, and $e_1$ is adjacent to $x_i$, and $e_{i,K}$ is adjacent to $x_j$. We use the notation $|P_{i,j}| = K$ to denote the length of $P_{i,j}$. Now, denote by $d_G(x_i,x_j)$ the distance function for a particular grid. It is defined as

$$ d_G(x_i,x_j) = \min\{|p| : p \in P_{i,j} \}, \quad (7) $$

where $P_{i,j}$ is the set of all paths that connect vertices $x_i$ and $x_j$.

As a first step in our coarsening procedure, we add all fine vertices to the set of available vertices $L = \{x_i\}_{i=1,...,N}$, and initialize the set of coarse vertices as the empty set: $C = \emptyset$. In the next step, we mark all boundary vertices that are necessary to properly resolve the geometry of the domain as coarse vertices.
and remove them from $L$: If $x_i^f \in L$ is such a vertex, let $L \leftarrow L \setminus \{x_i^f\}$, and let $C \leftarrow C \cup \{x_i^f\}$. Then, all of their neighbors, that is, all fine vertices $x_j^f \in L$ for which $d_G(x_j^f, x_i^f) = 1$, for any $x_i \in C$, are removed from $L$.

In a second step, we loop over the remaining boundary vertices in $L$. The current vertex $x_i^f$ in this loop is removed from $L$: $L \leftarrow L \setminus \{x_i^f\}$, and added to $C$: $C \leftarrow C \cup \{x_i^f\}$. Then all neighbors of $x_i^f$ in $L$ are also removed from $L$.

All other coarse vertices are determined in a greedy algorithm to find a maximal independent set. Pick any vertex $x^I \in L$, mark it as a coarse vertex $C \leftarrow C \cup \{x^I\}$, and remove it from $L$: $L \leftarrow L \setminus \{x^I\}$. Then, remove all its neighbors from $L$: For all $\tilde{x}^I \in L$, such that $d_G(x^I, \tilde{x}^I) = 1$, $L \leftarrow L \setminus \{\tilde{x}^I\}$. The algorithm terminates, when $L$ is empty.

We now have a set $C$ of vertices that are marked as coarse. To create a coarse grid from these, we use a the classic sweep line algorithm by S. Fortune (see [7]) to generate a Delaunay triangulation. For the simple domains we have considered so far, there have been no problems with boundary preservation. As a fix we plan to use constrained Delaunay.

Figure 3 illustrates the coarsening process. Given a fine grid (top), a maximal independent set of coarse points is determined (middle), from which a Delaunay triangulation is generated (bottom).

### 4.1 Restriction of the fine to the coarse grid

The described procedure yields a coarse grid, as well as a relationship between this coarse and the initial fine grid. We use this relationship to define the restriction operator. We keep the connectivity of both the fine and the coarse grid fixed. When the fine grid moves, the coarse grid moves along with it by virtue of each coarse vertex having the same location as its corresponding fine vertex. We call the process of updating the coordinates of the coarse grid vertices to the values of their corresponding fine grid vertices the restriction of the fine to the coarse grid.

### 5. Interpolating a Fine Grid from a Coarse Grid

In the previous section, we have introduced the restriction operator. After a coarse grid is created, we record for each fine vertex what its relation to the coarse grid is. There are two possible cases.

- First, a fine vertex $x^f$ was marked as coarse in the coarsening procedure ($x_i^f \in C$), and

- second, a fine vertex was discarded in the coarsening procedure ($x^f \notin C$).

In the first case, we interpolate by injection. More specifically, we replace the coordinates of the fine vertex with the coordinates of the coarse vertex. In the second case, we initially find the coarse triangle $\tau^c$, such that the coordinates of the fine vertex $x^f$ are inside of $\tau^c$. Then, we store the barycentric coordinates $(\xi_{x^f, \tau^c}, \xi_{x^f, \tau^c}, \xi_{x^f, \tau^c})$ of vertex $x^f$ inside of $\tau^c$. To interpolate the coordinates of $x^f$ from the coarse grid after the coarse grid has changed, we compute new physical for vertex $x^f$ using these barycentric coordinates.
5.1 Mesh folding induced by interpolation

It is possible that after interpolating a fine grid from a coarser grid, the fine grid is folded (i.e. some cell volumes are negative) in some places. Here we give an example to illustrate this possibility.

![Figure 4: An example for mesh folding induced by interpolation. The dashed lines are the fine grid and the solid lines the coarse grid. The initial grid is shown on top. In the bottom grid, the bottom right corner has moved up, and the fine grid has moved with it, consequently being folded at center coarse grid vertex.](image)

Figure 4 illustrates this case. The top grid shows the initial configuration, with dashed lines representing the fine grid and solid lines representing the coarse grid. In the bottom grid the bottom right corner of the initial grid has moved up. With it the fine grid has moved, and consequently one fine grid triangle is folded in the area of the center coarse grid vertex.

This grid folding can be fixed using the mesh untangling procedure introduced in [8]. In this procedure, a three stage approach is taken. In the first stage, the grid is untangled using the feasible set method. The feasible set for a vertex is defined as the set of all positions of the vertex for which each connected element is valid at the corners affected by the position of the vertex (see [8, chapter 2]). The feasible set can be interpreted as a set of inequality constraints, given by the the feasible half planes. See Figure 5 for an example of the feasible area of a vertex. Here the feasible area is the shaded area. In practice, the feasible set can be computed by using the simplex method to minimize simple linear functions (e.g. \( f(x, y) = 1, -1, y, -y \)) to find three distinct corners of the feasible set polygon. The vertex is then placed at the center of this triangle that is formed by these three corners.

The feasible set for a vertex can be empty. In this case the feasible set method fails. In a second stage a functional is minimized. Denote by \( \alpha(\tau) \) the area of element \( \tau \), and let \( \beta > 0 \) be a small parameter. The functional

\[
f(x) = \sum_{\tau \in G} \left( |\alpha(\tau) - \beta| - (\alpha(\tau) - \beta) \right)^2
\]

is minimized using standard minimization methods such as the conjugate gradient method [2, 9]. Note that functional \( f \) has only contributions from elements that have negative area. All other elements contribute zero. It is also important to observe that \( f \) is smooth and convex. In a third step, another sweep of the feasible set approach is applied.

The untangling method outlined above is not guaranteed to work. However, the cases of tangled meshes that arise in the multilevel procedure are well suited to this untangling procedure. Typically, only few cells become tangled in the interpolation procedure. In most cases one sweep of the feasible set approach is enough to untangle the grid.

6. DEFINING A COARSE GRID OPTIMIZATION PROBLEM

In the previous sections we have defined a restriction and, derived from it, an interpolation operator. Both operate on triangular grids. Our goal is to accelerate the original optimization problem, by solving a similar problem on a coarser mesh.
We refer to this procedure as a V-cycle. This V-cycle can be formulated recursively. We call the following procedure \( V(k, \nu_1, \nu_2) \) the iteration on level \( k \):

1. \( \nu_1 \) iterations of Gauss-Seidel Newton on level \( k \).
2. Coarsen the grid on level \( k \) to level \( k - 1 \).
3. If \( k > 2 \), call \( V(k - 1, \nu_1, \nu_2) \), else solve the minimization problem on level 1 to a high level of accuracy.
4. Interpolate the grid from level \( k - 1 \) to level \( k \).
5. \( \nu_2 \) iterations of Gauss-Seidel Newton on level \( k \).

8. COMPLEXITY ANALYSIS OF THE V-CYCLE MINIMIZATION

We first consider the coarsening procedure. The selection of coarse vertices on level \( k \) with \( N_k \) fine vertices takes \( O(N_k) \) steps. The number of selected coarse vertices depends on the connectivity of the fine grid. On a regular grid, the number of coarse points will be \( N_{k-1} \approx N_k/4 \), if the fine vertices are numbered in a lexicographic fashion. See figure 6 for an example. For an unstructured grid, it is difficult to give a good estimate of the ratio of the number of fine and coarse vertices after coarsening. However, the number of coarse vertices depends on the average degree of a fine vertex.

In our coarsening algorithm, coarse grids are generated using a Delaunay triangulation. The expected maximum degree of a vertex in a Delaunay triangulation is \( \Theta(\log n/\log\log n) \), where \( n \) is the number of vertices [10]. In practice, the coarsening ratio is close to 1/4.

The work per V-cycle is relative to \( N_{tot} \), the total number of vertices on all levels combined. Assuming...
a coarsening factor of $0 < \delta < 1$, we get

$$N_{\text{total}} \approx N_K + N_{K-1} + \ldots + N_1 \quad (15)$$

$$= N_K \sum_{i=1}^{K} \delta^{K-i} \quad (16)$$

$$\leq \frac{N_K}{1 - \delta} \quad (17)$$

Here, $N_K$ is the number of vertices in the finest level grid. For a coarsening factor of $\delta = 1/4$, this means $N_{\text{total}} \approx 4N_K/3$.

It is difficult to estimate the complexity of the Newton step that is performed inside the Gauss-Seidel loop. It is however reasonable to assume that this complexity is similar on all levels.

The complexity of the interpolation is $O(N_k)$ on level $k$, if no mesh folding occurs. So, in this case, over all levels, it is $O(N_{\text{total}})$. In practice, mesh folding only occurs in few locations on the fine grid, and can be very efficiently remedied with the untangling procedure outlined above. The computational effort spent in the untangling procedure is negligible.

The coarsening procedure, after the coarse grids have been created in an initial step, is of complexity $O(N_{\text{total}})$. The creation of coarse grids consists of the selection of coarse vertices, which is of complexity $O(N_k)$, on level $k$, and a Delaunay triangulation, which is of complexity $O(N_k \log N_k)$ on level $k$.

In conclusion the initial coarse grid creation is of complexity $O(N_K \log N_K)$. After that, each V-cycle is of complexity $O(N_K)$. Hence, the complexity of a V-cycle is the same as the complexity of a Gauss-Seidel Newton iteration.

## 9. A NUMERICAL EXAMPLE

We present a numerical example that is representative of the performance gain that can be achieved with our multilevel optimization procedure. The code is written in the C programming language.

We start with a regular triangular grid $G$ of with $20 \times 20$ vertices on the domain $(0, 1) \times (0, 1)$. The objective is to find the grid that minimizes functional

$$F(G) = \sum_{\tau \in G} \| f - \Pi_{V^\tau} \|_{L^2(\tau)} \quad (18)$$

with

$$f(x, y) = 20 \tanh(x - y). \quad (19)$$

This function has a very steep gradient along the line $x = y$, and is essentially flat elsewhere. An optimal grid will have most grid points clustered along this line.

![Image](image.png)

**Figure 7:** Fine grid after 500 iterations of Gauss-Seidel steepest descent (top), and 3 V-cycle iterations (bottom).

First, we used a Gauss-Seidel steepest descent procedure to minimize the objective functional. The grid that resulted after 500 iterations is depicted in figure 7 (top image).

Second, we used a V-cycle iteration with $\nu_1 = \nu_2 = 3$, i.e. three sweeps of the Gauss-Seidel steepest descent method were employed before restriction and after interpolation. The coarse grid optimization problem was solved using $\nu_{\text{coarse}} = 10$ iterations of a Gauss-Seidel steepest descent procedure. The grid that resulted after three V-cycle iterations is depicted in figure 7 (bottom image). Table 1 shows the value of the objective

<table>
<thead>
<tr>
<th></th>
<th>Gauss-Seidel</th>
<th>V-cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F(G)$</td>
<td>6.44(-5)</td>
<td>4.74(-5)</td>
</tr>
<tr>
<td># iterations</td>
<td>500</td>
<td>1027</td>
</tr>
<tr>
<td>time (sec)</td>
<td>3</td>
<td>25.9</td>
</tr>
</tbody>
</table>

**Table 1:** Comparison of iteration counts and wall clock times for the Gauss-Seidel steepest descent iteration (left column), and the V-cycle iteration (right column).
functional and the time required to reach this value. Note that the same computer code was used for the Gauss-Seidel steepest descent and the relaxation step inside the V-cycle iteration.

The V-cycle iteration converges much faster than the fine grid Gauss-Seidel steepest descent iteration. The number of iterations, as well as the time are much smaller for the multilevel method. Additionally, the V-cycle iteration reaches a minimum after only three V-cycle iterations. The Gauss-Seidel steepest descent iteration was terminated at a maximum iteration count of 500, for the lack of a good stopping criterion.

For a fair comparison between the two approaches, it is not useful to compare numbers of Gauss-Seidel iterations. Note that Gauss-Seidel iterations on coarse levels are less expensive than on fine levels. Assuming a coarsening factor of 1/4, we can estimate the cost of one iteration relative to the cost of one Gauss-Seidel steepest descent iteration at

$$\sum_{i=1}^{k-1} \frac{v_1 + v_2}{4^{i-1}} + \frac{v_{cycle} - v}{4^{k-1}}.$$  

(20)

Thus, in our example, we estimate the cost of one V-cycle iteration equivalent to 6 + 6/4 + 10/16 = 8.75 Gauss-Seidel steepest descent iterations. So, for the total cost of 3 × 8.75 = 26.25 Gauss-Seidel steepest descent iterations the problem is converged.

In the V-cycle iteration, six Gauss-Seidel steepest descent iterations are performed on each level. To estimate the overhead that is induced by coarse levels, we divide the time per V-cycle iteration \(tv_{cycle} = 8.63\text{sec}\) by the number of Gauss-Seidel steepest descent iteration per V-cycle:

$$\frac{tv_{cycle}}{4} = 2.15\text{sec}$$  

(21)

The average time for one iteration of the Gauss-Seidel steepest descent procedure is

$$t_{Gauss-Seidel} = 2.05\text{sec}$$  

(22)

Comparing the two, we observe that the additional work associated with coarser levels in a V-cycle iteration is not very significant. We also note that the Gauss-Seidel steepest descent iterations that are performed as part of a V-cycle tend to require less costly line searches than for the fine level case. This also helps to keep the average time per V-cycle per fine level Gauss-Seidel steepest descent iteration \(tv_{cycle}/4\) small.

10. EXTENSION TO 3D

The algorithm presented in this paper is a 2D algorithm. The coarsening procedure can be extended to 3D using Delaunay tessellations in 3D. However, as mentioned above, a coarsening procedure such as the one described in [6] might be better suited, since Delaunay tessellations in three dimensions can produce slivers. The untangling procedure is also extensible to 3D (see [8]). This indicates that an extension of our algorithm to 3D is possible. We are planning to pursue this direction in our future research.

11. CONCLUSIONS

We have introduced a new multilevel-type optimization procedure that is well suited to very efficiently solve optimization problems that commonly occur in grid generation applications. Our complexity analysis indicates that our algorithm scales linearly with the number of unknowns. Our research has been focused on the two dimensional case, however, all components of the algorithm are available in three dimensions, as well. We will investigate the three dimensional case in a future paper. It is essential for this approach that a coarse grid representation of the objective function that is to be minimized can be derived. In other words, the change in the initial grid that is required to obtain the optimal grid must be expressible as small changes of vertex positions relative to the positions of neighbor vertices, plus larger changes of positions of groups of vertices. This is possible for applications where the objective is to find a grid that is optimal for the approximation of a function. We plan to extend this work to include the case where an error estimate, and not the actual error, is to be minimized.

References


COMBINED LAPLACIAN AND OPTIMIZATION-BASED SMOOTHING FOR QUADRATIC MIXED SURFACE MESHES

Zhijian Chen, Joseph R. Tristano, Wa Kwok

Ansys, inc., 275 Technology Drive, Canonsburg, PA 15317, U.S.A.
{james.chen,joe.tristano,wa.kwok}@ansys.com

ABSTRACT

Quadratic elements place stringent requirements on a surface mesh smoother. One of the biggest challenges is that a good linear element may become invalid when mid-side nodes are introduced. To help alleviate this problem, a new objective function for optimization-based smoothing is proposed for triangular and quadrilateral elements, linear or quadratic. Unlike the current popular approaches, this objective function makes it possible for a smoothing algorithm to untangle and smooth in a single process. This objective function has higher order continuous derivatives and only one minimum, if any, that make it suitable for optimization techniques. Even though optimization-based smoothing obtains much higher quality results compared to other algorithms, such as constrained Laplacian smoothing, it is also slower than these algorithms. That said, we also present an effective way to limit the number of calls to optimization-based smoothing such that the highest quality mesh is obtained in the least amount of time.

Keywords: optimization-based smoothing, constrained-Laplacian smoothing, smoothing objective function, surface mesh

1. INTRODUCTION

Mesh quality is a key factor in FEM analysis. There are numerous ways to achieve a high quality mesh [1], such as controlling the discretization size, controlling the edge valence of mesh nodes and controlling the distortion of the individual element shapes. Mesh smoothing (relaxation) [2], improves quality by adjusting node locations to reduce the distortion of the element shapes without changing the topology of the mesh. In general, mesh smoothing can be classified into two major groups [3]: local and global. In local smoothing, nodes are moved one by one, while global smoothing changes all the nodal locations in a mesh simultaneously.

The most commonly used smoothing technique is Laplacian smoothing [4], which moves a given node to the geometric center of its incident nodes. Various weighted Laplacian smoothing algorithms have been developed to improve the performance of the original smoothing technique. Laplacian smoothing is computationally inexpensive but does not guarantee improvement in mesh quality. In fact, it is possible to create inverted or invalid elements with this technique. A valid mesh is one whose elements have acceptable quality metrics [2]. Constrained Laplacian smoothing [5] overcomes this problem by placing a node at a new location only when the mesh quality is improved. This method successfully prevents the degradation of mesh quality but does not always improve the quality of the mesh or place nodes at their best locations.

In recent years, optimization-based smoothing algorithms have been drawing the attention of the mesh generation community. Several optimization-based smoothing algorithms have been developed [2,3,4]. These algorithms integrate some mesh quality measures into objective functions. Optimization techniques should, in general yield a better mesh, if the objective function is properly formulated. Optimization-based smoothing varies based on: the type of mesh being smoothed, the optimization method used, and the distortion metric selected to construct the objective function.

One of the keys to the success of an optimization-based smoothing algorithm is to define an appropriate objective function. An inappropriate objective function can waste time in the optimization algorithm along with causing the algorithm to fail to improve the mesh quality. Most efficient optimization algorithms [6] require the objective function be $C^1$ continuous.

Various measures for element [7] quality have been used in the objective function, such as distortion metrics, aspect ratio, minimum angle, etc. Recently, the inspiring work of P. Knupp derived an objective function from the condition number of element Jacobian matrix [8]. His work along with the work of L Freitag [3,10], has lead to mesh quality improvement algorithms for 2D and 3D linear elements. S. Paoletti [9] stated that using Interpolation Tensor could be applied to various polyhedral meshes in 2D and 3D. Even though the published works show enormous potential, there are two general limitations in these algorithms:

1. They only apply to linear elements.
2. They require that the initial mesh is valid.

Quite often, the mesh to be smoothed is not valid. Most of the existing objective functions have been designed in such a way that the optimization smoothing schemes mentioned above cannot guarantee a converged solution for an invalid mesh. For this reason, untangling techniques[10] have been proposed to remove invalid elements from the mesh before executing optimization-based smoothing.
Canann [2] combined the use of Laplacian and optimization-based smoothing to speed up the smoothing process along with benefit of better mesh quality from optimization smoothing. For optimization-based smoothing, \( \alpha \) for a triangle, as defined by S.H. Lo [11], and \( \beta \) for a quadrilateral[2], are used in the objective function. Based on our experience, reasonably good meshes have been achieved in most cases. However, there are some cases, especially when smoothing nodes near curved boundaries, or smoothing nodes attached to higher order elements, in which the resultant mesh quality around these elements are not always satisfactory (Figure 1). As a modification to Canann’s work, we recently improved our smoother to make it suitable for working with quadratic elements by developing a new objective function to be used for optimization-based smoothing [12].

### 2. OBJECTIVE FUNCTION FOR SMOOTHING

A distortion metric is a measure of a mesh’s quality. Therefore, an objective function for smoothing is usually constructed based on some distortion metric or combination thereof. A metric is suitable for use in an objective function, if the following criteria are met:

1. **Efficiency:** Since optimization-based smoothing is computationally expensive, the metric used must be efficient to compute.
2. **Continuity:** Since derivatives are used during the optimization process, the objective function is expected to be continuous. In general, to have higher rate of convergence, higher order derivatives are used.
3. **Monotonically Decreasing:** If an objective function has multiple local optimum locations, it will be difficult for the optimization algorithm to find the best solution. If the metric is not monotonically decreasing, the optimized location may vary based on the initial location of the node to be smoothed.
4. **Shape Independence:** It is favorable for the metric to be defined and normalized in such a way that the metrics for all element shapes can work together

Historically, we at Ansys have used two shape metrics: \( \alpha \) for triangular elements and \( \beta \) for quadrilateral elements. Let us now discuss their properties.

#### 2.1 \( \alpha \) for triangular elements

The triangular metric, \( \alpha \), [11] is defined as,

\[
\alpha = \pm \frac{2 \sqrt{3} \cdot |BC \times AC|}{l_{AB}^2 + l_{BC}^2 + l_{CA}^2},
\]

where, \( l_{AB} \), \( l_{BC} \), and \( l_{CA} \) are the edge lengths of the triangle \( \Delta ABC \), and \( BC \) and \( AC \) are the edge vectors of the triangle, as shown in Figure 2. The metric is signed to account for a positive valid metric and a negative invalid or inverted element. For a linear triangular element, the numerator of equation (1) is directly related to the area of the triangle, while the denominator is the sum of the squared edge lengths of the triangle. The shape metric \( \alpha \) is bounded by \( \alpha \in [-1, 1] \). A value 1 corresponds to the best triangle, an equilateral triangle, while \( -1 \) indicates an inverted equilateral triangle. When all three points of the triangle are co-linear, the triangle has a zero area which yields a value of \( \alpha = 0 \).
2.3 Problems with higher order elements

It is quite often observed that a linear element is of acceptable quality but becomes unacceptable when it is converted to a higher order element. As demonstrated in Figure 4: The elements in (a) and (c) are of acceptable quality when the elements are linear. However, with the introduction of a mid-side node, the quadratic elements in (b) and (d) show interior angles close to 0 and 180 degrees.

2.2 \( \beta \) For quadrilateral elements

Similarly, we have used a quality metric \( \beta \) [2] (Figure 3) for quadrilaterals, which is comprised of a combination of the \( \alpha \)'s of the triangular elements that compose the given quadrilateral. The basic idea is to split a quadrilateral into four different triangles, \( \triangle abc, \triangle dac, \triangle abd, \) and \( \triangle dbc, \) Figure 8 (a). Each of these triangles has a quality metric, \( \alpha_1, \alpha_2, \alpha_3, \) and \( \alpha_4. \)

\[
\beta = \min(\alpha_1, \alpha_2, \alpha_3, \alpha_4) - n_{\text{neg}} \quad (2)
\]

where, \( \alpha_{90} \) is the \( \alpha \) right triangle having unit base and height lengths, and \( n_{\text{neg}} \) is the number of negative \( \alpha \). \( \alpha_{90} \) is used as a normalization factor where \( n_{\text{neg}} \) is a historical heuristic value that was placed into our code many years ago. \( n_{\text{neg}} \) was used to help the algorithm, published in [2], untangle tangled meshes such that inverted elements would have a very high weight in the objective function.

Since the \( \beta \) for quadrilaterals is derived from \( \alpha \) for triangle, the \( \beta \) has similar problems to \( \alpha \) (Figure 3).
where, \( f(\alpha) \) is based on the shape metric \( \alpha \) computed at the node of interest and \( f(\theta) \) is a penalty term based on element angles computed at the other two nodes. The purpose of the \( f(\theta) \) term is to prevent the element from inverting and smooth the element when it is quadratic.

\[
f(\alpha) = \sum_{i=1}^{n_e} f_i(\alpha)
\]

(4)

where, \( n_e \) is the number of element using the smoothing node, and \( f_i(\alpha) \) is the contribution from the \( i \)th element to the objective function.

\[
f_i(\alpha) = (1 - \alpha_i)^2
\]

(5)

Where, \( \alpha_i \) is the shape metric \( \alpha \) of the \( i \)th element. Since the function \( f_i(\alpha) \) is derived from \( \alpha \), the properties and problems of \( f_i(\alpha) \) are similar to \( \alpha \).

A penalty term, \( f(\theta) \), was introduced to give our objective function a monotonically decreasing property with one minima, which we have observed via empirical data. It is because of these penalty terms, that the optimization-smoothing algorithm is able to untangle invalid meshes. Interior angles, at node A and node B, are convenient means to determine the validity of a triangle. When smoothing quadratic elements, the quadratic edge tangent vectors at nodes A and B are used to compute the angles at A and B.

However, angle A and angle B in Figure 5 are not continuous functions on X-Y plane, Figure 6(a). There is a discontinuity on the line, \( \{x = 0\} \), where the angle jumps between \( -\pi \) and \( \pi \). When point C is on the positive x side of the plane, the angle is \( \pi \). When the point is at the negative x side of the plane, the angle is \( -\pi \). Therefore, the angle on the line of \( \{y > 1, x = 0\} \) is undefined.

Figure 5. triangle with node C as a moving node

For a node

\[
f(\alpha) = \sum_{i=1}^{n_e} f_i(\alpha)
\]

(4)

\[
f_i(\alpha) = (1 - \alpha_i)^2
\]

(5)

The adjusted angle is continuous everywhere but at point A and point B, Figure 7 (c). Our penalty term is introduced as.

\[
f(\theta) = k_1 e^{-\frac{\theta}{k_2}} \sqrt{\frac{a}{c}} - 1
\]

(8)

Where \( a \) is the edge length opposite the node where \( \theta \) is computed and \( c \) is the edge length opposite where \( \alpha \) computed. This smooth penalty term increases rapidly when an element angle approaches “0” and the negative angle region. When an element is in the valid region, the penalty is relatively small compared to the near invalid and invalid regions. The constants, \( k_1 \) and \( k_2 \) are used normalize the metric such that \( f(\theta) \) will have little to no effect for an equilateral triangle.

As stated previously, discontinuous functions are not suitable for optimization. To overcome the problem a weight, \( W \), is introduced as a multiplier to the angles to generate an adjusted angle, \( \omega \).

\[
w = 1 - e^{-2(\pi - |\theta|)}
\]

(6)

\[
\omega = w \theta
\]

(7)

Figure 6. angle is not a continuous function in (x,y) plane. (a) Angle distribution over x-y plane. The angle has a discontinuity along the line (x=0, y>1); (b) weight function introduced to make adjusted angle smooth; (c) the adjusted angle. Adjusted angle is continuous and smooth everywhere except at point A and B.
The effectiveness of the above formulation can be demonstrated by the example in Figure 9. As clearly indicated on (a), when the top edge is pushed toward the center node, the best location for the center node is below the intersection of the two dash lines. The other two illustrations show similar results for quadrilateral elements.

For quadrilateral elements, the objective function, $f_q$, is formulated as

$$f_q = f_1 + f_2 + f_3 \quad (9)$$

where, $f_1$, $f_2$, and $f_3$ are the objective functions from triangles $\Delta ADC$, $\Delta ADB$, and $\Delta BDC$, as shown in Figure 8, and the node $D$ is the moving node. Notice that $\Delta ABC$ has been omitted from this function since the movement of point $D$ has no effect on this triangle. The triangle is a dead zone to node $D$.

The issue of how to combine the nodal metrics from a mixed mesh arises when triangular and quadrilateral elements are present. This resolution of this issue is actually quite simple. As shown in equation (9), the objective function for a quadrilateral is made up of three functions using sub-triangles. Each of these sub-triangles is equivalent to equation (5). Therefore, for a mixed mesh, the contribution of each triangular element to the total objective function should be multiplied by a factor of three in order to evenly weigh triangles and quadrilaterals.

An example configuration similar to the one presented in [3] is used to show the result of our current work. Figure 10 illustrates that even though the level sets outside of the valid region for the center node is non-convex, the near convexity of the set enables faster convergence of the optimization algorithm.

As discussed above, each part of the objective function is continuous function with at least up to $2^{nd}$ order derivatives. This property makes the objective function very suitable for an optimization process. Derivatives such as these enable us to use gradient methods such as the method of conjugate gradients.

### 3 OPTIMIZATION-BASED SMOOTHING

Optimization-based smoothing is an iterative process. Each node for smoothing is optimized for location in a number of iterations. Generally, as in popular approaches, the optimization process is a constrained optimization process. However, the presented objective function actually combines angle constraints into part of the objective function, which enables the optimization-based smoothing process to be unconstrained.

Let $X$ be nodal location of a node, the optimization process is to find the best location in iterations:

$$X^q = X^{(q-1)} + sd^q \quad (10)$$

where, $q$ is the iteration number, $d^q$ is the vector of the search direction and $s$ is the step length to move in this search.
direction. Optimization is a classical area of study in mathematics and it is not our intent to discuss it in depth here. However, it is valuable to share some of our experiences with it when related to mesh smoothing.

### 3.1 Search direction

Since the objective function is smooth, the Fletcher-Reeves conjugate gradient method is used.

\[
d^q = -\nabla f(x^q) + r^q d^{(q-1)}
\]  

(11)

where,

\[
r^q = \frac{|\nabla f(x^q)|^2}{|\nabla f(x^{(q-1)})|^2}
\]  

(12)

In our implementation, the gradient direction is used at the first iteration and conjugate gradient direction is used in the consequent iterations. We have found that after a number of iterations, the convergence speed using the conjugate gradient direction actually slows down. Figure 11. illustrates, the iteration using gradient direction is marked with a letter “g” and the iteration using conjugate gradient directions are marked with a letter “c”. At the first iteration, when the gradient direction is used, the proceeding step length is relatively small, while at the second iteration, the conjugate gradient direction is used and the step length is much larger than the first step length. However, two more iterations later, the progress becomes relatively small. In this case, a new gradient direction is used and the process restarts.

### 3.2 Step length for one-dimensional search

A one-dimensional search is conducted to find a local minimum along the search direction. Quadratic interpolation is used in our one-dimensional search. Progressively, three points are found in order to compute a “high-low-high” pattern. For faster speed, an acceleration factor is used. The choice of the acceleration factor is very tricky. In the beginning of the search, because the initial step length is relatively small, a large factor, 8, is used. Other accelerating factors might also be used in the consequent searches. Once a failure to find the “high-low-high” pattern is encountered, the factor is reduced by half. The one-dimensional search fails if a “high-low-high” pattern cannot be found.

### 4. OVERALL SMOOTHING ALGORITHM

#### 4.1 The use of optimization-based smoothing

Since optimization-based smoothing is much slower than Laplacian smoothing, it is critical to make a correct decision when to use optimization-based smoothing to gain the best cost effectiveness. Here are our rules:

1. a node is connected to a curved boundary node;
2. a node is connected to an invalid element;
3. a node has failed to move from Laplacian smoothing;
4. forced to use optimization-based smoothing by caller.

According to the rules above, an untangling process, for example, is mostly smoothed by optimization smoothing because a tangled mesh contains many nodes that are connected to invalid elements. Once untangled, most interior nodes are actually smoothed by constrained-Laplacian smoothing.

Constrained-Laplacian smoothing has trouble dealing with concave regions. If a new location, determined, by Laplacian smoothing is not acceptable, optimization-based smoothing is called to resolve the problem.
There are cases when the caller detected that a mesh is not acceptable after smoothing. In such cases, one pass of optimization-based smoothing usually gives us satisfactory results.

### 4.2 Smoothing by priority order

It is found that node smoothing in order of “worst one first” is very helpful. As shown in Figure 13, when priority is used, smoothing takes 4 iterations for the tangled model to be untangled, while 7 passes are needed if nodes are smoothed in a order of “first come first serve”. The priority is simply computed based on the shape metrics of each node. For the node with the worst quality, the highest priority is assigned. The other priorities are computed by linearly dividing the range of shape metric value into 5 bins. The priority is then computed for each smoothing iteration. An inner priority loop counter sets the current priority during a smoothing iteration. If the current node’s priority is less than that of the current priority, it will not be smoothed in the inner priority loop.

### 5. EXAMPLES AND RESULTS

#### 5.1 Smoothing improvement statistics

The improved smoothing algorithm has been fully tested under the ANSYS/Classic and ANSYS/Workbench regression test sets along with many customer problems to verify that it is sufficiently robust and efficient as a commercial product.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Significance (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.00</td>
</tr>
<tr>
<td>Max</td>
<td>0.00</td>
</tr>
<tr>
<td>Avg</td>
<td>0.49</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 3 condenses the information found in Tables 1 and 2 with a paired samples t-test, a statistical test that compares means (the details of the test are outside the scope of this paper. For more information, consult almost any basic statistics textbook), for minimum, maximum, average, and standard deviation of shape metrics. The improvement minimum, average, and standard deviation of the shape metrics are significant while the decrease in the maximum is not statistically significant.

#### 5.2 Examples meshes

In this section, we will present a number of example meshes. Figure 13 is an example for the untangling of the “plate with a hole” model. The tangled mesh (Figure 13a) is created by perturbing the nodal locations of each interior node (a node not on surface boundaries) randomly. The tangled mesh has many nodes outside the surface domain. Figure 13b is the mesh after smoothing. It takes 3 iterations for this model to be untangled (Figure 14 and Figure 15).

Table 1. mesh quality without new smoothing

<table>
<thead>
<tr>
<th>NQUAD</th>
<th>NTRI</th>
<th>MIN</th>
<th>MAX</th>
<th>AVG</th>
<th>STDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td></td>
<td>0.76579</td>
<td>0.02914</td>
<td>0.37137</td>
<td>0.08026</td>
</tr>
<tr>
<td>AVG</td>
<td>133.549</td>
<td>0.457263</td>
<td>0.940831</td>
<td>0.307522</td>
<td>0.689708</td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td>0.32292</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. mesh quality with new smoothing

<table>
<thead>
<tr>
<th>NQUAD</th>
<th>NTRI</th>
<th>MIN</th>
<th>MAX</th>
<th>AVG</th>
<th>STDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN</td>
<td></td>
<td>0.70207</td>
<td>0.06631</td>
<td>0.52954</td>
<td>0.07245</td>
</tr>
<tr>
<td>AVG</td>
<td>137.708</td>
<td>0.415937</td>
<td>0.936968</td>
<td>0.415502</td>
<td>0.7136</td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td>0.32292</td>
<td></td>
</tr>
</tbody>
</table>

The above two tables are statistical results from meshing 183 complex and planar surfaces randomly picked from regression test sets. Some of the surfaces are meshed with just several elements where some are meshed with thousands of elements.

Table 1 is the result before the new smoothing was implemented where Table 2 is the result with new smoothing. A detailed explanation of Table 1 and Table 2 follows:

1. With new smoother, the number of Quadrilateral elements (NQUAD) generated is increased and consequently, the number of Tri elements (NTRI) generated is decreased. The reason the number of elements varies between the two smoothing algorithms is because the smoother is integrated into the mesh generation process. This phenomenon illustrates how different mesh generation algorithms are sensitive to node placement.

2. As indicated in the fourth column (MAX), the best quality elements generated with new smoothing is actually not as good as it used to be, however, this is not statistically significant (sig = .49 > .05).

3. The worst elements have improved substantially, column 5 (MIN).

4. The average, column 6 (AVG), of the element shape metrics has also increased.

5. The last column (STDEV) of the tables illustrates the standard deviation that measures the variance of the samples. It is clearly indicated that, the standard deviations of the minimum and average have decreased in Table 2. However, the max standard deviation remains the same.

Table 3. t-test for shape metrics
Figure 13. untangling example

(a) tangled plate with a hole  (b) untangled plate with a hole

Figure 16. a node with three edges close to a curved boundary and two quadratic boundary edges

Figure 17 illustrates another unit test example. Similar to the above example, a node is connected to some curved quadratic boundary elements. The optimization-based smoothing result (Figure 17a) yields higher quality than the same configuration smoothed with constrained Laplacian smoothing, (Figure 17b).

Figure 14. untangling, initial mesh and first iteration mesh

Figure 15. untangling, second iteration mesh and third iteration mesh

Figure 18 shows an example of high quality mesh generated using the new smoother.
Figure 18. example surface mesh

Figure 19 illustrates a mesh of a human head using the new smoothing algorithm.

Figure 20 illustrates the differences between the old objective (b) function and the new objective function (c) for a given configuration of quadrilaterals.

5.3 **Objective function comparisons**

Our previous method of constructing the objective function—based on the maximizing the minimum shape metric—posed serious convergence problems for any optimization method.
6. CONCLUSIONS AND FUTURE WORK

A new objective function for optimization-based smoothing is proposed for both triangular and quadrilateral elements. Unlike the current popular approaches, the new objective function makes it possible to untangle and smooth in a single process. The objective function has higher order continuous derivatives and only one minimum, if any, that make it quite suitable for optimization techniques. Because optimization-based smoothing is much slower than other algorithms, such as constrained Laplacian smoothing, an effective way to limit the number of calls to optimization-based smoothing is critical in order to obtain the best result in terms of quality and speed.

Future work in this area may include:

- Speed improvement on metric calculation so we can use optimization smoothing more often
- Mathematically prove properties of the objective function
- Extend the algorithm to solid elements

REFERENCES

A LOCAL CELL QUALITY METRIC AND VARIATIONAL GRID SMOOTHING ALGORITHM

L. Branets\textsuperscript{1} and G.F. Carey\textsuperscript{2}

\textsuperscript{1} CFDLAB, Institute of Computational Engineering and Sciences, The University of Texas at Austin, larisa@ices.utexas.edu
\textsuperscript{2} CFDLAB, Institute of Computational Engineering and Sciences, The University of Texas at Austin, carey@cfdlab.ae.utexas.edu

ABSTRACT

A local cell quality metric is introduced as a function of Jacobian matrix and used to construct a variational functional for a grid smoothing algorithm. The properties of the local quality measure which combines element shape and size control metrics are examined and the effectiveness of the algorithm is tested on several representative grids. The approach can be applied to different cell types and dimensions.

Keywords: mesh generation, smoothing, mesh quality measures

1. INTRODUCTION

Generating quality meshes that permit reliable accurate simulations remains a major technical and theoretical problem. The need to solve more complex applications for multi-material domains with irregular geometry and varying spatial scales exacerbates the difficulty. For example, large scale simulations now involve meshes with millions of cells and feature sizes that vary by orders of magnitude. Generated meshes are often mathematically incorrect (e.g. contain folded cells) or unreliable, so generating, remeshing, and improving the mesh have become time consuming but pressing tasks. This adversely impacts our ability to do efficient simulation and design. In simulations where moving boundaries arise, the situation can be much worse since the mesh deteriorates as cells deform degrading accuracy, conditioning, and computational effectiveness (more iterations and shorter timesteps). Periodic remeshing and frequent grid smoothing are needed, as well as other corrective actions.

The present work examines this issue from the standpoint of local cell quality and a novel scalar cell quality indicator is introduced. This indicator is used to construct a global functional for mesh optimization that we use here as a mesh smoothing and correction strategy that is investigated numerically in several test problems.

Algorithms based on such optimization strategies have been utilized previously, one of the earliest studies being that of Winslow [1]. More recent works [2] - [7] develop related ideas. The outline of the present treatment is as follows: we first summarize some basic ideas underlying harmonic maps and Laplace-type grid smoothing in a variational setting. This leads us to introduce a local “distortion measure” and its reciprocal defines the local mesh quality indicator. The global functional is constructed as an accumulation of the element contributions. The optimization descent problem is solved using a damped Newton scheme. Several case studies are provided in the results.

2. MAPPING AND VARIATIONAL STATEMENT

A popular strategy in structured grid generation is to map a uniform Cartesian grid in a reference domain $\Omega$ to a curvilinear grid of well-shaped cells in the physi-
cal domain $\Omega$. The properties of conformal maps can be exploited to ensure a good mesh. The associated properties of analytic functions of a complex variable have led to the use of harmonic functions from complex analysis and, indirectly, to schemes based on solving the associated Laplace equation. Since the potential and stream function solutions to Laplace's equation form a suitable coordinate family for a mesh, this idea has motivated PDE solver strategies for mesh generation. In particular, the Laplace problems

$$\Delta \xi = 0, \quad \Delta \eta = 0 \quad (1)$$

in the physical domain $\Omega$ can be mapped to quasilinear PDEs in the reference domain $\Omega$ and solved for untangled meshes in $\Omega$.

Alternatively, variational formulations for these PDE problems can be easily constructed and an equivalent optimization problem is obtained. For (1), the variational functional is the classical Dirichlet integral for the Laplacian and we have

$$I = \int_{\Omega} \left[ (\nabla \xi)^2 + (\nabla \eta)^2 \right] dx \, dy. \quad (2)$$

The pair of Laplace problems (1) follow as the associated Euler-Lagrange equations. Mapping the variational problems to the reference domain we have

$$I = \int_{\Omega} \frac{\text{tr}(S^T S)}{\det S} d\xi d\eta \quad (3)$$

which can be expressed compactly as

$$I = \int_{\Omega} \frac{\text{tr}(S^T S)}{\det S} d\xi d\eta$$

where $\text{tr}$ denotes the trace and $S$ is the Jacobian matrix of the map between the two frames. From this, following the idea of describing grid quality by metric tensor invariants [8, 5] or functions of Jacobian matrix [9], we define the local distortion measure

$$\beta(S) = \frac{\frac{1}{2} \text{tr}(S^T S)}{\det S} \quad (4)$$

and local mesh quality as $Q_0 = \beta^{-1}$. In fact, this distortion measure generalizes to any dimension directly [10] as

$$\beta(S) = \left( \frac{\frac{1}{2} \text{tr}(S^T S)}{\det S} \right)^{n/2} \quad (5)$$

The size of the cell can be controlled by introducing a second local measure which we term the “dilation” measure. Let $v$ be a “desired” element size measure (related to area in 2D and volume in 3D). For the “desired” uniform grid, $v$ is taken to be the average cell area

$$v = \frac{\int_{\Omega} \text{det} S \, d\xi d\eta}{\int_{\Omega} d\xi d\eta}$$

Then the ratios $\det S / v$ and $v / \det S$ indicate the departure of $\det S$ from $v$. Since $\det S$ can be above or below $v$ we introduce the symmetric dilation measure [7]

$$\mu(S) = \frac{1}{2} \left( \frac{v}{\det S} + \frac{\det S}{v} \right).$$

Then $\mu = 1$ when $v = \det S$ and $\mu \to \infty$ as $\det S \to \infty$ or 0. Following the ideas of multi-objective functions [2, 6], we introduce an additive distortion-dilation measure (other forms are clearly possible)

$$E_\theta = (1 - \theta) \beta(S) + \theta \mu(S), \quad (6)$$

where coefficient $0 \leq \theta < 1$ in linear combination can be adjusted to emphasize the respective distortion and dilation terms.

Now the variational grid smoothing formulation can be stated as follows: minimize the functional

$$I = \int_{\Omega} E_\theta(S) d\xi d\eta, \quad (7)$$

subject to relative boundary (or other) constraints. In order to obtain the discretized problem formulation (7), contributions to the functional from each cell $c$ are approximated using some numerical integration rule as

$$I_h = \sum_{c=1}^{N_c} \sum_{q=1}^{K_q} \sigma_{g(c)} E_\theta (S_{q(c)}), \quad (8)$$

where $\{q(c)\}$ identify the quadrature points and $\sigma_{g(c)}$ are the corresponding quadrature weights for cell $c$.

Remarks: 1. The formulation can be applied to any unstructured grid containing different types of cells, using appropriate quadrature rules; 2. Unstructured grids will have varying nodal valence and this effect will also be investigated in the numerical work of section 6.

3. LOCAL QUALITY MEASURE

Let us first mention some general properties of the local distortion measures (5) and (6) and then focus on their behavior for meshes using the basic triangle and quadrilateral cells since they are most widely used in grid generation.

The function $\beta(S)$, reformulated in terms of invariants of the metric tensor of coordinate transformation $G = S^T S$, was considered in [5]. It was shown that $\beta(S)$ controls the cell angles and cell aspect ratio in the 2D case and has similar properties in 3D. The estimates for the angle $\alpha$ between two cell edges and cell aspect ratio $F$ (ratio of the lengths of the edges) for 2D quadrilateral cells are

$$\sin^2 \alpha \geq (1/\beta)^2, \quad 2 \leq F + 1/F \leq 4\beta^2 - 2.$$
Thus $\beta \to 1$ enforce $\alpha \to \pi/2$ and $F \to 1$; i.e. a square cell.

The modified distortion measure $E_\theta$ retains these properties of $\beta(S)$. It is an indicator for quasi-isometry of the mapping $[7, 10]$ - analog of mapping conformality characterization, in the sense that

$$\gamma^2 I \leq S^T S \leq \Gamma^2 I,$$

where $\gamma$ and $\Gamma$ can be estimated from $E_\theta$.

In the following sections we will examine the local measure $E_\theta(S)$ or corresponding local quality measure $Q_\theta(S) = 1/E_\theta(S)$ ($Q_\theta(S) = 1/\beta(S)$ ) on linear mapped simplex elements and bilinear isoparametric elements in more detail.

### 3.1 Simplex elements

Let us first consider the 2D triangular element, which has been most extensively analyzed in the literature (see [5, 11] for references). Taking the reference element to be the equilateral triangle with sides of length 1 and computing the constant Jacobian matrix of the linear map onto an arbitrary triangular element with area $A$ and edges of lengths $l_1, l_2, l_3$, we get

$$\det S = \frac{4}{\sqrt{3}} A, \quad \text{tr}(S^T S) = \frac{2}{3}(l_1^2 + l_2^2 + l_3^2).$$

Thus

$$\beta = \frac{l_1^2 + l_2^2 + l_3^2}{4\sqrt{3}A}$$

and quality measure $Q_\theta$ is equal to

$$Q_\theta = \frac{4\sqrt{3}A}{l_1^2 + l_2^2 + l_3^2}.$$

This is a well known example [12, 13] of a "fair" geometric measure in the sense that it is equal to 0 on any type of degenerate triangle. It is also normalized (takes values from the interval $[0, 1]$).

The corresponding additive measure from (6) is

$$E_\theta = (1 - \theta) \frac{l_1^2 + l_2^2 + l_3^2}{4\sqrt{3}A} + \theta \left( \frac{\sqrt{3}}{4A} + \frac{4A}{\sqrt{3}} \right).$$

The level sets of the corresponding modified quality measure $Q_\theta = E_\theta^{-1}$ for a triangle with fixed edge $(0, 0) \to (0, 1)$ as a function of the coordinates $(x, y)$ of the opposite vertex are shown in Figure 1 for different values of parameter $\theta$. As $\theta$ increases, the quality measure becomes less restrictive in the sense that it admits more points in the regions $Q_\theta > \text{const}$, but it remains a "fair" measure.

For the mapping of an arbitrary tetrahedron onto the regular tetrahedral reference element with edges of

![Figure 1: Level sets of $Q_\theta(x, y)$ on triangle with vertices (0,0), (0,1), (x,y)](image)

length 1 we have

$$\det S = 6\sqrt{2}V, \quad \text{tr}(S^T S) = \frac{1}{7} \sum_{i=1}^{6} l_i^2,$$

where $V$ is the volume of the tetrahedron and $l_1, \ldots, l_6$ are its edge lengths. Thus for the quality and additive distortion-dilation measures respectively we get

$$Q_\theta = \frac{72\sqrt{3}V}{(\sum_{i=1}^{6} l_i^2)^{3/2}},$$

$$E_\theta = (1 - \theta) \frac{(\sum_{i=1}^{6} l_i^2)^{3/2}}{72\sqrt{3}V} + \theta \left( \frac{1}{6\sqrt{2}V} + 6\sqrt{2}V \right),$$

and $Q_\theta = E_\theta^{-1}$ which are also "fair" measures in the sense given above. A similar tetrahedron shape measure

$$\eta = \frac{12\sqrt{3}}{(\sum_{i=1}^{6} l_i^2)^{2/3}} = \left( Q_\theta \right)^{2/3}$$

was derived in [14] from the singular values of transformation $S$. Geometrically $\eta$ reflects the shape of the inscribed ellipsoid.

### 3.2 Tensor product linear elements

The case of the mapped tensor product linear cell in $n$ dimensions is more complex, since the Jacobian matrix is not constant on the cell. Nevertheless, $E_\theta$ satisfies a “maximum principle” [10], in the sense that it is bounded from above by a finite linear convex combination of its values on certain bases (matrices). Thus an upper bound for the additive measure (lower bound for quality measure) can always be computed. Matrices for this bound are a full set of constant matrices arising from a representation of the Jacobian matrix on the tensor product cell. For example, the bilinear map of unit square $0 \leq \xi_1, \xi_2 \leq 1$ onto the cell with vertices $r_{i,j}$, $j, k = 0, 1$ can be written as

$$r = \sum_{j,k=0}^{1} (1 - \xi_1)(1 - \xi_2) \xi_j^i \xi_k^j r_{i,j} \quad (9)$$
and its Jacobian matrix is

\[ S = \sum_{j,k=0}^{1} (1-\xi_j)^{(1-j)} \xi_j^k (1-\xi_k)^{(1-k)} \xi_k^j (r_{1k} - r_{0k}, r_{j1} - r_{j0}) \]

\[ = \sum_{j,k=0}^{1} c_{jk} \hat{S}_{jk}, \quad \text{where} \quad \sum_{j,k=0}^{1} c_{jk} = 1, \]

where \( c_{jk} = (1-\xi_j)^{(1-j)} \xi_j^k (1-\xi_k)^{(1-k)} \xi_k^j \) are the scalar coefficients in (10) and the corresponding pairs of vectors are the columns of matrices \( \hat{S}_{jk} = (r_{1k} - r_{0k}, r_{j1} - r_{j0}). \)

Thus for the bilinear cell, the bound is 1/4 of the sum of additive measures computed at vertices of the quadrilateral cell. That is

\[ E_\theta \leq \sum_{j,k=0}^{1} \frac{1}{4} E_\theta(\hat{S}_{jk}). \]

For a trilinear cell this type of representation of the Jacobian matrix contains 64 different constant matrices. They can be obtained from trilinear images of the basis triples in reference space. All 64 such basis triples can be obtained from the four distinct vectors shown in Figure 2 by rotation and reflection (after reflection the orientation should be changed to preserve right basis).

The level sets for the lower bound of the quality measure \( Q_\theta \) for the bilinear cell are shown in Figure 3, where quality contours are graphed as functions of the position \((x, y)\) of one vertex of the quadrilateral with the other vertices fixed at points \((0, 0), (0, 1)\) and \((1, 0)\).

The existence of the upper bound on the local additive measure \( E_\theta \) implies that in order to control cell quality it is sufficient to control the bound; that is, the values of the additive measure on a finite number of combinations of cell vertex basis vectors. Thus the choice of these combinations as quadrature points for approximating the discrete functional (8) will guarantee the improvement in mesh quality as the result of solving minimization problem (7).

4. NUMERICAL IMPLEMENTATION

The gradient of the smoothing functional (8) is nonlinear so an iterative optimization algorithm, such as Newton's method or another gradient descent method, should be applied to the associated algebraic problem. In this work, the damped Newton method is used. After each iteration the global minimum quality measure

\[ \left( Q_\theta \right)_{\text{min}} = \min_q \frac{1}{E_\theta(S_{\theta}(c))} \]

is computed in order to monitor the optimization process. Iterations cease when the difference between the minimum quality (11) of two subsequent grids is less than a given tolerance (other criteria are possible).

5. MODIFICATIONS OF THE METHOD

If the initial grid is folded or has nonconvex cells, the functional can be modified by adding penalty terms (see, for example, [6]). The original functional (8) has an infinite barrier on the set of grids with convex cells which is due to the presence of \( \text{det } S \) in the denominator of the integrand. A penalty formulation can be developed by replacing this factor \( \text{det } S \) by an exterior penalty function \( \chi_\delta(\text{det } S) \), such that the new integrand will be a finite approximation of the original infinite barrier. This modification allows the minimization procedure to start from a folded grid and, since the value of the functional \( I_k \) is significantly increased when folded cells are present in the grid, the final grid will not contain nonconvex cells (assuming there exists such a mesh solution for the given connectivity and boundary conditions).

Since the distortion measure \( \beta(S) \) provides control over element shape, one can define a priori the desired element shape by introducing a metric in reference coordinates. These metrics essentially use different reference elements for different cells in the grid. Minimization of the correspondingly modified functional will result in a grid with cells having the shape as close as possible (under given connectivity of the grid and imposed boundary conditions) to the target shapes.
6. NUMERICAL EXAMPLES

6.1 Smoothing of a triangular grid in non-convex domain

The Laplace smoother may produce overlapping grids in nonconvex domains, so it is important to check the behavior of the present type of smoother for such domains. Consider the nonconvex (v-shaped) domain with triangular grid and fixed boundary nodes shown on the left in Figure 4. Smoothing with the presented additive functional using $\theta = 0.8$ produces the grid on the right in Figure 4. There is no overlap and the mesh lines are well behaved. Cells at the peak on the symmetry line are slightly dilated and those at the reentrant corner are slightly compressed.

6.2 2D meshes with points of changing valence

The following numerical test demonstrates the advantages of the described smoothing algorithm, when operating on a grid with varying valence. Some algorithms will produce significant local dilation effects in the regions where valence changes [15].

Figure 5 demonstrates the smoother behavior on triangular grids with changing valence. All boundary nodes are fixed in this example. There is some disparity in dilation effects but the behavior is satisfactory.

The effect of smoothing on a mesh of quadrilateral cells is shown in Figure 6. The initial grid consists of two block-generated subgrids corresponding to a trapezoidal subdomain and its continuation to the annular region. Boundary nodes on the exterior circular boundary are fixed and nodes on the vertical diameter boundary of the semicircle are allowed to “slide” along this line. The initial mesh and the “evolving” mesh at iterations 1, 2, 3 are shown.

6.3 Mesh unfolding in nonconvex domain

Barrier formulations of variational smoothing algorithms facilitate mesh unfolding as well as smoothing.
As an example let us consider the unfolding of a folded quadrilateral mesh for an annular cylindrical domain. For the initial grid we relocate the nodes interior to a cylindrical polar mesh for a semicircular annulus and place them at the origin as indicated by the mesh on the left in Figure 7. After applying the smoothing algorithm for 5 iterations, the grid is close to equidistributed as seen on the right in Figure 7.

6.4 Initially adaptive quadrilateral grid with folded cells

In the next example the smoothing procedure is applied to a more elaborate grid, generated to adaptively fit a multi-airfoil domain. This grid has many nodes with irregular valence and it initially had several folded cells. The most relevant part of this grid before and after smoothing is shown in Figure 8. This example indicates the importance of cell size control (via $\mu(S)$), since without it the smoothing procedure "undoes" desired clustering near the airfoils and tends to promote a uniform grid, which is unacceptable because boundary layers need to be resolved. Initially, the volumetric factors $v$ were computed for each cell, and then the smoothing algorithm was run using these values. The improvement in the grid details can be seen in Figure 9.

It can also be observed from Figure 9 that smoothing may not retain enough clustering in the boundary layers. Thus, in order to have a grid that retains the initial mesh density in the boundary layer, a block smoothing strategy may be utilized.
7. CONCLUDING REMARKS

The variational smoothing algorithm developed here yields satisfactory results for triangular and quadrilateral meshes. In particular, it handles several of the difficulties that have been troublesome for other smoothers. It is applicable to general element types and hybrid grids as well as 3D (the 3D case is currently being tested). We have also carried out tests on higher resolution grids and explored the effect of varying $\theta$. Other issues related to the effect of different local valence are also being investigated.

Acknowledgement: This work has been supported in part by LANL grant number 62485-001-02.

References


[10] Branets L.V., Garanzha V.A. “Global Condition Number of Trilinear Mapping. Application to 3D


INCREASING TAU3P ABORT-TIME VIA MESH QUALITY IMPROVEMENT

Nate Folwell\textsuperscript{1*} Patrick Knupp\textsuperscript{2*} Michael Brewer\textsuperscript{2}

\textsuperscript{1}Stanford Linear Accelerator Center, Stanford CA U.S.A. folwell@slac.stanford.edu
\textsuperscript{2}Sandia National Laboratories, Albuquerque, NM U.S.A \{mbrewer,pknupp\}@sandia.gov

ABSTRACT

The Tau3P time-domain computational electromagnetics code is used at The Stanford Linear Accelerator Center for a wide variety of accelerator design tasks. The code uses the Discrete Surface Integral method to solve Maxwell’s equations on prismatic and dual hexahedral meshes. In this method run times are highly sensitive to mesh quality, particularly to minimum edge-length, element skew, and lack of smoothness. In this study we investigate methods for increasing Tau3P run times via mesh quality optimization. It is found that abort-time can be significantly increased via optimization of the mesh condition number or by Laplacian smoothing.

Keywords: mesh quality, mesh optimization, mesh smoothing, computational electromagnetics

1. INTRODUCTION

The Stanford Linear Accelerator Center (SLAC) is designing particle accelerators, in part, by performing computer simulations using a code called Tau3P to solve Maxwell’s Equations in the time domain. The discretization algorithm used in Tau3P is the Discrete Surface Integral (DSI) scheme [1]. The algorithm has been proven to be conditionally stable for strictly orthogonal structured grids. The particle accelerator hardware consists of waveguides and other parts forming complex 3D geometrical configurations that are impossible to mesh with strictly orthogonal structured grids. Presently SLAC is using the CUBIT mesh generation code [2] to create unstructured hexahedral and tetrahedral meshes which accurately represent the geometry but may lack the high quality of orthogonal mapped meshes. No DSI stability condition has been proven for these more general meshes. Empirically, some simulations using these meshes exhibit unstable behavior, suggesting that the DSI scheme is not even conditionally stable for the unstructured grids.

To work around this issue, a filtering technique [3] is used in Tau3P which permits useful calculations to be performed by delaying the onset of instability. With this filter, most Tau3P calculations are able to complete to the desired problem end-time. In some runs, however, the numerical solution grows exponentially, swamping the problem before the desired end-time is reached. To avoid computing meaningless solutions, Tau3P has a mechanism by which runs can be aborted before the end-time is reached. A comparison is made between the magnitude of the electric field vector at selected points and a value determined automatically given the problem. If the electric field vector exceeds the supplied value then the results are most likely non-physical, due to the unstable nature of the discretization algorithm. If this situation occurs, the run is aborted. We define the abort-time of a run to be the time (if any) at which the run is aborted.

It has long been observed that the abort-time encountered in many Tau3P simulations is sensitive to properties of the mesh. Much time has been spent tinkering with and regenerating meshes so that a given

\textsuperscript{*}The first authors’ work was funded by the DOE SciDAC Accelerator Project

\textsuperscript{1}The second and third authors’ work was funded by the DOE SciDAC-TSTT Center under contract DE-AC04-94AL85000
simulation will not abort before the desired problem end-time is reached. To reduce the time needed to create an acceptable mesh, it would be helpful to know precisely which mesh properties play a crucial role in determining abort-time. An empirical investigation to identify the strongest correlations between the abort-time and various mesh properties [4] was conducted. Having identified the crucial mesh properties, we seek in this study to improve mesh quality with respect to the crucial properties, resulting in calculations that have a higher probability of running to completion.

2. MESH QUALITY METRICS

In the previous report [4], a series of meshes was studied to determine which quality metrics are most important in affecting abort-time. The meshes were created on a pillbox geometry (see Figure 1) that is representative of a simplified accelerator structure. Four cases (PBU, PBF, PPU, and PPF) were considered, resulting from combinations of beam/pulse excitation and unfiltered/filtered. To limit the number of cases to a manageable number, the present study focuses on just one of the four cases. The PPU case was selected because results are easiest to interpret from a physical viewpoint. The study showed that the most important metrics were:

- Minimum primal edge size (MPES).\(^1\) This is the minimum edge length in the mesh, including boundary edges. In general, this metric can range from zero to infinity. For the pillbox geometry, the approximate acceptable range of this metric is MPES greater than 0.00040 meters (edge-lengths smaller than that tend to result in relatively small abort-times).

- Maximum primal condition number (MPCN). This is the maximum quadrilateral condition number in the mesh. The quadrilateral condition number is defined as the maximum condition number at any of the four vertices of the quadrilateral. The condition number at a vertex is the sum of the squares of the two adjacent edge lengths, divided by twice the area. This metric ranges from one to infinity. The approximate acceptable range is between 1.0 and 2.0, with 1.0 being ideal. MPCN larger than 2.0 tends to cause relatively small abort-times.

\(^1\)MPES is not a scaled metric, but an absolute number. Because the comparisons we make are always on the same model, scaling is not an issue for the results presented in this paper. We recognize that it could be a problem if we were to try to make comparisons across geometric models. We have devised a number of scaled metrics to supplant MPES, but our studies to date have not found them as useful.

\[
\begin{align*}
    s_{j,k} &= \left| \frac{A_k - A_j}{A_k + A_j} \right| \\
    t_k &= \max_j s_{j,k} \\
    MP\text{SM} &= \max_k (t_k) / \bar{t}
\end{align*}
\]

where \( \bar{t} \) is the average value of \( t_k \) over the entire mesh. This metric is effective in detecting jumps in area between adjacent quadrilaterals. If element \( k \) is on the boundary, it is not included in the calculation. This metric is only meaningful provided the elements areas are all positive. This metric ranges from zero to infinity, with an approximate acceptable range of zero and 5.0. When MP\text{SM} exceeds 5.0, abort-times tend to be relatively small.

- Maximum primal scaled Jacobian (MPSJ). This is the maximum sine of the angles in the mesh. The metric ranges from -1.0 to +1.0, with an approximate acceptable range of 0.40 to 1.0. When MPSJ is less than 0.40, abort-times tend to be relatively small.

These metrics reflect some of the basic properties of geometric mesh quality: size, smoothness, angle, and aspect ratio. Many other metrics which measure these generic properties are also correlated with abort-time and thus could have been used in this study.

The previous study [4] established the importance of these metrics by generating a set of 25 pillbox
meshes and determining an abort-time for each of the meshes. For each of the 25 meshes one may calculate various mesh quality metric values. Each metric value can, in turn, be paired with the abort-time for the given mesh to create a scatter plot. If a given metric is a poor predictor of abort-time, the scatter plot will show no pattern. On the other hand, a strong predictor of abort-time will show a distinctive 'L' pattern. Figure 2 shows the scatter plot obtained for the Maximum Primal Condition Number metric. This metric exhibits the 'L' pattern: if the mesh has good quality (small condition number values) then the abort-time is independent of the metric (horizontal branch of the 'L'). In that case, the abort-time is determined in conjunction with other mesh quality metrics. If the mesh has poor quality (large condition numbers), the points in the scatter plot lie on the vertical branch of the 'L'. This indicates that the abort-time is relatively small and that this is likely due to mesh elements having poor shape quality (large condition number). If a metric exhibits a strong 'L' pattern in its scatter plot, one can define an acceptable range for the metric, namely, the set of values for which abort-time is independent of the metric. In the case of the MPCN scatter plot, for example, the acceptable range is approximately between 1 and 2.

The present situation for Tau3P meshing is iterative:

- Generate a mesh for a given geometry,
- Perform a Tau3P run,
- If the run aborts (prematurely), generate a new mesh and return to previous step.

- If the run does not abort prematurely, halt.

Such a procedure is clearly expensive in terms of the number of man-hours that potentially may be devoted to a given meshing problem because both the meshing step and the Tau3P calculation are within the iterative loop. Smoothing is one way of producing another mesh for the iterative procedure. Our results will show that smoothing often produces a mesh which increases the abort-time, thus increasing the likelihood that the iteration will halt. However, to further reduce the time spent in meshing, the following modified iteration is proposed:

- Generate a mesh for a given geometry,
- Calculate the quality metrics (MPES, etc),
- If any metric lies outside the acceptable range, improve the mesh in various ways until the metrics are acceptable,
- Perform a Tau3P run, halt the procedure

In this approach, the iteration loop does not include Tau3P, so the iteration proceeds much faster than the previous iterative scheme. If one's best mesh still results in a prematurely aborted Tau3P run, one should probably not continue to search for better meshes because one has already used the best mesh.

The goal of the present and future investigations is to find ways to improve Tau3P abort-time on the FPU (Pulse Pillbox Unfiltered) and, later, the other cases by modifying the meshes to improve the various mesh quality metrics. In doing so, it is believed that a methodology will be developed that can be applied to more realistic meshing problems that commonly arise in the modeling and simulation of accelerators. Our tools for improving the abort-time are smoothing (node repositioning), edge biasing, interval assignment, remeshing of certain regions, and inserting prisms. In this paper we confine our activities to smoothing.

3. SMOOTHING STRATEGY

A typical 3D pillbox mesh (See Figure 1) consists of hexahedral elements. The hexahedral mesh is generated by first creating quadrilateral meshes on the beam cross-section (a quarter circle) and on the collar cross-section (a quarter annulus). These meshes are 'swept' along the beam axis to create hexahedra.

We limited our attention in this study to sixteen 2D beam cross-section meshes. The previous study showed that these cross-sections cover the full range of Tau3P abort-times. QC2 is our 'best' mesh in that,
without smoothing, it resulted in the largest abort-time. QC3 is our ‘worst’ mesh, having a singularity on the boundary. 2 The abort-time for QC3 is several orders of magnitude smaller than the other meshes. Four representative cross-section meshes are shown in Figure 3.

Our approach to smoothing of the pillbox mesh has potentially two stages: (1) smooth the beam cross-section mesh consisting of quadrilateral elements (nodes on the bounding curves of the cross-section are held fixed), and (2) smooth the hexahedral mesh (holding nodes on the bounding surfaces fixed).

We do not smooth the collar cross-section mesh (see Figure 4) since it would be difficult to improve the quality obtained on that surface (the meshes are smooth and nearly orthogonal). For the same reason, we do not attempt to improve the quality of any of the linking surface meshes. Finally, since the 3D mesh is a simple translation of the 2D cross-section meshes, there is little to be gained by smoothing the hexahedral mesh as in Step 2 above. For more general geometries, the choice of which surfaces to smooth would have to be considered on a case by case basis. 3 This issue will be explored in future work.

In the original study there were sixteen quadrilateral meshes generated on the beam (quarter-circle) cross-section. For each such cross-section, there are two meshes of interest in this study which we denote by the following types:

- Type UU. Neither the quadrilateral nor the hexahedral meshes are smoothed,

- Type SU. The quadrilateral mesh on the beam cross-section is smoothed, then swept to create the hexahedral mesh. The hexahedral mesh is not smoothed.

4. SMOOTHING THE BEAM CROSS-SECTION

In this section we compare Type UU meshes to Type SU meshes to explore potential benefits of 2D mesh smoothing to improve quality (and thus abort-time). To create the smoothed SU meshes, we first try condition number smoothing [5], as implemented in the CUBIT code. This smoother minimizes the following

---

2That does not mean that there is an element with zero area. Rather the underlying continuum map has a singularity. No element in this mesh has zero area.

3On more complex geometries one may still wish to smooth some or all of the surface meshes, thus 2D metrics remain relevant. On the other hand, 3D metrics are clearly needed when one needs to smooth the volume.

---
objective function by changing vertex positions:

\[ F = \sum_{j} Q(\varepsilon_j) \]  

(1)

where

\[ Q(\varepsilon_j) = \sum_{i=1}^{n} \kappa_{ij}^2 \]  

(2)

and

\[ \kappa_{ij} = \frac{L_{ij}^2 + L_{i+1,j}^2}{2A_{ij}} \]  

(3)

This smoother is designed to improve element shape (angles and aspect ratio) while avoiding inverted elements. A relatively tight tolerance was used during the optimization to ensure the SU meshes were close to the minimum of \( F \).

A visual inspection of the SU meshes reveals little difference between the unsmoothed and smoothed meshes, so the latter are not shown to save space. Differences are more apparent in the metric values. The metrics in the original study, MPCN, MPSM, and MPSJ, were calculated using only the quadrilateral beam meshes. That is, they are 2D metrics. To be more precise in this study, we re-label these metrics MPSCN-2D, MPSM-2D, and MPSJ-2D. The MPSCN metric in the previous study was not a 2D metric as it used information from the hexahedral mesh. For consistency we’ve introduced a new 2D metric, MPSCN-2D, which is computed using only the quadrilateral beam mesh. An additional metric, average primal condition number, APCN-2D, was added to monitor changes in the average values of the metrics to better understand the effect of condition number smoothing on abort-time.

We give values of MPSCN-2D, MPCN-2D, MPSM-2D, MPSJ-2D, and APCN-2D in Tables 1, 2, 3, 4, and 5 for both the initial unsmoothed (UU) meshes and for the cross-section smoothed (SU) meshes, along with the percent increase or decrease in abort-time. Table 6 gives the Tan3P abort-time for both the unsmoothed and smoothed meshes. Thirteen of the sixteen cross-sections smoothed resulted in an increased abort-time, a success rate of 81%. The average increase in abort-time was 40% and in several cases the abort-time increased an order of magnitude via smoothing. Abort-time decreased due to smoothing on three cross-sections, one, as much as 89.2%. We thus see that smoothing is potentially of considerable use in improving Tan3P abort-times but cannot be applied routinely.

To understand better what smoothing has wrought, we present in Figure 5 plots of percent change in the five metrics MPSCN, MPSM, APCN, MPSCJ and MPSJ versus percent change in abort-time. Each line segment corresponds to a single run such as QC1. The origin represents the UU position, while the other end of each line segment is the SU position.

Looking first at the plot for APCN, one sees that this metric value always decreases. This is because the condition number objective function that we minimize serves as an upper bound to the average condition number given by APCN. Next, consider the plot for MPSCN. The south-east (SE) quadrant of this plot consists of those runs for which MPSCN decreased (i.e., the quality improved) while the actual abort-time increased. Runs lying in the SE-quadrant thus have abort-time predictions which are consistent with abort-times achieved. Eight of the sixteen runs lie in the SE quadrant. Similarly, the north-west (NW) quadrant consists of runs for which MPSCN increased (i.e., the quality decreased) while the actual abort-time decreased. Runs in the NW-quadrant have abort-time predictions consistent with actual abort-times. Only one run (QC3) lies in the NW-quadrant. Runs in the NE and SW-quadrants have abort-time predictions which are inconsistent with the actual abort-times achieved. There were two runs in the SW-quadrant and five runs in the NE-quadrant. Presumably, runs in the SW and NE-quadrants appear inconsistent because other metrics are determining the actual abort-time achieved. Plots for MPSCJ and MPSJ show similar results. QC10bias stands out as highly unusual in that while MPSCN decreased modestly, abort-time decreased considerably.

**QC10bias** How can we explain the behavior of QC10bias (see Figure 6 for the UU and SU meshes)? As the figure shows, the mesh consists of a near-polar coordinate system in the outer regions of the beam, combined with a triangle-primitive (three block-structured submeshes) near the beam center.
<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPES-2D</th>
<th>Mesh</th>
<th>MPCM-2D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type UU</td>
<td>Type SU</td>
<td>% Change</td>
</tr>
<tr>
<td>QC1</td>
<td>0.00090</td>
<td>0.00092</td>
<td>+2.2</td>
</tr>
<tr>
<td>QC2</td>
<td>0.00107</td>
<td>0.00105</td>
<td>-1.9</td>
</tr>
<tr>
<td>QC2b</td>
<td>0.00102</td>
<td>0.00103</td>
<td>+1.0</td>
</tr>
<tr>
<td>QC3</td>
<td>0.00175</td>
<td>0.00143</td>
<td>-18.3</td>
</tr>
<tr>
<td>QC5</td>
<td>0.00107</td>
<td>0.00118</td>
<td>+10.3</td>
</tr>
<tr>
<td>QC5b</td>
<td>0.00109</td>
<td>0.00109</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC6</td>
<td>0.00070</td>
<td>0.00091</td>
<td>+30.0</td>
</tr>
<tr>
<td>QC7</td>
<td>0.00143</td>
<td>0.00101</td>
<td>-29.4</td>
</tr>
<tr>
<td>QC10t</td>
<td>0.00039</td>
<td>0.00037</td>
<td>-5.1</td>
</tr>
<tr>
<td>QC10p</td>
<td>0.00039</td>
<td>0.00047</td>
<td>+20.5</td>
</tr>
<tr>
<td>QC10c</td>
<td>0.00055</td>
<td>0.00067</td>
<td>+21.8</td>
</tr>
<tr>
<td>QC11</td>
<td>0.00079</td>
<td>0.00082</td>
<td>+3.8</td>
</tr>
<tr>
<td>QC11C</td>
<td>0.00060</td>
<td>0.00060</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC12</td>
<td>0.00040</td>
<td>0.00086</td>
<td>+115</td>
</tr>
<tr>
<td>QC10t</td>
<td>bias</td>
<td>0.00039</td>
<td>0.00032</td>
</tr>
<tr>
<td>QC11b</td>
<td>0.00079</td>
<td>0.00087</td>
<td>+3.8</td>
</tr>
</tbody>
</table>

Table 1: MPES-2D Metric for cross-section condition number smoothed meshes.

Table 2: MPCM-2D Metrics for cross-section condition number smoothed meshes.

The UU and SU meshes are nearly indistinguishable in the figure. Overlaying them directly on top of one another shows that the only visible difference in the SU mesh shows smoothing has pulled in the azimuthal lines somewhat closer to the beam center. Other possibly significant changes can be detected by looking at the metrics. First, the worst angle in the mesh occurs at the three-valent node. This angle remains virtually unchanged by smoothing and cannot account for the change in abort-time. The MPES metric shows that the minimum mesh edge-length was small in the UU mesh and made smaller by smoothing, this change might partly account for the decrease in abort-time. The MPES metric shows the SU mesh is smoother than the UU mesh (in terms of area transitions). This should have helped increase the abort-time, but evidently, this was insufficient to overcome other, adverse changes in the mesh. The most likely explanation for the decrease in abort-time is thus the decrease in MPES, especially since this occurred near the beam center, a critical region in which the solution is developed. In terms of developing a practical procedure for increasing abort-time, it would be helpful if one could know a priori that smoothing this mesh via condition number would decrease the abort-time, but a great deal of experience would be needed to have forseen this result.

QC11bias We also examine QC11bias to determine why smoothing had a slightly adverse effect on the abort-time. Figure 7 shows that the main change in the smoothed mesh is the improvement in MPSJ which occurs for one of the quadrilaterals in the paved portion of the mesh. MPSM decreased 6.1%, but this does not seem sufficient to account for the decrease in abort-time. Again, it is next to impossible to determine a priori from the metrics that condition number smoothing would decrease the abort-time in this example since the UU mesh has no obvious defect. Probably what would have worked better in this case is to add a few intervals along the biased curves to reduce the aspect ratio and to improve the paved mesh quality.

In summary, the important points illustrated in this section are, first, that condition number smoothing is often successful in significantly increasing abort-time. Second, our model of the relationship between abort-time and certain quality metrics is generally but not completely effective in predicting the effect of a particular smoothing operation on abort-time. This is seen by the preponderance of runs that fall in the SE and NW-quadrants of our compass plots. It is not known if our model could be made more effective by including additional metrics or if some other effect, such as relation of the mesh to the solution, is clouding the picture.

5. LAPLACIAN SMOOTHING

To understand better whether or not the choice of condition number smoothing is critical to increasing abort-time, we smoothed the same sixteen cross sections using Cubit’s Laplacian smoother. We believe this is a fair comparison because the QC meshes have good topology on a convex domain. Thus Laplacian smoothing is not expected to invert any of the cells of the mesh. In addition, it will tend to make the mesh
Figure 5: Effect of Condition Number Smoothing on Abort-Time

Figure 6: The UU and SU meshes for QC10tbias
<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPSM-2D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type UU</td>
</tr>
<tr>
<td>QC1</td>
<td>5.81</td>
</tr>
<tr>
<td>QC2</td>
<td>4.35</td>
</tr>
<tr>
<td>QC2b</td>
<td>4.05</td>
</tr>
<tr>
<td>QC3</td>
<td>3.29</td>
</tr>
<tr>
<td>QC5</td>
<td>5.40</td>
</tr>
<tr>
<td>QC5b</td>
<td>5.43</td>
</tr>
<tr>
<td>QC6</td>
<td>6.07</td>
</tr>
<tr>
<td>QC7</td>
<td>6.38</td>
</tr>
<tr>
<td>QC10t</td>
<td>9.23</td>
</tr>
<tr>
<td>QC10p</td>
<td>10.2</td>
</tr>
<tr>
<td>QC10c</td>
<td>8.27</td>
</tr>
<tr>
<td>QC11</td>
<td>5.79</td>
</tr>
<tr>
<td>QC11C</td>
<td>8.05</td>
</tr>
<tr>
<td>QC12</td>
<td>7.47</td>
</tr>
<tr>
<td>QC10t+bias</td>
<td>5.80</td>
</tr>
<tr>
<td>QC11bias</td>
<td>6.11</td>
</tr>
</tbody>
</table>

**Table 3**: MPSM-2D Metrics for cross-section condition number smoothed meshes.

lines smoother. The basic question is does smoothing of the mesh lines increase abort-time as effectively as condition number smoothing?

We give values of MPES-2D, MPCN-2D, MPSM-2D, MPSJ-2D, and APCN-2D in Tables 7, 8, 9, 10, and 11 for both the initial unsmoothed (UU) meshes and for the cross-section Laplacian-smoothed (SU) meshes, along with the percent increase or decrease in abort-time. Table 12 gives the Tau3P abort-time for both the unsmoothed and smoothed meshes. Thirteen of the sixteen cross-sections smoothed resulted in an increased abort-time, the same success rate as was achieved by condition number smoothing. The average increase in abort-time was 333% and in several cases the abort-time increased an order of magnitude via smoothing. Abort-time decreased due to smoothing on three cross-sections, one, as much as 61.7%. In six of sixteen cross-sections, Laplacian smoothing was more effective than condition number in increasing abort-time. We thus see that Laplacian smoothing is potentially of considerable use in improving Tau3P abort-times (at least when it does not invert the mesh). As illustrated by Figure 9, meshes which do not visually appear to have good quality can, in fact, increase the abort-time (probably in this case because more mesh nodes are concentrated in the critical area of the beam).

In Figure 8, we use the same type of plots shown in Figure 5 to examine the relationship between percentage change in MPCN, MPSM, APCN, MPES and MPSJ and percentage change in abort-time. The plot of APCN shows that most of the time the average con-
<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPSJ-2D</th>
<th>Type UU</th>
<th>Type SU</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1</td>
<td>0.836</td>
<td>0.809</td>
<td>-3.2</td>
<td></td>
</tr>
<tr>
<td>QC2</td>
<td>0.725</td>
<td>0.783</td>
<td>+8.0</td>
<td></td>
</tr>
<tr>
<td>QC2b</td>
<td>0.727</td>
<td>0.828</td>
<td>+13.9</td>
<td></td>
</tr>
<tr>
<td>QC3</td>
<td>0.039</td>
<td>0.039</td>
<td>+0.00</td>
<td></td>
</tr>
<tr>
<td>QC4</td>
<td>0.729</td>
<td>0.805</td>
<td>+10.4</td>
<td></td>
</tr>
<tr>
<td>QC5b</td>
<td>0.770</td>
<td>0.805</td>
<td>+4.6</td>
<td></td>
</tr>
<tr>
<td>QC6</td>
<td>0.706</td>
<td>0.760</td>
<td>+7.6</td>
<td></td>
</tr>
<tr>
<td>QC7</td>
<td>0.707</td>
<td>0.802</td>
<td>+13.4</td>
<td></td>
</tr>
<tr>
<td>QC10t</td>
<td>0.758</td>
<td>0.660</td>
<td>-12.9</td>
<td></td>
</tr>
<tr>
<td>QC10p</td>
<td>0.760</td>
<td>0.814</td>
<td>+7.1</td>
<td></td>
</tr>
<tr>
<td>QC10c</td>
<td>0.710</td>
<td>0.694</td>
<td>-2.3</td>
<td></td>
</tr>
<tr>
<td>QC11</td>
<td>0.737</td>
<td>0.708</td>
<td>-3.9</td>
<td></td>
</tr>
<tr>
<td>QC11C</td>
<td>0.679</td>
<td>0.608</td>
<td>-10.5</td>
<td></td>
</tr>
<tr>
<td>QC12</td>
<td>0.609</td>
<td>0.673</td>
<td>+10.5</td>
<td></td>
</tr>
<tr>
<td>QC10t+</td>
<td>0.758</td>
<td>0.764</td>
<td>+0.8</td>
<td></td>
</tr>
<tr>
<td>QC11bias</td>
<td>0.737</td>
<td>0.760</td>
<td>+14.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: MPSJ-2D Metrics for cross-section condition number smoothed meshes.

The condition number decreases. The two runs, QC10p and QC11bias stand out as having an increase APCN. The increase in QC11bias is a modest +0.9% and can be explained as due to the biasing along the sides of the mesh. The increase in QC10p is fairly large compared to most other runs and could be attributed to a high amount of 'pulling' towards the beam. The effect of Laplacian smoothing on MPCN is seen to be mostly random. We see again a small number of runs in the SE and NW quadrants for MPCN. Also the most runs occur in the NE quadrant for MPCN. For MPSM we see runs in all four quadrants, but with most occurring in the NE quadrant which is what we would expect.

6. CONCLUSION

In this study we smoothed sixteen pillbox cross-section meshes. Two different smoothers were used: condition number and Laplacian. Both smoothers proved generally effective in increasing the abort-time compared to the unsmoothed meshes. Of the unsmoothed meshes, QC2 had the largest abort-time (1.10e-06). Of the smoothed meshes, QC5 with Laplacian smoothing had the largest abort-time (1.36e-06). Thus smoothing was able to improve on the overall best abort-time achieved with no smoothing by 23.6%. This result suggests that, on this simple model problem, one can obtain good quality meshes even without smoothing. In such cases, smoothing will result in only a small increase in abort-time. In spite of this result we remain confident that smoothing will be an important tool in increasing abort-time in general because there are much more difficult SLAC geometries to mesh than the one selected for this study. This will be especially true for...
<table>
<thead>
<tr>
<th>Mesh</th>
<th>APCN-2D</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type UU</td>
<td>Type SU</td>
</tr>
<tr>
<td>QC1</td>
<td>1.08</td>
<td>1.06</td>
</tr>
<tr>
<td>QC2</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>QC2b</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>QC3</td>
<td>1.21</td>
<td>1.20</td>
</tr>
<tr>
<td>QC5</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>QC5b</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>QC6</td>
<td>1.03</td>
<td>1.02</td>
</tr>
<tr>
<td>QC7</td>
<td>1.05</td>
<td>1.04</td>
</tr>
<tr>
<td>QC10t</td>
<td>1.25</td>
<td>1.24</td>
</tr>
<tr>
<td>QC10p</td>
<td>2.05</td>
<td>2.00</td>
</tr>
<tr>
<td>QC10c</td>
<td>1.10</td>
<td>1.10</td>
</tr>
<tr>
<td>QC11</td>
<td>1.07</td>
<td>1.07</td>
</tr>
<tr>
<td>QC11C</td>
<td>1.06</td>
<td>1.04</td>
</tr>
<tr>
<td>QC12</td>
<td>1.06</td>
<td>1.04</td>
</tr>
<tr>
<td>QC10t+bias</td>
<td>1.25</td>
<td>1.24</td>
</tr>
<tr>
<td>QC11bias</td>
<td>1.08</td>
<td>1.07</td>
</tr>
</tbody>
</table>

**Table 5:** APCN-2D Metric for cross-section condition number smoothed meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>UU Abort</th>
<th>SU Abort</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1</td>
<td>2.30e-07</td>
<td>5.45e-07</td>
<td>+137.0</td>
</tr>
<tr>
<td>QC2</td>
<td>1.10e-06</td>
<td>1.21e-06</td>
<td>+10.0</td>
</tr>
<tr>
<td>QC2b</td>
<td>2.68e-07</td>
<td>9.26e-07</td>
<td>+245.5</td>
</tr>
<tr>
<td>QC3</td>
<td>2.62e-07</td>
<td>2.39e-10</td>
<td>-8.7</td>
</tr>
<tr>
<td>QC5</td>
<td>2.15e-07</td>
<td>9.57e-07</td>
<td>+345.1</td>
</tr>
<tr>
<td>QC5b</td>
<td>2.02e-07</td>
<td>8.30e-07</td>
<td>+310.9</td>
</tr>
<tr>
<td>QC6</td>
<td>2.96e-08</td>
<td>3.01e-07</td>
<td>+916.9</td>
</tr>
<tr>
<td>QC7</td>
<td>2.53e-08</td>
<td>3.16e-07</td>
<td>+1149</td>
</tr>
<tr>
<td>QC10t</td>
<td>2.00e-08</td>
<td>1.10e-07</td>
<td>+450.0</td>
</tr>
<tr>
<td>QC10p</td>
<td>1.48e-08</td>
<td>1.20e-07</td>
<td>+710.0</td>
</tr>
<tr>
<td>QC10c</td>
<td>2.71e-08</td>
<td>3.10e-07</td>
<td>+1044</td>
</tr>
<tr>
<td>QC11</td>
<td>8.93e-08</td>
<td>2.55e-07</td>
<td>+185.6</td>
</tr>
<tr>
<td>QC11C</td>
<td>2.84e-08</td>
<td>2.86e-07</td>
<td>+907.0</td>
</tr>
<tr>
<td>QC12</td>
<td>2.12e-07</td>
<td>4.90e-07</td>
<td>+131.1</td>
</tr>
<tr>
<td>QC10t+bias</td>
<td>1.52e-07</td>
<td>1.64e-08</td>
<td>-89.2</td>
</tr>
<tr>
<td>QC11bias</td>
<td>8.81e-08</td>
<td>7.63e-08</td>
<td>-13.4</td>
</tr>
</tbody>
</table>

**Table 6:** PPU Abort-times with percentage change for cross-section condition number smoothed meshes.

When applied to 3D hex meshes. On difficult geometries, the initial, unsmoothed mesh is likely to be of relatively poor quality, perhaps more along the lines of some of the less effective templates used in this study. As this study shows, when the initial quality is relatively poor, smoothing can result in considerable gains in abort-time (300-400 percent on average and an order of magnitude in some cases).

If one is going to smooth meshes to increase the abort-time, it is natural to ask what smoother should be used. As this paper shows, both condition number and Laplacian smoothing can be effective. Both smoothers increased the UU abort-time for a given topology around 80% of the time. Condition number smoothing increased the abort-time an average of 400% while Laplacian smoothing increased the abort-time an average of 333%. The abort-time achieved with condition number was better than the abort-time achieved with Laplacian smoothing in 60% of the cases. On the other hand, Laplacian smoothing gave the largest abort-time over all of the mesh topologies (QC5). In one case (QC11bias), neither condition number nor Laplacian smoothing was able to improve the UU abort-time. Often, one can use the metrics to predict which of the smoothed meshes will give the larger abort-time. For example, in QC12, the condition number-smoothed mesh clearly has better quality than the Laplacian mesh: MPCN is 1.49 vs. 2.21, MPSM is 6.15 vs. 8.15, and MPSJ is 0.673 vs. 0.453, respectively. Indeed, the abort-time for QC12 was almost six times longer for condition number than for Laplacian. In many cases, however, it is much less clear from looking at the metrics, which of the two smoothing methods will give the larger abort-time. For example, the abort-time for QC10t+bias was almost twenty times longer for Laplacian smoothing than for Condition number, yet a comparison of their metrics does not reveal anything particularly in favor of Laplacian (except possibly MPSM). Unless there is something obviously wrong with a given mesh, it does not appear possible to reliably predict the abort-time in advance using just metric values. This makes achievement of our modified iteration procedure (described in section 2) appear less attainable.

An obvious future line of inquiry would be to construct a hybrid objective function which trades off various aspects of mesh smoothness, edge-length, and element shape in order to simultaneously improve as many aspects of mesh quality as possible. This idea will be explored using the MESQUITE software [6].

To make this work of practical value to the SLAC meshing effort we will need to develop a methodology for when to smooth, what to smooth, and to incorporate other mesh quality improvement techniques such as mesh insertion. The methodology will be tested on more difficult meshing problems that will require investigation into 3D smoothing and mesh quality.

**References**

Figure 9: The SU mesh for QC10t bias when Laplacian smoothed


<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPES-2D Type UU</th>
<th>MPES-2D Type SU</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1</td>
<td>0.00090</td>
<td>0.00097</td>
<td>+1.8</td>
</tr>
<tr>
<td>QC2</td>
<td>0.00107</td>
<td>0.00116</td>
<td>+8.4</td>
</tr>
<tr>
<td>QC2b</td>
<td>0.00102</td>
<td>0.00109</td>
<td>+6.9</td>
</tr>
<tr>
<td>QC3</td>
<td>0.00175</td>
<td>0.00157</td>
<td>-10.3</td>
</tr>
<tr>
<td>QC5</td>
<td>0.00107</td>
<td>0.00119</td>
<td>+11.2</td>
</tr>
<tr>
<td>QC5b</td>
<td>0.00109</td>
<td>0.00114</td>
<td>+4.6</td>
</tr>
<tr>
<td>QC6</td>
<td>0.00070</td>
<td>0.00092</td>
<td>+31.4</td>
</tr>
<tr>
<td>QC7</td>
<td>0.00143</td>
<td>0.00091</td>
<td>-36.4</td>
</tr>
<tr>
<td>QC10t</td>
<td>0.00030</td>
<td>0.00029</td>
<td>-25.6</td>
</tr>
<tr>
<td>QC10p</td>
<td>0.00030</td>
<td>0.00036</td>
<td>-7.7</td>
</tr>
<tr>
<td>QC10c</td>
<td>0.00055</td>
<td>0.00065</td>
<td>+18.2</td>
</tr>
<tr>
<td>QC11</td>
<td>0.00079</td>
<td>0.00079</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC11C</td>
<td>0.00060</td>
<td>0.00082</td>
<td>+36.7</td>
</tr>
<tr>
<td>QC12</td>
<td>0.00040</td>
<td>0.00096</td>
<td>+140</td>
</tr>
<tr>
<td>QC10t bias</td>
<td>0.00030</td>
<td>0.00029</td>
<td>-25.6</td>
</tr>
<tr>
<td>QC11bias</td>
<td>0.00079</td>
<td>0.00079</td>
<td>+0.0</td>
</tr>
</tbody>
</table>

Table 7: MPES-2D Metric for cross-section Laplacian smoothed meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPCN-2D Type UU</th>
<th>MPCN-2D Type SU</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1</td>
<td>1.20</td>
<td>1.40</td>
<td>+16.7</td>
</tr>
<tr>
<td>QC2</td>
<td>1.38</td>
<td>1.38</td>
<td>0.0</td>
</tr>
<tr>
<td>QC2b</td>
<td>1.38</td>
<td>1.37</td>
<td>-0.7</td>
</tr>
<tr>
<td>QC3</td>
<td>2.29</td>
<td>2.55</td>
<td>+11.4</td>
</tr>
<tr>
<td>QC5</td>
<td>1.30</td>
<td>1.30</td>
<td>0.0</td>
</tr>
<tr>
<td>QC5b</td>
<td>1.31</td>
<td>1.41</td>
<td>+7.6</td>
</tr>
<tr>
<td>QC6</td>
<td>1.88</td>
<td>1.21</td>
<td>-35.6</td>
</tr>
<tr>
<td>QC7</td>
<td>1.43</td>
<td>1.58</td>
<td>+10.5</td>
</tr>
<tr>
<td>QC10t</td>
<td>1.53</td>
<td>1.52</td>
<td>-0.6</td>
</tr>
<tr>
<td>QC10p</td>
<td>3.49</td>
<td>3.62</td>
<td>+3.7</td>
</tr>
<tr>
<td>QC10c</td>
<td>1.41</td>
<td>1.29</td>
<td>-8.5</td>
</tr>
<tr>
<td>QC11</td>
<td>1.37</td>
<td>1.90</td>
<td>+38.7</td>
</tr>
<tr>
<td>QC11C</td>
<td>1.72</td>
<td>3.23</td>
<td>+87.8</td>
</tr>
<tr>
<td>QC12</td>
<td>1.64</td>
<td>2.21</td>
<td>+34.8</td>
</tr>
<tr>
<td>QC10t bias</td>
<td>1.53</td>
<td>1.59</td>
<td>+3.9</td>
</tr>
<tr>
<td>QC11bias</td>
<td>1.81</td>
<td>2.15</td>
<td>+18.8</td>
</tr>
</tbody>
</table>

Table 8: MPCN-2D Metrics for cross-section Laplacian smoothed meshes.
<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPSM-2D</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type U</td>
<td>Type SU</td>
<td>% Change</td>
</tr>
<tr>
<td>QC1</td>
<td>5.81</td>
<td>6.83</td>
<td>+17.6</td>
</tr>
<tr>
<td>QC2</td>
<td>4.35</td>
<td>3.98</td>
<td>-8.5</td>
</tr>
<tr>
<td>QC2b</td>
<td>4.05</td>
<td>3.69</td>
<td>-8.9</td>
</tr>
<tr>
<td>QC3</td>
<td>3.29</td>
<td>7.26</td>
<td>+121</td>
</tr>
<tr>
<td>QC5</td>
<td>5.40</td>
<td>4.24</td>
<td>-21.5</td>
</tr>
<tr>
<td>QC5b</td>
<td>5.43</td>
<td>5.34</td>
<td>-1.7</td>
</tr>
<tr>
<td>QC6</td>
<td>6.07</td>
<td>4.77</td>
<td>-21.4</td>
</tr>
<tr>
<td>QC7</td>
<td>6.38</td>
<td>5.05</td>
<td>-20.9</td>
</tr>
<tr>
<td>QC10t</td>
<td>9.23</td>
<td>2.80</td>
<td>-69.7</td>
</tr>
<tr>
<td>QC10p</td>
<td>10.2</td>
<td>4.62</td>
<td>-54.7</td>
</tr>
<tr>
<td>QC10c</td>
<td>8.27</td>
<td>3.53</td>
<td>-57.3</td>
</tr>
<tr>
<td>QC11</td>
<td>5.79</td>
<td>9.34</td>
<td>+61.3</td>
</tr>
<tr>
<td>QC11C</td>
<td>8.05</td>
<td>10.9</td>
<td>+35.4</td>
</tr>
<tr>
<td>QC12</td>
<td>7.47</td>
<td>8.15</td>
<td>+9.1</td>
</tr>
<tr>
<td>QC10t+</td>
<td>5.80</td>
<td>2.78</td>
<td>-52.1</td>
</tr>
<tr>
<td>QC11+</td>
<td>6.11</td>
<td>8.54</td>
<td>+39.8</td>
</tr>
</tbody>
</table>

Table 9: MPSM-2D Metrics for cross-section Laplacian smoothed meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>APCN-2D</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type U</td>
<td>Type SU</td>
<td>% Change</td>
</tr>
<tr>
<td>QC1</td>
<td>1.08</td>
<td>1.03</td>
<td>-4.6</td>
</tr>
<tr>
<td>QC2</td>
<td>1.02</td>
<td>1.02</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC2b</td>
<td>1.02</td>
<td>1.02</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC3</td>
<td>1.21</td>
<td>1.21</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC5</td>
<td>1.02</td>
<td>1.02</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC5b</td>
<td>1.02</td>
<td>1.02</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC6</td>
<td>1.03</td>
<td>1.01</td>
<td>-1.9</td>
</tr>
<tr>
<td>QC7</td>
<td>1.05</td>
<td>1.03</td>
<td>-1.9</td>
</tr>
<tr>
<td>QC10t</td>
<td>1.25</td>
<td>1.24</td>
<td>-0.8</td>
</tr>
<tr>
<td>QC10p</td>
<td>2.05</td>
<td>2.20</td>
<td>+7.3</td>
</tr>
<tr>
<td>QC10c</td>
<td>1.10</td>
<td>1.06</td>
<td>-3.6</td>
</tr>
<tr>
<td>QC11</td>
<td>1.07</td>
<td>1.07</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC11C</td>
<td>1.06</td>
<td>1.03</td>
<td>-2.8</td>
</tr>
<tr>
<td>QC12</td>
<td>1.06</td>
<td>1.04</td>
<td>-1.9</td>
</tr>
<tr>
<td>QC10t+</td>
<td>1.25</td>
<td>1.25</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC11+</td>
<td>1.08</td>
<td>1.09</td>
<td>+0.9</td>
</tr>
</tbody>
</table>

Table 11: APCN-2D Metric for cross-section Laplacian smoothed meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>MPSJ-2D</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type U</td>
<td>Type SU</td>
<td>% Change</td>
</tr>
<tr>
<td>QC1</td>
<td>0.836</td>
<td>0.716</td>
<td>-14.4</td>
</tr>
<tr>
<td>QC2</td>
<td>0.725</td>
<td>0.723</td>
<td>-2.8</td>
</tr>
<tr>
<td>QC2b</td>
<td>0.727</td>
<td>0.731</td>
<td>+5.5</td>
</tr>
<tr>
<td>QC3</td>
<td>0.039</td>
<td>0.039</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC5</td>
<td>0.729</td>
<td>0.772</td>
<td>+5.9</td>
</tr>
<tr>
<td>QC5b</td>
<td>0.770</td>
<td>0.712</td>
<td>-7.5</td>
</tr>
<tr>
<td>QC6</td>
<td>0.706</td>
<td>0.837</td>
<td>+18.6</td>
</tr>
<tr>
<td>QC7</td>
<td>0.707</td>
<td>0.670</td>
<td>-5.2</td>
</tr>
<tr>
<td>QC10t</td>
<td>0.758</td>
<td>0.660</td>
<td>-12.9</td>
</tr>
<tr>
<td>QC10p</td>
<td>0.760</td>
<td>0.591</td>
<td>-22.2</td>
</tr>
<tr>
<td>QC10c</td>
<td>0.710</td>
<td>0.804</td>
<td>+13.2</td>
</tr>
<tr>
<td>QC11</td>
<td>0.737</td>
<td>0.526</td>
<td>-28.6</td>
</tr>
<tr>
<td>QC11C</td>
<td>0.679</td>
<td>0.310</td>
<td>-54.3</td>
</tr>
<tr>
<td>QC12</td>
<td>0.609</td>
<td>0.453</td>
<td>-25.6</td>
</tr>
<tr>
<td>QC10t+</td>
<td>0.758</td>
<td>0.631</td>
<td>-16.8</td>
</tr>
<tr>
<td>QC11+</td>
<td>0.737</td>
<td>0.510</td>
<td>-30.8</td>
</tr>
</tbody>
</table>

Table 10: MPSJ-2D Metrics for cross-section Laplacian smoothed meshes.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>UU Abort</th>
<th>SU Abort</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1</td>
<td>2.30e-07</td>
<td>6.60e-07</td>
<td>+187</td>
</tr>
<tr>
<td>QC2</td>
<td>1.10e-06</td>
<td>4.21e-07</td>
<td>-61.7</td>
</tr>
<tr>
<td>QC2b</td>
<td>2.68e-07</td>
<td>4.26e-07</td>
<td>+59.0</td>
</tr>
<tr>
<td>QC3</td>
<td>2.62e-10</td>
<td>2.62e-10</td>
<td>+0.0</td>
</tr>
<tr>
<td>QC5</td>
<td>2.15e-07</td>
<td>1.36e-06</td>
<td>+533</td>
</tr>
<tr>
<td>QC5b</td>
<td>2.02e-07</td>
<td>5.58e-07</td>
<td>+176</td>
</tr>
<tr>
<td>QC6</td>
<td>2.96e-08</td>
<td>2.14e-07</td>
<td>+623</td>
</tr>
<tr>
<td>QC7</td>
<td>2.53e-08</td>
<td>1.43e-07</td>
<td>+465</td>
</tr>
<tr>
<td>QC10t</td>
<td>2.00e-08</td>
<td>2.03e-07</td>
<td>+915</td>
</tr>
<tr>
<td>QC10p</td>
<td>1.48e-08</td>
<td>6.03e-08</td>
<td>+307</td>
</tr>
<tr>
<td>QC10c</td>
<td>2.71e-08</td>
<td>1.69e-07</td>
<td>+524</td>
</tr>
<tr>
<td>QC11</td>
<td>8.93e-08</td>
<td>1.32e-07</td>
<td>+478</td>
</tr>
<tr>
<td>QC11C</td>
<td>2.84e-08</td>
<td>4.60e-07</td>
<td>+1520</td>
</tr>
<tr>
<td>QC12</td>
<td>2.12e-07</td>
<td>8.36e-08</td>
<td>-60.6</td>
</tr>
<tr>
<td>QC10t+</td>
<td>1.52e-07</td>
<td>3.07e-07</td>
<td>+102</td>
</tr>
<tr>
<td>QC11+</td>
<td>8.81e-08</td>
<td>8.26e-08</td>
<td>-6.2</td>
</tr>
</tbody>
</table>

Table 12: PPU Abort-times with percentage change for cross-section Laplacian-smoothed meshes.
BACK TO EDGE FLIPS IN 3 DIMENSIONS

Paul Louis GEORGE† & Houman BOROUCHAKI‡

† INRIA, projet GAMMA, Domaine de Voluceau Rocquencourt
BP 105, 78153 Le Chesnay cedex, France
email : paul-louis.george@inria.fr
‡ Université de Technologie de Troyes, GSM-LASMIS
BP 2060, 10010 Troyes cedex, France
email : houman.borouchaki@utt.fr

ABSTRACT

We return to the general edge flip in three dimensions. We demonstrate it is nothing other than a combination of elementary flips. Various properties of this operator are discussed, including conditions that make a flip possible (thus making a set of tets sharing an edge reducible). We discuss also about the existence of other edge based operators and a number of applications are envisaged including an exotic use of some degree of anisotropy.

Keywords: Tetrahedral meshes, edge flip, edge removal (reduction), anisotropy, optimization, delanunisation, boundary integrity.

1. INTRODUCTION AND MOTIVATIONS

The “2-3” flip considers 2 tetrahedra sharing a face and replaces these elements by 3 tetrahedra sharing the edge whose endpoints are opposite the common face, this being made if the resulting pattern is still valid. This flip is the immediate extension of the well-known edge flip (the “2-2” flip or diagonal swapping) in two dimensions. The “2-3” flip was discussed a long time ago and used for various purposes including mesh optimization, boundary enforcement in Delaunay based mesh generation method and some other mesh modifications. The “3-2” flip replacing, when valid, the 3 tets sharing an edge by means of 2 tets sharing a face (that opposite the two endpoints of the above edge endpoints) can be seen as the inverse of the “2-3” flip. The general (of arbitrary order) flip dealing with tets sharing a given edge is the natural extension to three dimensions of the “2-2” flip. The complexity of such flips is cubic (w.r.t. the number of interested tets) while a subtle implementation leads to a almost linear time.

A number of authors, [10], [11], [12], [2], [3], [14], etc., discussed about flips regarding, in specific, the Delaunay triangulation construction in three dimensions. They show that the “2-3” and “3-2” flips can be used to optimize to some degree an arbitrary triangulation with respect to the Delaunay criterion.

In this paper we discuss about the general flip and we show that it is a combination of elementary “2-3” flips together with a “3-2” or a “4-4” flip. This general flip can be used to remove an edge in a mesh. We give some conditions that make this removal (reduction) possible. Also we discuss if there is any other type of flips and, to conclude, we indicate various applications of such flips and we propose an anisotropic point of view.

2. BACK TO THE “2-3” FLIP

The “2-3” flip considers the polyhedron made up of 2 tets sharing a face. If this polyhedron is convex, Figure 1 (left), or not, Figure 1 (right), there exists an alternate tet configuration made up of 3 tets which covers the same volume or such a solution is not valid. The
initial situation reads $K_1 = [M_1M_2M_3\alpha]$ and $K_2 = [\beta M_1M_2M_3]$. The resulting situation, when valid, reads: $K^*_1 = [M_1\beta \alpha M_2]$, $K^*_2 = [M_2\beta \alpha M_3]$ and $K^*_3 = [M_3\beta \alpha M_1]$. In other words, we obtain a shell made up of 3 elements, such a set being defined now. 

**Definition 1.** Given an edge, a shell is the polyhedron made up of the tets sharing this edge. The common edge is the generating edge of the shell. The vertices other than the edge endpoints constitute the generating polygon of the shell. 

Note that the above polygon is ordered and, in general, non planar. Note also that only closed shells are discussed in the paper, eg the edge is fully surrounded by tets.

From the topological point of view, the “2-3” flip removes one edge and creates one face. To validate such a flip, one has to check the positiveness of the 3 resulting tets (in other words, the polyhedron is convex or $\alpha \beta$ passes through triangle $M_1 M_2 M_3$).

3. **BACK TO THE “3-2” FLIP**

The “3-2” flip considers the polyhedron made up of 3 tets sharing an edge. If the so-defined polyhedron is convex, Figure 3 (left), or not, Figure 3 (right), there exists a alternate valid configuration made up of 2 tets which covers the same volume, Figure 4, or this case is not valid. Note that in a non-convex case, edge $\alpha \beta$ does not cut the triangle whose vertices are other than $\alpha$ and $\beta$ while the supporting line of $\alpha \beta$ cuts it. In other words, the plane of this triangle separates $\alpha$ from $\beta$ or not. The configuration where the plane is not a separation plane is called a perfect Christmas tree and cannot be remeshed.

**Definition 2.** A shell is a perfect Christmas tree if its generating polygon is planar and does not separate the two endpoints of its generating edge. 

4. **THE NECESSARY “4-4” FLIP**

A 4-tet shell simply reads (after permuting the indices of its generating polygon): $K_1 = [M_1\alpha \beta M_2]$, $K_2 = [M_2\alpha \beta M_3]$, $K_3 = [M_3\alpha \beta M_4]$ and $K_4 =$
Definition 3. A shell is a Christmas tree if no mesh of its generating polygon exists which separates $\alpha$ from $\beta$. □

A shell is not a Christmas tree if a mesh of its generating polygon for which all the triangles are visible by $\alpha$ and $\beta$ exists and, as a consequence, edge $\alpha\beta$ cuts this (separation) mesh at a unique point (inside a triangle or on one triangle edge).

A perfect Christmas tree is then a peculiar Christmas tree. A non-Christmas tree shell with a planar generating polygon has a separation plane for $\alpha$ and the other points and another separation plane (possibly the same) for $\beta$ and the other points, thus the definition is consistent.

4.1 Reducing a 4-tet shell using “2-3” and “3-2” flips

As $K_2$ and $K_3$ share face $(\alpha M_2 \alpha M_3)$, a “2-3” flip can be envisaged. If possible, a brute force reading leads to $T^*_1 = [\alpha M_1 \beta M_2 \beta M_3 \beta M_4]$ and $T^*_2 = [\beta M_2 \beta M_3 \beta M_4]$ and $T^*_3 = [\beta M_1 \beta M_3 \beta M_4]$, to which we must add $K_2$ and $K_4$. Edge $\alpha\beta$ is now common to 3 elements, eg $T^*_1$, $T^*_3$, and $K_4$. A “3-2” flip can be then envisaged. From $T^*_1 = [\alpha M_1 \alpha M_3 \beta M_4]$ and $K_3 = [\beta M_2 \beta M_3 \beta M_4]$ and $K_4 = [\beta M_1 \beta M_3 \beta M_4]$, we find, when valid, $T^*_4 = [\alpha M_1 \alpha M_3 \beta M_2 \beta M_4]$ and $T^*_5 = [\alpha M_1 \alpha M_3 \beta M_2 \beta M_4]$. The resulting tets are then $T^*_2 = [\alpha M_1 \alpha M_3 \beta M_2 \beta M_4]$, $T^*_3 = [\beta M_2 \beta M_3 \beta M_4]$, $T^*_4 = [\alpha M_1 \alpha M_3 \beta M_2 \beta M_4]$, and $T^*_5 = [\alpha M_1 \alpha M_3 \beta M_2 \beta M_4]$. In other words, the $M_i$’s polygon is now meshed by means of 2 triangles, eg $M_1 M_3 M_4$ and $M_1 M_2 M_3$. The solution is made up of 4 tets formed by joining $\alpha$ and $\beta$ with these 2 triangles.

While edge $\alpha\beta$ is no longer a mesh edge, we say we have reduced the given shell (thus the term shell reduction).

Using a “2-3” flip on elements $K_2$ and $K_3$, we obtain an alternate solution based on the two alternate faces covering the polygon, eg $M_1 M_2 M_4$ and $M_2 M_3 M_4$. Moreover, there is no more solutions, another “2-3” flip leading to the same combinations.

4.2 Impossible reduction of a 4-tet shell using these flips

There is a configuration for which exists an alternate mesh which is not obtained in the above way (eg by means of “2-3” and “3-2” flips).

Let us consider a convex shell (thus an alternate mesh clearly exists) and let us assume that edge $\alpha\beta$ cuts segment $M_1 M_3$ together with segment $M_2 M_4$, then no “2-3” flip is valid. As a consequence, it is strictly needed to define a “4-4” flip which directly constructs the solution whose existence is known in advance.

4.3 The “4-4” flip

Here we follow a simple idea (used for the higher order flips). We consider the $M_i$’s polygon, we mesh it by means of triangles and, finally, we join these triangles with vertices $\alpha$ and $\beta$. Therefore, there are at most 2 solutions.

5. GENERAL HIGHER ORDER FLIPS

Such flips consider shells made up of $n$ elements. Such a shell reads $K_i = [M_i \alpha \beta M_{i+1}]$ with $i = 1, \ldots, n$ and $M_{n+1} \equiv M_1$.

5.1 Combination of elementary flips and direct flip

Let $C_n$ be a $n$ tet shell, with evident while abusive notations we can write $C_n = (2-3)^n \cup C_{n-1}$, as soon as $n > 4$, this being possible as one “2-3” flip is valid. Therefore, if $n = 4$ such flips are valid, one can write $C_n = (n-4)2-3^n \cup C_0$, and an ultimate “4-4” flip, if valid, removes edge $\alpha\beta$ from this last shell. Using the same argument as that for a “4-4” flip, we can say that $C_n = (n-3)^{n-3}2-3^n \cup C_3$, is not, in general, a way to access the expected solution, in specific, while knowing it exists.

However, the computer writing, for $n \geq 4$, is much more faster (while more technical) if we are given in advance all the candidate solutions. This reduces to enumerate all the $a priori$ possible remeshing of a
polygon with $n$ sides, see Figure 5 for $n = 5$ and Figure 6 for $n = 6$. Each triangle in these remeshings is then connected with $\alpha$ and $\beta$ to constructing the desired tets. The direct flip is based on the data of the cat-

![Figure 5](image1)

Figure 5. The 5 solutions related to a 5-side generating polygon.

alogue of all the a priori possible solutions and leads to pick up in this series one or more instead of applying a number of “2-3” flips so as to obtain a 3 (or 4)-tet shell.

![Figure 6](image2)

Figure 6. The 14 solutions for a 6-side generating polygon. Top, 3 edges are incident to each $M_i$, then, 2 edges are incident to $M_1$ to $M_2$ and to $M_3$ where only 2 cases appear; then, the other cases, related to $M_4$ and $M_5$, have already been seen.

5.2 Is a shell reducible?

Definition 4. A shell is reducible if there exists a remeshing of the corresponding polyhedron where its generating edge is no longer a tet edge (it has been removed).

On the fly, we already meet some conditions that make a shell reducible or not.

- A perfect Christmas tree is not reducible.
- For $n = 3$, a shell other than a (perfect) Christmas tree is reducible (indeed, edge $\alpha \beta$ cuts the generating triangle and, therefore, as $\alpha$ and $\beta$ see this triangle, the 2 solution tets are valid).

To continue the discussion, we have to consider the general case ($n \geq 3$ and a non planar polygon) and to find conditions that make the shell reducible. We will demonstrate that:

- if the polygon is planar, all shells other than a (perfect) Christmas tree are reducible,
- for $n = 4$, the same occurs for a shell other than a Christmas tree provided an additional condition,
- and this extends to an arbitrary $n$.

Planar polygon. For a planar generating polygon, the proof is obvious. Any triangular meshes of the polygon is such that each of its triangles is visible by $\alpha$ and by $\beta$ (as the plane separates these two points), thus the resulting tets are valid (positive volumes). Note that the fact that the polygon is convex or not is not an issue. This ends the proof of reductibility for a shell with a planar polygon.

Now, we will demonstrate that this proof can be obtained by means of a number of “2-3” flips ended by an unique “3-2” or “4-4” flip. Let $P$ be the intersection point of segment $\alpha \beta$ with the plane supporting the polygon. If this polygon has more than 5 sides, there exists one index $i$ such that triangles $M_{i-1} M_i M_{i+1}$ and $M_{i+1} M_{i+2} M_{i+3}$ (after a modulo) lie inside the polygon. As the intersection of these 2 triangles is point $M_{i+1}$, point $P$ cannot belong to these 2 triangles. As a consequence, applying a “2-3” flip reduces by 1 the polygon. For example, if triangle $M_{i-1} M_i M_{i+1}$ does not contain $P$, a “2-3” flip applied to tet $M_{i-1} M_i M_{i+1}$ and $M_{i+1} M_{i+2} M_{i+3}$ results in tets $M_i M_{i-1} M_{i+1} \beta \beta M_{i+1} M_{i+2} M_{i+3}$ and, therefore, only one of these tets includes edge $\alpha \beta$. Note that this flip removes triangle $M_i \beta M_{i+1}$ and forms edge $M_{i-1} M_{i+1}$. This operation is possible because triangle $M_{i-1} M_i M_{i+1}$ separates $\alpha$ from $\beta$ and edge $M_{i-1} M_{i+1}$ cuts triangle $M_i \beta M_{i+1}$. Thus, point $M_i$ is no longer a vertex of the generating polygon. We then repeat the same construction until a polygon with 3 or 4 sides remains where an ultimate “3-2” or “4-4” flip applies.

Non planar polygon. In this case, the proof for the reductibility is obvious while it is more subtle to see that a combination of flips gives the solution (if valid).
The reductibility results from the definition of what a Christmas tree is. For shells other than such a tree, the existence of a mesh such that \( \alpha \) and \( \beta \) are visible by the triangles covering the polygon guarantees that the corresponding tets are valid. Thus, edge \( \alpha \beta \) is no longer a mesh edge and the reductibility holds.

To see that the solutions results from a combination of flips, we first look at the case \( n = 4 \) before noticing that the general case reduces to the same simple situation.

For \( n = 4 \), assumed a non-Christmas tree case, there exists a mesh made up of 2 triangles visible by \( \alpha \) and \( \beta \). The generating edge necessarily cuts this mesh. If the intersection falls inside one of these triangles the other triangle allows for a “2-3” flips resulting in a 3-tet shell, which is necessarily convex, thus one “3-2” flips gives a solution. If the intersection is on the edge common to these 2 triangles, a “4-4” flip gives a solution. Therefore, for \( n = 4 \), a shell is reducible.

For an arbitrary \( n \), if the shell is reducible it exists a triangular mesh of the generating polygon which separates \( \alpha \) from \( \beta \). In this mesh exists (see below) a triangle made up of three consecutive \( M_i \)’s, say \( M_{j-1} \tilde{M}_j M_{j+1} \), that does not cut \( \alpha \beta \) (and separates \( \alpha \) from \( \beta \)). This property makes the “2-3” flip removing triangle \( \beta \alpha \tilde{M}_j \) possible and reduces by 1 the size of the generating polygon (indeed, vertex \( M_j \) is no longer a member of the updated polygon). As the reduced shell remains reducible (with the remaining triangles of the initial triangular mesh of the generating polygon), the same applies for the various reduced configurations. Once these flips have been applied, it remains a shell where \( n = 3 \) (thus reducible) or \( n = 4 \) for which the above discussion applies. To conclude, a shell where \( n > 4 \) is reducible as soon as it is not a Christmas tree.

To complete the proof, it is needed to see that above triangle \( \tilde{M}_{j-1} \tilde{M}_j M_{j+1} \) exists. Let us consider a plane orthogonal to \( \alpha \beta \) cutting this segment. The projection of the polygon onto this plane is a simple polygon (eg. non self-intersecting) surrounding \( \alpha \beta \). Indeed, this polygon is star-shaped with respect to the intersection point of the plane with segment \( \alpha \beta \) because all the tets in the shell with \( \alpha \) and \( \beta \) have a positive volume and the projection of the plane of these tets maintains the orientation of the boundary of the polygon with respect to segment \( \alpha \beta \).

In other words, the correctness of the orientation of the projected polygon holds if the volume of the \( M_i \alpha \beta \tilde{M}_{i+1} \)’s has, for each of these tets, the same sign than the volume of the tets \( \tilde{M}_i \alpha \beta \tilde{M}_{i+1} \) where \( \tilde{M}_i \) is the projection of \( M_i \) onto the plane. At a factor 6, we have \( V = \beta \alpha . (\tilde{M}_i \wedge \beta \tilde{M}_{i+1}) \), then we compute \( \tilde{V} = \beta \alpha . (\beta \tilde{M}_i \wedge \beta \tilde{M}_{i+1}) \).

Thus, it is sufficient to analyse the (planar) projected configuration. As \( n > 4 \), there are at least 2 triangles based on three consecutive vertices and edge \( \alpha \beta \) cannot cut both of them. Therefore, one of these triangles allows for the solution.

In other words, if a shell is reducible, its reduction can be obtained using a number of “2-3” flips with a “3-2” or a “4-4” flip.

Note that in this reasoning we have considered the solution (the above triangular mesh) to determine the necessary “2-3” flips. Thus, these flips are not known in advance and the complexity of the method relies in effectively finding what flips must be applied.

A couple of remarks. All the previous discussion (apart for the non reductibility) is no longer valid, in practice, if one likes to include quality concerns (and not only a volume check). Also, a more restrictive definition of a Christmas tree can be advocated, eg, a Christmas tree occurs when there are not two planes (and not only a non planar triangular mesh) that separates, one \( \alpha \) and the other points, the other \( \beta \) and the other points.

6. COMPLEXITY ISSUES

To discuss the complexity of the flips, we first recall the number of possible triangulations and the number of different triangles covering the generating polygon of an arbitrary shell. Then, we turn to the theoretical complexity of a flip (for shell reduction or for mesh optimization) before restricting ourselves to the actual cases where \( n \) is relatively small (up to 6 or 7).

6.1 Number of solutions versus \( n \)

Table 1 gives \( N_n \), the number of possible triangulations as a function of \( n \). It also gives \( T n \) the number of different triangles in one possible triangulation. This concerns the topological point of view and not any validity aspect.
We have \( N_n = \text{Cat}(n - 1) \) where the Catalan number is involved which reads \( \text{Cat}(n) = \frac{(2n-2)!!}{n!(n-1)!} \). On the other hand, \( Tr_n = C_n^0 \) holds.

While being a classical result, we have pleasure to establish the value of the Catalan number. To this end, let us consider a (ordered) series of objects simply denoted as \((1 \, 2 \, 3 \ldots \, n)\). Let \( S_n \) be the number of combinations of the various different grouping of those objects. To find a recursion about \( S_n \), we can write as a first case the grouping of \((1)\) with the \( n-1 \) other objects \((23 \ldots n)\): \( G_1 = (1) (23 \ldots n) \), as a second case, we consider the grouping of \((12)\) with \(34 \ldots n\), eg \( G_2 = (1 \, 2) (3 \ldots n) \) and, ..., as case \# \( i \), we have \( G_i = (12 \ldots i) (i+1 \, i+2 \ldots n) \).

Thus, \( S_n = G_1 + G_2 + \ldots + G_{n-1} \), in other words, \( S_n = S_1 S_{n-1} + S_2 S_{n-2} + S_3 S_{n-3} + \ldots + S_{n-1} S_1 \), and then \( S_n = \sum_{i=1}^{n-1} S_i S_{n-i} \) holds. To exhibit an explicit writing for \( S_n \), we consider the polynomial associated with \( S_n \), eg \( F(z) = \sum_{i=1}^{n} z_i^2 \). A simple calculation shows that \( F(z) F(z) = F(z) - z \), from which we have \( F(z) = \frac{1-\sqrt{1-4z}}{2} \).

Let us expand \( \sqrt{1-4z} \) nearby 0. To this end, we look at the expansion of \((1+z)^n\) for a small \( z \). We have \((1+z)^n = 1 + n z + \frac{n(n-1)}{2} z^2 + \frac{n(n-1)(n-2)}{3!} z^3 + \ldots \). For \( m = \frac{1}{2} \) and \( \varepsilon = -4z \), the coefficient of the term in \( z^i \), for \( i \) not 0, reads: \( \frac{i!}{n!} \left( \frac{1}{2} - 1 \right) \left( \frac{1}{2} - 2 \right) \ldots (\frac{1}{2} - i + 1) \left( -4 \right)^i \), after factorizing \( \frac{1}{2} \), we have: \( \frac{1}{n!} \left( \frac{1}{2} \right)^i (\frac{1}{2} - 1)(1 - 2)(1 - 4) \ldots (1 - 2i + 2) \), or again \(-\frac{1}{n!} (\frac{1}{2} - 1)(1 - 2)(1 - 4) \ldots (1 - 2i + 2) \), or, finally, \(-\frac{1}{n!} (\frac{1}{2} - 1)(1 - 2)(1 - 4) \ldots (1 - 2i + 2) \). As in \( F(z) \) the coefficient of the term in \( z^i \) is 1, we have \( F(z) = \sum_{i=1}^{\infty} \frac{(2i-2)!}{(i-1)! i!} z^i \), therefore, after identification, \( S_n = \frac{(2n-2)!}{(n-1)! n!} \) holds which gives the value of \( \text{Cat}(n) \) (eg \( S_n \)).

### 6.3 Theoretical complexity

The complexity involves three parts, one related to constructing the list of the \( N_n \) candidate solutions, the second related to exhibit the number of different triangles for the generating polygon, \( Tr_n \), and, finally, the cost needed to validate such or such solution with respect to the purpose (reducibility, optimization or whatever).

\( N_n \), the number of candidate triangulations of a shell increases as an exponential in \( n \), thus the cost to exhibit these triangulations is \textit{a priori} non polynomial.

Therefore, optimizing a shell is non polynomial. However, if we consider only the triangulations star-shaped with respect to one of the vertices in the polygon, \( N_n \) becomes linear in \( n \).

Edge flips for reduction purpose has a non-polynomial cost. Indeed, the triangular mesh solution gives the order in which the “2-3” flips must be applied and this solution must be exhibited among the \( N_n \) cases. However, in the planar case, the cost is only quadratic, in fact, we can consider every three consecutive vertices in the polygon leading to a triangle which allows for a “2-3” flip and the resulting polygon is reduced by one, then, the same applies.

In the non-planar case, this simple procedure does not apply because it is not proved that reducing by one a reducible shell results in a shell which is still reducible. In fact, simple cases can be constructed where applying a flip may results in a non-flippable reduced shell (while being reducible before). This suggests defining an order when choosing a flip to maintain a reducible shell at each step until the final reduction.

### 6.4 Computer issues

Reducing the effective cost of a flip is achieved by a rapid rejection of as many \textit{a priori} candidate solutions as possible when evaluating the various cases. A simple idea allows for this. We just have to classify the candidate triangles as a function of their frequency. Figure 7. Therefore, a negative analyse of one (at most 2) tets related to one such triangle allows to immediately reject a number of cases. In the example in the figure, rejecting triangle \( t_1 \) in case \( i \) leads to reject case \( iv \) and thus triangle \( t_0 \) (and related tets) are never considered. Moreover, instead of considering case \( i \) and then the next case, triangles will be checked following the above classification. Clearly, the higher order the shell the higher benefit. In this way, analyzing all the possible cases is unlikely to be possible. Actually, only shell of order up to 6 are of real interest and the cost is negligible. In this case, the

### Table 1. Number of possible triangulations versus the number of sides of the generating polygon.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( 3 )</th>
<th>( 4 )</th>
<th>( 5 )</th>
<th>( 6 )</th>
<th>( 7 )</th>
<th>( 8 )</th>
<th>( 9 )</th>
<th>( 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_n )</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>14</td>
<td>42</td>
<td>132</td>
<td>429</td>
<td>1430</td>
</tr>
<tr>
<td>( Tr_n )</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>20</td>
<td>35</td>
<td>56</td>
<td>84</td>
<td>120</td>
</tr>
</tbody>
</table>
full list of the candidate solutions is made in advance (by hand) and thus is a null cost process (while being non polynomial in general).

7. OTHER FLIPS ?

As soon as the generating polygon is not a planar polygon and has at least 4 points, it is possible to construct tets with positive volume whose vertices are 4 of these points. The simpler example concerns a 4-tet shell.

If tet \([M_1 M_2 M_3 M_4]\) is positive, this shell can be possibly written as a polyhedron with 5 tets, \([M_1 M_2 M_3 M_4]\) \([M_1 M_2 M_4 \beta]\) \([M_2 M_3 M_4 \beta]\) and \([\alpha M_1 M_2 M_3]\) \([\alpha M_1 M_3 M_4]\).

If \([M_1 M_2 M_3 M_4]\) is negative, the writing is \([M_1 M_2 M_3 M_4]\) \([M_1 M_2 M_4 \beta]\) \([M_1 M_3 M_4 \beta]\) and \([\alpha M_1 M_2 M_3]\) \([\alpha M_1 M_3 M_4]\).

Is it a new flip ? No, it is not, to be convinced, consider the last remeshing and compare it to the second writing already seen (in Section about the “4-4” flip) : \([\alpha M_2 M_3 M_4]\) \([M_1 M_2 M_3 M_4 \beta]\) \([M_1 M_2 M_4 \beta]\) and \([\alpha M_1 M_2 M_3]\) \([\alpha M_1 M_2 M_3 M_4]\).

A “2-3” flip applied to \([M_1 M_2 M_4 \beta]\) and \([M_1 M_3 M_4 \beta]\) with the common face \(M_1 \beta M_4\) results in the 3 tets \([M_3 M_4 M_1 M_2]\) \([M_1 M_2 M_3 \beta]\) \([M_1 M_3 M_4 \beta]\).

Thus the 5-tet solution is obtained after applying a “2-3” flip (which is often possible as the 4 \(M_i\)’s are not planar) to 2 tets in the classical solution. So it is for \(n > 4\).

8. FACE FLIPS ?

Edge flips remove an edge, a question is then to decide if exists similar transformations which remove a face (which could be seen as a face removal operator).

An idea is to find such a transformation as the inverse of an edge flip. Indeed, the “2-3” flip , inverse to a “3-2” flip, seems to be an example of such a transformation and thus each edge flip, \(n > 3\), should have a corresponding face flip.

Actually, designing a face flip reduces to find 2 points \(\alpha \) and \(\beta\) and a polygon made up of faces that see these 2 points. This implies some properties about these faces.

Let \(\alpha\) be a vertex of tet \(K_1\) and let \(f_\alpha\) be its opposite face in this tet \((K_1 = [\alpha f_\alpha]\)) . Let \(K_2\) be the tet sharing face \(f_\alpha\) with \(K_1\). Point \(\beta\) opposite this face in \(K_2\) defines with \(\alpha\) segment \(\alpha \beta\). If \(\alpha \beta\) does not cut \(f_\alpha\), we return to a known case (2 tets sharing a face) where a “2-3” flip applies which removes face \(f_\alpha\) and construct edge \(\alpha \beta\). If segment \(\alpha \beta\) does not cut \(f_\alpha\), we find the tets cut by this segment. After some conditions we return to a pattern that can be seen as the inverse of an edge flip.

Above conditions reduces to one condition :

- either there is only one tet face cut by \(\alpha \beta\),
- or \(\alpha \beta\) cuts one edge, \(ab\), common to a number of tets whose other vertices are \(\alpha\) and \(\beta\).

As a consequence, the related polygon is either made up of the vertices of the cut face or made up of the vertices of the 2 faces sharing edge \(ab\).

In other words, all the other cases are not candidate to a flip thus reducing the field of applications of such an (face flip) operator. For completeness, however, it must be noticed that some peculiar meshes exhibit candidate cases.

To end, let us remark in the case of a “2-3” flip, in a convex case, that the polygon is a priori made up of the ordered list of the vertices of the common face but, after some conditions, it could be augmented by some neighbouring faces so as to arrive to a larger pattern, Figure 9. The aim is here to increase a quality
criterion, let us think to a case where $\alpha/\beta$ cuts triangle $M_1M_2M_3$ close to edge $M_2M_3$.

The envisioned application concerns the tedious boundary enforcement in a Delaunay based mesh generation method. It is known that such a process is mainly based on edge flips (at least in the approach we have proposed a number of years ago). The idea (which is so simple but took about 10 years to be matured !) trivially consists in governing such flips using a local and temporary defined metric based on the edge we like to create.

The above anisotropic metric (matrix) is then defined in accordance. Let us recall that $\mathcal{M}$, a general metric, reads as a $3 \times 3$ symmetric positive definite matrix which can be also written as $\mathcal{M} = \mathcal{R} \Lambda^t \mathcal{R}$, with $\mathcal{R}$ an orthogonal matrix and $\Lambda$ a diagonal matrix with positive entries.

Therefore, if $AB$ is the sought edge, we define the metric as

$$h = ||AB||$$

$$\Lambda = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{(k \cdot h)^2} & 0 \\ 0 & 0 & \frac{1}{(k \cdot h)^2} \end{pmatrix}, \quad (1)$$

$$d_1 = \frac{AB}{k} \text{ and } d_2 \text{ such that } <d_1,d_2> = 0 \text{ and } ||d_2|| = 1 \text{ and, finally, } d_3 = \frac{d_1 \wedge d_2}{||d_1 \wedge d_2||} \text{ with } k << 1.$$ From $d_1$, $d_2$ et $d_3$, we define

$$\mathcal{R} = \{d_1,d_2,d_3\}.$$ Thus $\mathcal{M}$ is well defined and enjoys the good properties.

Note that this definition allows to artificially make points $A$ and $B$ closer while the other points are, temporarily, made farther. Also the direction of the vector supporting edge $AB$ is favored.

In [9] (in french) we show how this trivial method is used and reduces the cost of the boundary enforcement step included in a Delaunay based mesh generation method.

10. APPLICATIONS

Edge flips allow for a variety of applications among which we select what follow.
10.1 Tet mesh optimization

Edge flips (together with node repositioning) is one of the tool used for tet mesh optimization purpose, [1]. All (internal) edge are considered as candidate for a flip.

Criterion is no longer the volume positiveness but a quality function. As already mentioned, a rapid rejection of the unlikely suitable solutions is crucial leading to a rather effective method with a low cost (in specific as compared with the cost of node repositioning). In this way, optimizing a large number of tets takes only a couple of seconds (0.75 sec for a mesh with 85 764 tets, 4.00 sec. for 430 033 tets in our computer implementation).

10.2 Tet mesh delaunisation

In this section, we turn to two a priori different questions. One could be “is it possible to replace the Delaunay kernel, [6], by just locally splitting the element within which falls the point under insertion and then applying a series of edge flips”. The other could be to see “if edge flips allow to make an arbitrary mesh a Delaunay mesh”.

Direct point insertion plus edge flips. Such a method perfectly runs in two dimensions. Inserting a point reduces to find the triangle(s) within which the point falls, split this triangle(s) into 3 (4) sub-triangles and apply a series of edge flips until the Delaunay criterion is locally satisfied. Is it the case in three dimensions? This problem has been discussed in [12] and [11] which consider the configurations of 5 distinct non-coplanar points. In [11], an algorithm using low order edge flips is proposed which assumes that the point insertion follows a peculiar order: the current point to be inserted must be outside the convex hull of the already inserted points. In the general case, this problem appears to be still open. However we think that this problem can be translated in another one which says that some point to point connections are missing while some others are to be deleted. The idea could be to remove the extra connections while re-creating the missing ones using flips. The key would be to prove that flips never lead to a non Delaunay configuration which is no longer “flippable”.

Delaunisation of an arbitrary mesh. We are given an arbitrary mesh (eg non Delaunay) and we like to apply a number of edge flips so as to arrive to a Delaunay mesh

This is known in two dimensions for a triangulation and it is also true for a mesh not for the Delaunay criterion but for a constrained variant of this property. Is it the case in three dimensions?

In [10] is given a 3D example for which using “2-3”, “3-2” and “4-4” flips to fulfill the Delaunay criterion does not complete a Delaunay triangulation. The best we can do is to conjecture that applying such flips even with a Delaunay criterion violation result in a Delaunay triangulation.

The same question for a meshing problem is much more tedious since constrained entities (must) exist. In this case, it is not safe to formulate any conjecture.

In other words, this question seems to be still open.

10.3 Boundary enforcement in a Delaunay based mesh

As partially evocated, a natural use of edge flips is to remove edges (and faces) in a tet mesh and to create alternate edges and faces. This is the key point (while being not sufficient) in the method we proposed in the mesh generator developed at INRIA, [5], [7].

10.4 Anisotropic meshing

In this application, we are given a classical (thus isotropic) tet mesh and we like to introduce some degree of anisotropy in some regions, [13]. The “2-3” flip appears to be attractive to handle such a problem. Let us consider 2 adjacent tets where the common face has a nice quality (following an isotropic quality function). Clearly, in this pattern (common face $M_1 M_2 M_3$ and opposite points $\alpha$ and $\beta$), the distance between $\alpha$ and $\beta$ is larger than the other distances from point to point. Slipping the common face and constructing edge $\alpha\beta$ reduces to artificially make those points closer while the other are put farthest, [9], and, actually, this operation introduce some degree of anisotropy in the mesh. Indeed, such a flip can be seen as an anisotropic optimization, thus a way to optimize a mesh with respect to an anisotropic metric.

We have then in hand a simple and low cost method which introduces some anisotropy in a given mesh. Nevertheless, the sole use of “2-3” flips results in constructing 3-tet shells which, as well known, are under-connected (there are not enough tets around an edge). Therefore, this sole operator is not fully satisfactory and higher order flips must be envisaged.

Notice, to end the discussion, that constructing anisotropic meshes in this way is an alternate solution to an anisotropic metric.

\footnote{We assume the reader familiar with the difference between a triangulation problem and a meshing problem.}
to a direct method (see [8], in french, for such a direct approach).

11. CONCLUSION AND FUTURE WORK

We demonstrated that the natural extension of the “2-2” flip in two dimensions is the edge flips discussed in this paper. We showed it is nothing other than a combination of elementary flips. Various properties of this operator were discussed, including conditions that make a flip possible (thus making a shell of tets reducible). We considered also complexity issues for different purposes (reducibility, optimization, ...). We discussed also about the existence of other edge based operators and a number of applications were envisaged including exotic uses of some degree of anisotropy.

Future works may include computer implementation of anisotropic edge flips (as needed in a general anisotropic mesh generation method), also, a number of applications can be envisaged (as a perspicacious reader can easily imagine !).

REFERENCES


INCREASING THE NUMBER AND VOLUME OF HEXAHEDRAL AND PRISM ELEMENTS IN A HEX-DOMINANT MESH BY TOPOLOGICAL TRANSFORMATIONS

Soji Yamakawa\textsuperscript{1} and Kenji Shimada\textsuperscript{2}

The Department of Mechanical Engineering, Carnegie Mellon University,
\textsuperscript{1}soji@andrew.cmu.edu
\textsuperscript{2}shimada@cmu.edu

ABSTRACT

This paper describes a new method for increasing the number and the volume of hexahedral and prism elements in a hex-dominant mesh by topological transformations. The method takes as input a hex-dominant mesh consisting of hexahedrons, prisms, pyramids and tetrahedrons and modifies the mesh to increase the number and the volume of hexahedrons and prisms while maintaining the relaxed conformity criteria, which allows a connection from two tetrahedrons to a quadrilateral face of a hexahedron or a prism. If a hex-dominant mesh satisfies the relaxed conformity criteria, it can be used in the finite element analysis by applying an error reduction scheme on non-conforming faces \cite{1-3}, inserting pyramids on non-conforming faces \cite{4}, or converting the mesh to an all-hex mesh by a template method \cite{5, 6}. With more hexahedrons and prisms in a hex-dominant mesh, a more accurate finite element solution can be obtained in a shorter time. Hence the proposed method increases the practical value of a hex-dominant mesh. Several experiments showed the number of hexahedrons increased by about 10\% to 20\%, yielding hex-dominant meshes with 70\% to 90\% hexahedron volume ratio.

Keywords: hex-dominant mesh, topological transformation

1. INTRODUCTION

This paper describes a new method for increasing the number and the volume of hexahedral elements in a hex-dominant mesh by transforming elements. In general, a hex-dominant mesh yields a more accurate finite element solution with a shorter computational time than a tetrahedral mesh with the same degrees of freedom, and is easier to create automatically than an all-hex mesh. However, even with an equal degrees of freedom, the accuracy and computational time of the finite element analysis depends on the ratio of the number and the volume of hexahedrons included in the mesh; increasing the number of hexahedrons increases the accuracy of the solution and decreases the computational time. Hence, it is important to create a hex-dominant mesh with as many hexahedrons as possible and with as much volume as possible being hexahedrons. Or, if possible, a post-process for a hex-dominant mesh would be useful to increase the number and volume of hexahedrons. (Please note that applying subdivision patterns used in \cite{7-9} increases the number of hexahedrons, but it does not increase the total volume filled by hexahedrons.)

The proposed method takes as input a hex-dominant mesh consisting of hexahedrons, prisms, pyramids and tetrahedrons, and increases the number and the volume of hexahedrons and prisms while maintaining the relaxed conformity criteria, which allows a connection from two tetrahedrons to a quadrilateral face of a hexahedron or a prism. The details of the relaxed conformity criteria are discussed in Section 3. The method first subdivides prisms and pyramids included in the input mesh into tetrahedrons. Then, the method applies sequences of topological transformations (explained in Section 4) to increase the number of hexahedrons. Some of the remaining tetrahedrons are then merged and converted to prisms \cite{10}. If the mesh must conform, pyramid elements can be
inserted between tetrahedrons and a quadrilateral face of a hexahedron or a prism via the method presented by Owen et al. [4].

The organization of the paper is as follows. Section 2 reviews previous work of hex-dominant mesh generation. Section 3 discusses the relaxed conformity criteria and gives a definition of a hex-dominant mesh which is dealt with in the proposed method. Section 4 describes three types of topological transformations that modifies some elements while maintaining the relaxed conformity criteria. Section 5 presents a strategy for applying the transformations to increase the number and volume of hexahedrons. Section 6 gives some results of the method, followed by discussions in Section 7 and conclusions in Section 8.

2. RELATED RESEARCH

A few hex-dominant mesh generation techniques have been published. Meyers et al. [11] and Tuchinsky and Clark [12] present a method that creates a hex-dominant mesh by expanding the plastering method, which is presented by Blacker and Meyers [13]. Their method creates hexahedrons from the boundary inward by the plastering method, and if the plastering method cannot fill entire volume with hexahedrons, their method fills the remaining volume with tetrahedrons.

Owen and Saigal present an algorithm called H-Morph [14], which converts a tetrahedral mesh to a hex-dominant mesh by creating hex elements one by one starting from domain boundaries and moving inward. The method always maintains a valid hexahedron-tetrahedron mixed mesh during the process.

Meshkat and Talmor present an algorithm that converts a tetrahedral mesh into a hex-dominant mesh based on the graph theory [15]. Their method takes a tetrahedral mesh as input and creates a graph that represents the topology of the tetrahedral mesh. Their method then searches for a pattern that can be converted to a hex or a prism in the graph, and when a pattern is found, a new hex or a new prism is created, and the graph is updated accordingly.

Yamakawa and Shimada present a method that first creates good node locations for a hex-dominant mesh by packing rectangular solid cells in the target geometric domain. A tetrahedral mesh is then created using the node obtained by the packing. Finally, the tetrahedral mesh is converted to a hex-dominant mesh by merging some tetrahedrons [10].

There is a highly effective post-process for a tetrahedral mesh called local transformation [16], which improves the quality of a tetrahedral mesh, though, no such method has been published for a hex-dominant mesh. Bern and Eppstein present one method of transforming hexahedral elements [17]. However, it is unclear if their method can improve the quality of an all-hex mesh or a hex-dominant mesh.

3. RELAXED CONFORMITY CRITERIA

The topological transformations described in Section 4 deal with a hex-dominant mesh that satisfies the relaxed conformity criteria. The criteria allows a connection between two tetrahedrons and a hexahedron or a prism through a quadrilateral face. The purpose of the relaxed conformity criteria is to increase the applicability of the mesh to the finite element analysis and relieve the burden of hex-dominant mesh generation. The traditional conformity criterion demands that an interface between two elements be identical and that the interface must not be shared by a third element. A tetrahedron thus cannot be directly connected to a hexahedron by a face because a hexahedron has only quadrilateral faces, and a tetrahedron has only triangular faces; this condition makes a hex-dominant mesh very difficult to create. To relieve the burden of hex-dominant mesh generation, the relaxed conformity criteria is introduced, and it allows connection between two triangular faces of two tetrahedrons to a quadrilateral face of a hexahedron by sharing the four nodes. Such a quadrilateral face connected to two triangular faces is called a non-conforming face, or more explicitly, a non-conforming quadrilateral.

Although the relaxed conformity criteria has been implicitly used [1-3, 12, 15], a clear definition of the relaxed conformity criteria has not been given; here we propose a definition of the relaxed conformity conditions.

Condition 1. For a non-conforming quadrilateral, there must be exactly two tetrahedrons that each has only one triangle sharing three of the four nodes of the non-conforming quadrilateral. The quadrilateral obtained by merging the two triangles must be equal to the non-conforming quadrilateral.

Condition 2. A diagonal of a non-conforming quadrilateral must not be an edge of another quadrilateral.

Condition 3. If two quadrilaterals share only two nodes, the two quadrilaterals must share an edge (i.e., sharing only a diagonal is not permissible.)

Condition 4. Two quadrilaterals must not share only three nodes.

For example, the case illustrated in Figure 1 (a) violates Conditions 1 and 4 because the left hexahedron is connected to the right hexahedron and the tetrahedron between the two hexahedrons through a quadrilateral face, and the quadrilateral of the right face of the left hexahedron and a quadrilateral of the left face of the right hexahedron shares only three nodes. The case illustrated in Figure 1 (b) violates Condition 1 because only one tetrahedron is connected to the right-hand side quadrilateral of the hexahedron, and half of the quadrilateral is exposed to the exterior of the mesh. The case illustrated in Figure 1 (c) violates Condition 1 because the right-hand side quadrilateral of the top hexahedron is connected to three tetrahedrons, and the right-hand side quadrilateral of the bottom hexahedron is connected to four tetrahedrons. The case illustrated in Figure 1 (d) violates Conditions 2 and 3...
because an edge of a quadrilateral is lying on the diagonal of another quadrilateral. The case illustrated in Figure 1 (e) violates Condition 3 because the right-hand side quadrilateral of the left hexahedron and the left-hand side quadrilateral of the right hexahedron share two nodes, but do not share an edge.

(a) Violating conditions 1 and 4
(b) Violating condition 1
(c) Violating 1
(d) Violating 2 and 3
(e) Violating 3

**Figure 1** Examples of violation of the relaxed conformity conditions

If a hex-dominant mesh satisfies the four relaxed conformity conditions, it can be used in the finite element analysis by:

Method 1. Inserting a pyramid on each non-conforming quadrilateral to recover perfect conformity via the method presented by Owen et al. [4].

Method 2. Applying an error reduction scheme, such as MPCs [1, 2] or Dohrmann et al.’s [3], or

Method 3. Converting it to an all-hex mesh by applying conversion templates, such as HEXHOOP templates [5] or Geode template [6]. (Geode template requires another condition in addition to the relaxed conformity criteria to be applicable; all non-conforming quadrilaterals must form a topological ball or terminate at the boundary. Nonetheless, satisfying the relaxed conformity criteria makes Geode template more likely to be applicable.)

Applicability of the above methods also depends on a finite element solver. Method 1 can be applied when a solver can take pyramid elements. Method 2 can be applied when a solver is capable of imposing an error reduction scheme. Method 3 can be applied when a solver is not too restrictive about the quality of the all-hex mesh converted from a hex-dominant mesh. Nonetheless, satisfying the relaxed conformity criteria makes a hex-dominant mesh more likely to be applicable to the finite element analysis.

The next section shows three types of topological transformations for a hex-dominant mesh that transform elements while maintaining the relaxed conformity criteria.

### 4. TOPOLOGICAL TRANSFORMATIONS FOR A HEX-DOMINANT MESH

This section explains three types of topological transformations developed in our research: (1) edge-collapse, (2) node insertion, and (3) shear. Each transformation is performed while maintaining the relaxed conformity criteria defined in Section 3. Applying these transformations systematically increases the number and volume of hexahedrons in a hex-dominant mesh. The strategy of applying these transformations is explained later in Section 5.

#### 4.1. Edge-collapse transformation

The edge-collapse transformation collapses an edge that is used only by tetrahedrons. All tetrahedrons using the edge are deleted and the two nodes of the edge are joined into one node. If the two nodes of the edge are not used by any hexahedrons, the transformation can be easily accomplished by merging two nodes into one and deleting all degenerating tetrahedrons. However, if one of the nodes of the edge is used by a hexahedron, the relaxed conformity criteria may be violated as a result of the transformation, and thus the program must check for the violation.

When the relaxed conformity criteria are violated as a result of the edge-collapse transformation, the violation may be resolved by applying a subsequent edge-collapse transformation. Figure 2 shows an example of such resolution. Figure 2 (a) shows a portion of a mesh, two hexahedrons and three tetrahedrons between them. When edge $CF$ is collapsed, the relaxed conformity criteria are violated because quadrilaterals $ABC'D$ and $ADC'E$ share only three nodes as shown in Figure 2 (b). To resolve the violation, edge $BE$ must be collapsed as shown in Figure 2 (c). The second edge-collapse transformation may yield another violation of the relaxed conformity criteria, and if so, the method attempts to resolve the violation by applying an additional edge-collapse transformation. Thus, one edge-collapse transformation may yield a series of the edge-collapse transformations, which continue until all violations are resolved. However, the violation is not always resolved by applying a series of the transformations, and if a violation cannot be resolved in the end, the series of the transformations must be cancelled.
4.2. Node insertion

The node insertion transformation adds a node on an edge and subdivides elements using the edge. If a target edge (an edge on which a node is inserted) is used only by tetrahedrons, each tetrahedron using the edge is simply subdivided into two tetrahedrons by a triangle formed by the node inserted on the edge and the two nodes of the original tetrahedron that are not connected to the edge.

Inserting a node on an edge used by a hexahedron, however, is not easily accomplished because four subdivision patterns are possible; the appropriate pattern must be chosen for a configuration around the hexahedron, or the transformation cannot be applied if none of the patterns fits the configuration. As shown in Figure 3, when node X is inserted on edge FG of hexahedron ABCDEFGH, the hexahedron can split into four possible subdivision patterns:

1. one hexahedron ABCDEFXH and one tetrahedron CHGX as shown in Figure 3 (a),
2. one hexahedron ABDEFXH and two tetrahedrons DHGX and CDGX as shown in Figure 3 (b),
3. one hexahedron ABCDExGH and one tetrahedron BEXF as shown in Figure 3 (c), and
4. one hexahedron ABCDExGH and two tetrahedrons AEXF and BAXF as shown in Figure 3 (d).

To distinguish these patterns, a subdivision pattern is denoted as $PQ-R_n$, which means a node is inserted on edge PQ and R becomes a node of a new tetrahedron (where R is equal to either P or Q), and n new tetrahedrons are created by the node insertion. Subdivision pattern (1) is thus denoted as FG-G1, pattern (2) FG-G2, pattern (3) FG-F1 and pattern (4) FG-F2.

To apply pattern $PQ-R_n$, the face of the hexahedron using node R and not using edge PQ must be either a non-conforming quadrilateral or a quadrilateral exposed to the exterior of the mesh. For example, if pattern EH-H1 or EH-H2 is applied to the mesh shown in Figure 4, quadrilaterals DCGX and DCGH will share only three nodes after the node insertion, and condition (4) of the relaxed conformity criteria will be violated.

In addition, to apply pattern $PQ-R_n$, if a quadrilateral face of the hexahedron using edge PQ is a non-conforming quadrilateral, one of the edges of the two tetrahedrons connected to the quadrilateral face lying on a diagonal of the quadrilateral face must not use node R. For example, when two tetrahedrons ADBT and DCBT are connected to face ABCD as shown in Figure 5, applying pattern AB-Bn yields a violation of condition (1) of the relaxed conformity criteria because a non-conforming quadrilateral AXCD will not be connected to exactly two tetrahedrons by faces.

If all hexahedrons using edge PQ satisfy the above conditions for pattern $PQ-R_n$ ($n$ can vary across the hexahedrons), a node can be inserted on edge PQ. Note that yet the appropriate $n$ must be chosen. If the quadrilateral face using node R and not using edge PQ is a non-conforming quadrilateral, one of the edges of the two tetrahedrons connected to the quadrilateral face must be lying on a diagonal of the quadrilateral face. If the edge of the tetrahedrons lying on a diagonal of the quadrilateral face is using node R, $n$ must be 2 as shown in Figure 6 (a). Otherwise, $n$ must be 1 as shown in Figure 6 (b).
4.3. Shear transformation

The shear transformation shears a hexahedron by reconnecting a hexahedron and tetrahedrons sharing an edge and/or a diagonal with one of the six faces of the hexahedron. Suppose some tetrahedrons sharing an edge and/or a diagonal of a face of a hexahedron can be merged and converted to a prism. For example, tetrahedrons JFEI, IEDF and EADF shown in Figure 7 (a) are sharing an edge and/or a diagonal of quadrilateral IFAD of hexahedron ABCDFGHI and can be merged and converted to a prism. Now the hexahedron can be split into two prisms by a quadrilateral formed by connecting two edges: (1) an edge connecting the two triangles of the pseudo-prism and being shared by the pseudo-prism and the hexahedron, and (2) the opposite edge of edge (1) in the hexahedron. In Figure 7 (a), edge (1) can be either ID or FA, and the quadrilateral can be IDBG or FACH respectively. One of the two prisms that came from a hexahedron shares a quadrilateral face with the pseudo-prism and can be merged with the pseudo-prism to become a hexahedron. The other prism must be split into tetrahedrons while maintaining the relaxed conformity criteria. If quadrilateral IDBG is chosen to split the original hexahedron ABCDFGHI in Figure 7 (a), three tetrahedrons DBCG, DGCH, DGII, and a hexahedron ABDEFGIJ are created as shown in Figure 7 (b).

The shear transformation splits two quadrilateral faces of a hexahedron into triangles by adding a diagonal edge. In Figure 7 (a), quadrilaterals HIDC and CBGH are split. There are two possible tessellations for each quadrilateral. However, when such a quadrilateral is a non-conforming quadrilateral, only one of the two tessellations can conform two tetrahedrons connected to the quadrilateral. In such a case, an appropriate tessellation must be chosen, and three new tetrahedrons must conform to such a tessellation.

One shear transformation may yield subsequent shear transformations when a sheared quadrilateral face of the hexahedron is connected to another hexahedron. For example, if the shear transformation is applied to hexahedron DABCIFGH of Figure 8 (a) and transformed into hexahedron ABDEFGIJ as shown in Figure 8 (b), quadrilaterals JIGF and IHGF violate the relaxed conformity criteria because the two quadrilaterals share only three nodes. To resolve this violation, the shear transformation must be applied to the adjacent hexahedron connected to quadrilateral IHGF as shown in Figure 8 (c). Furthermore, the second shear transformation may yield the third shear transformation, and the series of transformations continues until all violations are resolved, or the series of transformations must be cancelled if a violation cannot finally be resolved.
5. INCREASING HEXAHEDRONS VIA TOPOLOGICAL TRANSFORMATIONS

This section explains three typical patterns in which the number and the volume of hexahedrons are increased by applying the topological transformations explained in Section 4. The pattern shown in Figure 9 (a) can be solved either by collapsing edge HI or inserting a node on edge AD. Figure 9 (a) shows a portion of a hex-dominant mesh consisting of a hexahedron on the right and tetrahedrons filling the remainder of the portion. If tetrahedrons left of quadrilateral CDHG in Figure 9 (a), are converted to hexahedron ABCDEFGH, quadrilaterals CDHG and CDIG share only three nodes violating the relaxed conformity criteria. If node X is inserted on edge AD as shown in Figure 9 (b), tetrahedrons left of quadrilateral XCGH can be converted to hexahedron ABCXEFGH, and tetrahedrons filling between hexahedrons ABCXEFGH and DCKIJGML can be converted to a prism. Another resolution for this case is to collapse edge HI. If edge HI is collapsed into node Y as shown in Figure 9 (c), tetrahedrons left of DCGY can be converted to hexahedron ABCDEFGY.

Figure 10 (a) shows a pattern in which some tetrahedrons are trapped between two hexahedrons on the left and right. An extra node is needed to convert the tetrahedrons to a hexahedron, or a node must be deleted to convert tetrahedrons to a prism. Since the volume between the two hexahedrons is enclosed by four quadrilaterals and two triangles, we call it a 4Q2T pattern. The 4Q2T pattern shown in Figure 10 (a) can be resolved by inserting a node on either edge JK as shown in Figure 10 (b) or LJ as shown in Figure 10 (c). If a node is inserted on edge JK or LJ, tetrahedrons between the two hexahedrons will be converted to a hexahedron. Another resolution is achieved by collapsing edge BC as shown in Figure 10 (d) or MN as shown in Figure 10 (e). If edge BC or MN is collapsed, tetrahedrons between the two hexahedrons will be converted to a prism.
Figure 10  Resolving a 4Q2T pattern (not all internal edges are drawn)

Figure 11 shows a pattern that can be resolved by the shear transformation. Figure 11 shows three rows of elements, the top row has two hexahedrons on the left and right, and the mid row has a hexahedron on the left, and the bottom row has two hexahedrons on the left, and the remaining portion is filled with tetrahedrons. If tetrahedrons left of GIDB in the mid row are converted to hexahedron AEDBFJIG, quadrilateral FJIG shares only three nodes with the bottom face of the right hexahedron in the top row, and if tetrahedrons right of GJEB in the mid row are converted to hexahedron BEDCGJIIH, quadrilateral BEDC shares only three nodes with the top quadrilateral of the right hexahedron in the bottom row; either case creates a violation of the relaxed conformity criteria.

If the right hexahedron in the top row is sheared so that the top row pattern matches the bottom row, tetrahedrons left of GIDB in the middle row can be converted to hexahedron AEDBFJIG, and the remaining tetrahedrons will be converted to prisms as shown in Figure 11 (b). Or if the right hexahedron in the bottom row is sheared so that the bottom row pattern matches the top row, tetrahedrons right of GJEB in the middle row can be converted to hexahedron BEDCGJIIH and remaining tetrahedrons will be converted to prisms as shown in Figure 11 (c).

A pattern sometimes appears as a combination of the above basic patterns. A sequence of transformations can often resolve such patterns.

6. RESULTS

Figure 12, Figure 13, and Figure 14 show experimental results of the method. For each example, an input hex-dominant mesh is created by the method presented in [10]. Each example includes screenshots of hex-dominant meshes before and after applying the transformations, remaining tetrahedrons before and after applying the transformations, statistics before and after applying the transformations.

In these example cases, the percentage of the number and volume of hexahedrons increases, and the number and volume of tetrahedrons is reduced by the transformations. The percentage of the number of hexahedrons increases from 30% to 50% in the hex-dominant mesh shown in Figure 12, from 32% to 46% in the hex-dominant mesh shown in Figure 13, from 25% to 34% in the hex-dominant mesh shown in Figure 14, and from 33% to 43% in the hex-dominant mesh shown in Figure 15. The percentage of the volume of hexahedrons increases from 67% to 82% in the hex-dominant mesh shown in Figure 12, from 70% to 80% in the hex-dominant mesh shown in Figure 13, from 66% to 76% in the hex-dominant mesh shown in Figure 14, and from 71% to 80% in the hex-dominant mesh shown in Figure 15.
These results show that the method effectively increases the number and volume of hexahedrons, and decreases the number and volume of tetrahedrons.

7. DISCUSSIONS

The three topological transformations described in Section 4 deal only with hexahedrons and tetrahedrons. Hence, when the input hex-dominant mesh includes prisms and pyramids, all of them must be subdivided into tetrahedrons. Or, the three transformations cannot be performed near a prism or a pyramid.

The transformations deal only with hexahedrons and tetrahedrons because there are too many configurations of neighboring elements if pyramids and prisms are included. There are many variations of a configuration of neighboring elements even with only hexahedrons and tetrahedrons. The transformations would become too complex due to an overwhelmingly large number of variations of neighboring element configuration, if prisms and pyramids are considered. Thus it was deemed reasonable to limit transformations for hexahedrons and tetrahedrons, and subdivide prisms and pyramids in the input mesh into tetrahedrons first, and re-create them after the transformations are performed.

The result of the proposed method depends on the order of the transformations. Some transformations performed earlier may interfere with further transformations, and fewer hexahedrons can be created as a result of this interference. In the current implementation, the program searches and resolves the patterns shown in Figure 9 and Figure 11 first. It then searches and resolves the patterns shown in Figure 10. It iterates the search and resolution until no such pattern is found. However, it is very difficult to find the optimal order of the transformations, and thus it becomes a subject for future research.

The effectiveness of the method also depends on an input hex-dominant mesh. However, we could only test the method with a hex-dominant mesh created by the method presented in [10]. The proposed method works well when there are some islands of tetrahedrons, which gives some freedom for the transformations. Therefore, we expect that the proposed method will also work well for a hex-dominant mesh created by Meshkat and Talmor’s method [15]. However, the proposed method may not perform well for advancing front type methods [11, 12, 14] because such methods tend to create less islands of tetrahedrons and the remaining tetrahedrons tend to be flatter. The relation between the effectiveness of the proposed method and hex-dominant mesh generation algorithms is another issue for future research.

Also, when there is more than one option for resolving a case, the program must decide which option to choose. And, one choice may interfere with further transformations yielding fewer hexahedrons. The current implementation chooses the option that gives better quality elements measured by the scaled Jacobian [18, 19]. However, choosing a less but acceptable quality element may yield more hexahedrons. Choosing an option to resolve a case also becomes an issue for future research.

In addition to the three transformations described in Section 4, there can be more possible transformations. There may be a better strategy of applying the transformations than the strategy described in Section 5. An ultimate goal of this method is to convert the input hex-dominant mesh into a fully conformed hex-dominant mesh without using pyramids while filling most of the volume with hexahedrons. However, it is unclear if such transformations and strategies are possible, and thus it is another issue for future research.

8. CONCLUSIONS

This paper has presented a new method for increasing the number and volume of hexahedrons in a hex-dominant mesh by applying topological transformations. Three types of transformations are presented, and by applying them strategically the number and volume of hexahedrons in the hex-dominant mesh increase. Some experimental results show the method works effectively, and it increases the practical value of a hex-dominant mesh. The experimental results show the number of hexahedrons increased by about 10% to 20%, yielding hex-dominant meshes with a 70 to 90% hexahedral volume ratio.

REFERENCES


Number of nodes 2,375
Number of Elements 3,308
Number of Hexes 1,008(30%)
Number of Prisms 402(12%)
Number of Tets 1,898(57%)
Total Volume 3,337.7
Hex Volume 2,243.8(67%)
Prism Volume 422.8(13%)
Tet Volume 671.2(20%)

Number of nodes 2,336
Number of Elements 2,367
Number of Hexes 1,193(50%)
Number of Prisms 302(12%)
Number of Tets 872(36%)
Total Volume 3,333.4
Hex Volume 2,734.1(82%)
Prism Volume 302.1(9%)
Tet Volume 297.2(9%)

Figure 12 Hex-dominant mesh of a mechanical part
Number of nodes 10,462
Number of Elements 10,086
Number of Hexes 3,263(32%)
Number of Prisms 1,459(14%)
Number of Tets 5,364(53%)
Total Volume 5,843.8
Hex Volume 4,067.4(70%)
Prism Volume 846.0(14%)
Tet Volume 930.4(16%)

(a) The hex-dominant mesh before applying transformations
(b) Remaining tetrahedrons before applying transformations
(c) Statistics before applying transformations

Number of nodes 10,221
Number of Elements 7,979
Number of Hexes 3,684 (46%)
Number of Prisms 1,028 (12%)
Number of Tets 3,267 (40%)
Total Volume 5,818.5
Hex Volume 4,682.8(80%)
Prism Volume 596.8(10%)
Tet Volume 538.9( 9%)

(d) The hex-dominant mesh after applying transformations
(e) Remaining tetrahedrons after applying transformations
(f) Statistics after applying transformations

Figure 13 Hex-dominant mesh of a cell-phone case

Number of nodes 4,918
Number of Elements 5,382
Number of Hexes 1,374(25%)
Number of Prisms 695(12%)
Number of Tets 3,313(61%)
Total Volume 12,263.8
Hex Volume 8,117.9(66%)
Prism Volume 1,821.2(15%)
Tet Volume 2,324.8(19%)

(a) The hex-dominant mesh before applying transformations
(b) Remaining tetrahedrons before applying transformations
(c) Statistics before applying transformations

Number of nodes 4,849
Number of Elements 4,530
Number of Hexes 1,576(34%)
Number of Prisms 502(11%)
Number of Tets 2,452(54%)
Total Volume 12,240.5
Hex Volume 9,363.1(76%)
Prism Volume 1,316.4(11%)
Tet Volume 1,561.0(13%)

(d) The hex-dominant mesh after applying transformations
(e) Remaining tetrahedrons after applying transformations
(f) Statistics after applying transformations

Figure 14 Hex-dominant mesh of a PDA case
Figure 15  Hex-dominant mesh of a blood vessel for analysis of aneurysm: Due to varying cross-section, it is difficult to create a mesh of this geometry with a conventional method.
ABSTRACT

Topology modification of hexahedral meshes has been considered difficult due to the propagation of topological modifications non-locally. We address this problem by working in the dual of a hexahedral mesh. We prove several relatively simple combinatorial aspects of hex mesh duals, namely that they are both complexes of simple polytopes as well as simple arrangements of pseudo-hyperplanes. We describe a set of four atomic dual-based hex topology modifications, from which the flipping operations of Bern et. al can be constructed. We also observe several intriguing arrangements and modification operations, which we intend to explore further in the future.

Keywords: hexahedral mesh generation; arrangements; dual; simple polytope.

1. INTRODUCTION

Many finite element analysis practitioners prefer using all-hexahedral meshes. These meshes are believed to yield more accurate solutions for a given computational expense, especially in the non-linear analysis regime (theoretical studies investigating this issue are appearing now [1], joining the more empirical studies on the subject [2][3]). Generating all-hexahedral meshes suitable for finite element analysis remains an active area of research[4]. Working with hexahedral meshes has been considered difficult in part because of the non-local nature of connectivity modifications inside hex meshes. That is, until recently there were relatively few known options for modifying the topology interior to a hexahedral mesh which did not propagate through the mesh. This is in contrast to tetrahedral meshes, where local connectivity modifications are a crucial part of most robust tetrahedral mesh generation algorithms [5]. Furthermore, local connectivity modifications have also played an important role in all-quadrilateral meshing [6][7]. It is reasonable to expect that post-meshing topology modifications will play an important role in any successful automatic hexahedral meshing algorithm. In addition to mesh generation, local connectivity modification in hexahedral meshes could also be useful for adaptive refinement and for local mesh quality improvement, analogous to its counterpart in tetrahedral meshes. For all these reasons, we believe local connectivity modification in hexahedral meshes to be an important technology.

1.1. Hexahedral Dual

We study hex topology modification in the dual, for reasons which will become apparent later in this paper. The dual of a hex mesh is analogous locally to Voronoi diagrams for tetrahedral meshes, where (in three dimensions) each primal entity of dimension $k$ (e.g. node, edge, face) has a corresponding $(3-k)$-cell (e.g. 3-cell, 2-cell, 1-cell). Non-locally, the dual of a hex mesh has special properties: 1-cells and 2-cells can be grouped into larger structures which have non-local extent in the mesh. Indeed, the dual of a hexahedral mesh can be viewed as an arrangement of surfaces, with $(3-k)$-faces in the arrangement corresponding to $k$-dimensional entities in the primal mesh. Although this characteristic was recognized quite some time ago[8], its application to hexahedral meshes was not recognized until much later[9]. The physical interpretation of these surfaces is of topologically 2d layers of hexahedral elements; pairwise intersections of these surfaces represent columns of hex elements in the mesh. Both these structures are typically non-local in the mesh, and are sometimes self-intersecting. These structures are referred to as sheets and chords, respectively, and have been used in the development of the Whisker Weaving dual-based hex meshing algorithms[11][12].

One useful way of representing dual sheets (from here on referred to as sheets) is by viewing their 2d projection, where...
intersections with other sheets are depicted by lines (pairwise intersection) and vertices (three-way intersection). These “sheet diagrams” simplify the study of the dual arrangement by representing it as a series of 2d projections[11]. An example of a simple hex mesh, its dual, and its representation as a series of sheet diagrams is shown in (fig). The outer loop of each sheet represents the sheet’s intersection with the outer boundary of the mesh. The lines of intersection with other sheets we refer to as chords; on the sheet diagrams intersections with other chords represent hex elements, and intersections with the outer loop represent the emergence of a chord at a face on the boundary of the mesh. Loop-intersecting and interior vertices on sheet diagrams are sometimes labeled with the face/hex to which they correspond, respectively.

![Figure 1: Hex mesh with three elements (top); dual surfaces (sheets) and dual vertices shown. Two-dimensional projection into "sheet diagrams" shown below.](image)

One can see that:
- Each chord appears on the two sheets whose intersection form the chord
- A hex element appears as a vertex on three sheet diagrams as the pairwise intersection of three chords
- Each 1-cell (or segment of a chord) and 2-cell in the sheet diagrams represents a face and edge in the primal mesh, respectively

Sheet diagrams, and the 1-cells and 2-cells on them, are used later in this paper to depict hex topology modifications.

1.2. Prior Work

Relatively little work has been done in the area of hex mesh topology modification. This subject was treated peripherally in the development of Whisker Weaving [13], and in the study of “knife” element resolution [14]. While some of the structures being used in the current work were evident, no effort was made to utilize them in local mesh improvement aside from the constructive process. Knupp & Mitchell studied hex mesh topology optimization in connection with hex meshes arising from tetrahedral mesh subdivision [15]. The topology modification operations described there were similar to some described in this paper, but the completeness of these operations and their unification under a common theoretical framework was not addressed. Bern et. al research topology modifications of hex meshes in connection with “flips” of hex meshes, where it is shown that there exists a complete set of hex mesh transitions, each of whose outer boundary corresponds to the six faces of a cube [17]. More recently, this research is being extended to consider face collapsing inside the mesh [18]. This work has played an important role in our research, both as inspiration for starting the work as well as serving as a benchmark in various ways. We describe these links later in this paper. As mentioned before, the general approach of using local mesh improvement in a post-meshing context has been used in both the tetrahedral [5] and quadrilateral [6][7] meshing areas. While we believe the various cleanup methods demonstrated for quads could be unified under a common framework similar to the current work, no attempt has been made to do that since this seems to be treated adequately by prior methods.

1.3. Current Work

In this paper we outline the basic operations of and motivations for locally modifying the topology of an existing hexahedral mesh. Our initial goal has been to define a complete set of local, atomic operations on the topology of a hexahedral mesh. By “complete” we mean that we can transform a given hexahedral mesh to any other with the same quadrilateral boundary using only these operations. By “local” we mean that the connectivity altered by the sequence of operations is local to the actual hexes targeted by the operations. This boundary is denoted in the following sections by a dotted line which replaces the outer boundary of a sheet diagram, i.e. it is the group of quadrilateral faces bounding the region of interest, just as the outer boundary of a normal sheet diagram is a group of quadrilaterals on the outer boundary of the solid. Finally, the use of “atomic” above is intended to indicate that the operations we seek to define are in some sense irreducible, that is, cannot be done as a sequence of other atomic operations. In particular, we show that the flipping operations presented in [18] can be reproduced as a series of these elementary operations. The contribution of this work is the reduction of other known hex mesh topology modification operations into a much smaller set of atomic operations. This work also unifies these operations under a common theoretical framework of modifications to a simple arrangement of surfaces, while showing how these operations can be directed toward specific mesh improvement or mesh modification for other purposes. The remainder of this paper is arranged as follows. Section 2 reviews elements of the dual of a hexahedral mesh, and proves several new results for those types of duals. Section 3 describes the general approach we take to hex mesh topology modification. Section 4 describes the atomic operations we use to describe others’ topology modification operations. Section 5 relates these operations to the flipping operations described by Bern et. al. Section 6 gives conclusions and future directions for this work.
2. THE HEXAHEDRAL MESH DUAL

Before describing our work in hex topology modification, we describe here some characteristics of the dual in terms of complexes of polytopes and arrangements. Describing the hex dual in these terms allows us to take advantage of the wealth of prior work on combinatorial relations for polytope complexes and arrangements (e.g. [8]).

To define the dual, we start by identifying the hexahedral mesh as a cell complex $P$ whose cells $P_i$ are hexahedral. Then, the dual is defined as a one-to-one mapping $\Psi(P): P \rightarrow P^*$ which preserves but reverses incidence relations:

$$P_i \subset P \iff \Psi(P_i) \subset \Psi(P).$$

The dual $P^*$ we identify as a polytope complex, with each $k$-face dual to a $(d-k)$-face in the hexahedral mesh.

A great deal of results have been published on various combinatorial relations of polytope complexes and arrangements. Describing the dual cells must be simple polytopes. In particular, if it can be shown that the dual vertices bounding those cells. Therefore, (1) must apply to all the dual cells as well as all the dual vertices bounding those cells. Therefore, the dual cells must be simple polytopes.

2.1. The hex dual is a complex of simple polytopes

Theorem 1: The dual of a hexahedral mesh is a complex of simple polytopes.

Proof: Consider a hex element $P$, one of its bounding vertices $V(P)$, and the sets of 1-faces $F^1(P, V)$ and 2-faces $F^2(P, V)$ of $P$ incident on $V$. Because a hexahedron is a simple polytope, $\text{card}(F^2(P, F_0^0)) = d$. Also, since every $k$-face of a simple polytope is formed by the intersection of $(d-k)$ supporting faces of the polytope,

$$\text{card}(F^1(P, F_0^0)) = \left(\frac{d}{d-1}\right) = d.$$ 

Since the dual preserves incidence relations (and does not change dimension),

$$\Psi(P) \subset \Psi(F^2(P, V)) \subset \Psi(F^1(P, V)) \subset \Psi(V),$$

or

$$V^* (P^*) \subset F^1 (V^*, P^*) \subset F^2 (V^*, P^*) \subset P^*,$$

with

$$\text{card}(F^2 (V^*, P^*)) = \text{card}(F^1(P, V)) = d$$

and

$$\text{card}(F^1 (V^*, P^*)) = \text{card}(F^2(P, V)) = d.$$ (1)

However, because the dual relation between entities is one-to-one and we chose an arbitrary vertex and any of the hex elements bounding it, (1) must apply to all the dual cells as well as all the dual vertices bounding those cells. Therefore, the dual cells must be simple polytopes.

2.2. The hex dual is a simple arrangement of hyperplanes

Theorem 2: The dual of a hexahedral mesh forms a simple arrangement of hyperplanes.

Proof: We start by deriving some incidence relations for vertices in a simple arrangement, by constructing a simple arrangement containing a single vertex. Consider a single surface, $h_p$, passing through a point $p$. Now, add a second surface, $h_q$, different from the first, also passing through $p$. The intersection of the two planes is a line contained in each plane, which partitions each plane into two facets. Similarly, add a third plane, $h_r$, which also passes through $p$, and is affinely independent of the other two planes (so its intersection with each plane generates a new line on that plane). Note that the third plane has to be independent of the first two, since for simple arrangements each 1-face can only be formed by the intersection of 2-planes. When $h_r$ is added, a new line of intersection is formed with each of the other planes, such that each plane is partitioned by two lines, formed by its intersection with the other two planes. Since all the lines pass through $p$, each plane is partitioned into 2-faces. The vertex at $p$, formed by the intersection of these three hyperplanes, is therefore connected to 12 2-faces of the arrangement. The number of 1-faces incident on the vertex is computed by recognizing that three lines pass through, and are each partitioned in half by $p$. Therefore, 6 1-faces are incident on the vertex. Finally, because each plane is affinely independent of the others, and passes through $p$, they each partition all existing cells in half (where $R^1$ is taken as a single cell), so the vertex is connected to $2^3 = 8$ 3-faces. No more planes can be added which pass through $p$, by definition of a simple arrangement. Therefore, for a simple arrangement:

$$f_0(V) = 6$$

$$f_1(V) = 12$$

$$f_2(V) = 8$$

Next, we consider a single hexahedral element $P$ in the mesh. A hex is a cuboid polytope, which by definition is bounded by 2$d$ $(d-1)$-faces (e.g. one pair for each parametric direction):

$$f_{d-1}(P) = 2d$$

Again, because $P$ is a simple polytope, it obeys the Dehn-Sommerville relations, and specifically for $d=3$, we have[19]:

$$f_0(P) = 2f_2 - 4$$

$$f_1(P) = 3f_2 - 6$$

Combining these, we get

$$f_0(P) = 4(d - 1) = 8$$

$$f_1(P) = 6(d - 1) = 12$$

$$f_{d-1}(P) = 2d = 6$$

Now, we apply the dual transformation to these relations. As in the previous section, $d$ is unchanged, and each $k$-face corresponds to a $(d-k)$-face in the dual. Since Eqs. (2) apply to $P$ in the primal, they apply to $\Psi(P) = V^*$ in the dual; furthermore, the cardinalities don’t change, since $\Psi$ preserves incidence relations. Therefore, Eqs. (2) become:

$$f^*_0(V) = 8$$

$$f^*_1(V) = 12$$

$$f^*_2(V) = 6$$

However, these are the same numbers of incidences as occur in simple arrangements. No other 1-, 2- or 3-faces can be connected to $V^*$, because by dual correspondence that would

\[\text{card}(F^2(P, F_0^0)) = d.\]
mean adding extra 2-, 1- or 0-faces to the original polytope, respectively, in which case the polytope would no longer be a hexahedron. Since all vertices in the dual arrangement of a hex mesh must have incidences as in (3), and since all simple d-dimensional arrangements must have vertices with valences as in (2), then the dual of a hex mesh must form a simple arrangement.

\[\text{Note: Most combinatorial relations for arrangements assume that hyper-surfaces intersect each other at most a constant number of times \(s\). However, Theorems 1-2 above use only information local to the polytope/vertex combinations. Therefore, as long as the 2-faces local to the polytope and vertex are distinct and not formed from the same hyper-surface locally, we argue that then the Dehn-Sommerville relations hold locally and multiply-intersected hyper-surfaces do not change the results.}\]

2.3. General comments

By themselves, Theorems 1-2 do not seem to have much relevance to the subject of hexahedral meshing. However, showing that a hexahedral mesh dual is a simple arrangement allows us to take advantage of the Dehn-Sommerville relations, which govern the counting relations between entities in any valid hex mesh. We plan to use these relations to define permissible states, and transitions between those states, in the dual arrangement, thereby proving the completeness of our set of atomic operations. This paper is a work in progress toward that goal.

3. GENERAL APPROACH

One way to look at the hex mesh dual is that hex elements are induced by the intersection of 3 dual sheets pairwise. It follows that mesh connectivity can be modified by locally “deforming” sheets to produce more intersections with other sheets in the neighborhood of the deformation. If only sheet interiors are deformed, mesh connectivity outside the deformed region is unchanged. Thus, the topology modification is local.

We characterize local hex mesh topology modification in terms of atomic, local combinatorial modifications to the dual arrangement, and show that these operations can describe other known topology modifications in hexahedral meshes. Atomic modifications to the arrangement are defined as the smallest units of combinatorial change to the arrangement which keeps the arrangement simple. Using these atomic operations, non-local topology modifications can be accomplished by applying the operations sequentially. Since each operation is reversible, this sequence includes some operations applied in the forward sense and others in the reverse sense. Although some operations introduce what would normally be considered poor quality elements in the primal mesh, this is an intermediate state which gets removed eventually by other operations. This is analogous to the construction of quadrilateral meshes by various algorithms, where poor-quality quads are formed initially but then removed a-posteriori to meshing.

There are two primary applications of these modification operations: quality improvement, and adaptive refinement/coarsening, analogous to the application of analogous operations in tetrahedral meshes. (In practice, these applications are not completely independent of each other.) We speculate that applying our dual-based operations will require “steering” sheet deformations toward areas of interest, either to eliminate structures in the arrangement which induce poor quality, or to enrich/coarsen the arrangement in areas where adaptive refinement or coarsening is desired. We leave this subject for future work.

4. ATOMIC OPERATIONS

We present candidates for atomic, local, dual-based hex topology modification operations in turn in subsequent sections. Formally, we identify four reversible operations: chord push, hex push, minimal pillowing, and face/ring collapse. The first two operations modify interior regions on three and four sheets, respectively. Minimal pillowing creates a new sheet interior to the mesh along with modifying two existing sheets. Face collapse merges two sheets by forming and joining a new interior boundary on each sheet. We also describe two higher-level operations, the triple-chord push and triple-hex push, which are combinations of elementary operations which occur frequently when describing topology modifications studied in previous works.

4.1. Chord Push

The minimum combinatorial change one can make in an arrangement is the addition or removal of d-facets in the arrangement. Since we restrict ourselves to locally-simple arrangements, new facets must be introduced in a way which preserves that characteristic. That means, for example, that the arrangement cannot be modified by deforming two dual surfaces such that they meet only at a single vertex or 0-facet. By this reasoning, then, the smallest (constructive) combinatorial change in a simple arrangement is to deform two sheets along a common third sheet which intersects both, such that they intersect locally. This “chord push” operation is depicted in Figure 2; starting with the arrangement shown on top, the 2-cells ab and cd are deformed such that they intersect, modifying the arrangement to look like Figure 2, bottom. A chord push operation introduces two new 3-cells, 4 new 2-cells, six new 1-cells, and two new 0-cells in the dual. In the primal, two new hex elements and nodes are introduced.

\[\text{Note: Most combinatorial relations for arrangements assume that hyper-surfaces intersect each other at most a constant number of times \(s\). However, Theorems 1-2 above use only information local to the polytope/vertex combinations. Therefore, as long as the 2-faces local to the polytope and vertex are distinct and not formed from the same hyper-surface locally, we argue that then the Dehn-Sommerville relations hold locally and multiply-intersected hyper-surfaces do not change the results.}\]
Again, there are no combinatorial changes to all involved sheets outside the bounding circle on each sheet, therefore this operation is also local.

Figure 2: The chord push operation in the dual. Intersecting two 1-cells (ab and cd) to form 2 new 0-cells (x and y), 6 new 1-cells (ax, cx, xy (2), yb, yd), 6 new 2-cells (axc, bdy, yxy (4)), and 2 new 3-cells. New xy 1-cells make up blind chord on sheets 2 and 3; xy 1-cells on 12 and 34 chords correspond to 1-cells ab and cd, resp., in original arrangement (top, sheets 2 and 3).

Since there are no combinatorial changes outside the bounding circle on each sheet, the chord push operation is local.

In the primal, the interpretation of a chord push is an opening of two interior faces, creating two deformed hexes that share four faces (Figure 3). Clearly, six new faces are introduced (four interior and two copies of the original two faces) along with six edges (5 interior & 1 copy) and two new nodes, corresponding to the new d-cells in the arrangement enumerated in Figure 2. Although a chord push induces poor connectivity initially, a sequence of this and other operations will result in improved quality.

Figure 3: Primal view of chord push operation.

4.2. Hex Push

The hex push operation (see Figure 4) is similar to a chord push, except that, instead of deforming the sheets along a common sheet, they are deformed along a common intersecting sheet pair, or chord. The two vertices, each the intersection between that chord and one of the sheets being deformed, are reversed in their sequence of intersection along the common chord. This appears on both sheets forming the common chord, and on each deformed sheet, as a new line of intersection with the other deformed sheet surrounding the intersection with the common chord.

We are not certain a hex push is an atomic operation. Repeated applications of chord push operations allows one to come close to surrounding two vertices; we have not yet worked out the final stage of this operation, where the two hexes are pushed through each other. We speculate that this is really the result of somehow collapsing the original hexes and opening new hexes in the appropriate places in the arrangement. This is the subject of further study.

In the primal representation (Figure 5), a hex push swaps the relative locations of the hexes involved in the push. Yet, since the chords of the crossing of dual nodes are anchored on the bounding circle, we create four other crossings as well. These new crossings become four new hexes in the primal mesh. These hexes somewhat “pillow” the newly swapped hexes (Figure 5) all within the bounding faces of this operation. In contrast to the chord push operation, which provably forms elements with negative jacobian quality metrics, the hex push operation forms hexes which do not have intrinsically degenerate quality (though their quality will usually be poor).
4.3. (Minimal) Pillowing

In a “pillowing” operation, a new dual sheet is inserted in the dual arrangement such that it induces new vertices. In previous work, the (traditional) pillowing operation has been formulated such that it surrounds an existing vertex in the arrangement [15][17]. However, our view of this operation is a smaller, atomic operation, which is formed along a 1-cell in the arrangement instead of around a vertex. We will sometimes refer to this as a (minimal) pillow operation, to distinguish it from the (traditional) pillowing operation. The dual representation of this operation is shown in Figure 6. The primal corresponding to the (minimal) pillowing operation is shown in Figure 7. Clearly, the quality of the hex elements formed by this operation will be intrinsically degenerate. However, as before, a sequence of operations can be used to improve the quality.

It is straightforward to show that the (traditional) pillowing operation is not atomic; this is shown in Figure 8. Starting with a single dual vertex, a (minimal) pillow is formed along one of the 1-cells to form two new vertices, then a hex push operation is performed between one of the new vertices and the original vertex. The result is the dual arrangement corresponding to a (traditional) pillowing operation performed on the original arrangement. Thus, a (traditional) pillowing operation is a sequence of two atomic operations: a (minimal) pillow, followed by a hex push.

4.4. Face Collapse

We have observed cases where two distinct hex meshes with identical quadrilateral boundaries whose dual arrangements have different numbers of non-pillow hyperplanes, with chords which have different starting and ending quadrilateral faces. For example, the (2,1)-(1,2) flip from [18] starts with five dual sheets in (2,1), but merges two of them to arrive at four dual sheets in (1,2). By definition, transitioning between such meshes using the atomic operations defined thus far is impossible, since none of our operations allows chords to be broken and reconnected, or dual sheets to be broken and reconnected. On the other hand, this type of operation should not be disallowed, as long as it can be done without modifying the quadrilateral boundary, or some bounded region inside the dual arrangement.

We have derived an operation which performs such a modification of the arrangement. This operation has an intermediate state which includes “knife” elements, which are produced when an interior face in the mesh is collapsed by
joining two opposite nodes. If the chord corresponding to the face being collapsed is a self-terminating chord (one which does not emerge on the region boundary), the collapse operation produces a single line of self-intersection which terminates on both ends at knife elements. Collapsing the column of elements corresponding to that chord removes the knife elements, changing connectivity only inside the local region. The total operation, consisting of collapsing an entire “ring” of elements, can be considered atomic (because there are no intermediate states with a “valid” arrangement). However, this operation is different from the others, in the sense that it can only be applied to a self-terminating chord (in order to avoid modifying the surface mesh). A ring collapse also results in a merge of two surfaces in the arrangement. The ring collapse, combined with the two hex push operations, combine to form the results in the transition to the (2,1)-(1,2) flip from [18]. In this case, it happens that one of the “blind” chords produced from the hex pushes is the one collapsed.

4.5. Elemental Operations

We define elemental operations as operations which are not atomic, but which occur often enough that they are useful as distinct sets of atomic operations. We identify three elemental operations: the triple-chord push and the triple-hex push, along with the “traditional” pillowing already described in Section 4.3.

4.5.1. Triple-Chord Push

A triple-chord push modifies the topology of three sheets. In essence, this operation “pulls apart” three faces sharing a common node, such that the result is two hexes sharing three faces. There are many interesting things about this particular arrangement, whose dual and primal representations are shown in Figure 9 and Figure 10, respectively. First, the outer boundary of the polytope shown in Figure 10 has the same number of faces, edges and vertices as a hexahedron, but its dual is not isomorphic to the simple arrangement resulting from the hexahedral elements we have considered up to this point. If it were not for that fact, this arrangement would result in a “parity-flip” operation described in [17]. The composition of a triple-chord push of atomic operations has not yet been derived. We speculate, though, that this operation is composed of three hex pushes (to separate the three faces, and the edges shared by them pairwise, into two sets of three faces each, all sharing the interior node), followed by a “node separate”. This last operation might be one more fundamental than a hex push, and could replace the hex push as an atomic operation (meaning that the hex push would be the combination of several chord pushes then a node separate). We plan to investigate this set of operations further in the future.

4.5.2. Triple-Hex Push

Executing on four sheets, a triple-hex push is accomplished by pushing a dual surface through a dual vertex (see Figure 11).
In addition to future theoretical work, we also plan to develop applications of these operations, specifically for the purposes of hex mesh improvement. Using the non-local information provided by dual sheets, we plan to develop methods for "steering" modifications towards areas of poor quality, enriching or coarsening the topology in order to improve mesh quality locally.

7. REFERENCES


TETRAHEDRAL MESH GENERATION FROM SEGMENTED VOXEL DATA

Tushar Udeshi

Zyvex Corporation, 1321 N. Plano Road, Richardson, TX, U.S.A. tudeshi@zyvex.com

ABSTRACT

We describe a robust algorithm to generate a tetrahedral mesh from segmented voxel data. The majority of mesh generation algorithms work from boundary representations. However, measured data such as CT and MRI scans as well as simulated data such as results of Micro Electromechanical Systems (MEMS) process emulation often exist in voxel format. Our algorithm generates a graded and adaptive tetrahedral mesh without the need for extracting a boundary mesh. The algorithm generates a consistent mesh when multiple materials are present. Local error-control and refinement are also supported.

Keywords: Tetrahedral mesh generation, voxels, shared octree, medical FEA

1. INTRODUCTION AND PREVIOUS WORK

The MEMulator™ software [1, 2] developed at Zyvex Corporation, for MEMS process emulation runs in voxel space. MEMS processing consists of a series of deposition and etch steps. Traditional MEMS design software use a solid modeling kernel to emulate these steps. However, this approach has robustness and accuracy issues. Deposition is emulated using a solid modeling offset operation which is prone to numerical instability due to the complex surface intersections which occur. Moreover, the resulting geometry generated is not accurate. The MEMulator, on the other hand, uses voxels to represent the simulation space. This technique does not have any robustness issues. The high memory and computational cost typically associated with voxels were solved by the shared-octree data structure and results-matching technique [2]. This resulted in accurate MEMS geometry in voxel space. We then faced the problem of generating a volumetric mesh to enable finite element analysis of these parts.

Most mesh generation algorithms start with a polygonal boundary representation of the domain. These include Constrained Delaunay tetrahedralization-based algorithms [3], octree-based methods [4, 5] and advancing front techniques [6]. These methods need a smooth watertight boundary representation as input and suffer from the drawback of committing too early to the surface mesh density. Early octree-based meshing algorithms [7] started with a volumetric representation of the domain, where one didn’t have an explicit boundary representation but could do an inside/outside test. They generated a modified octree which included octants that were cut and a tetrahedralization was generated from this. When the model consists of multiple regions, the advancing-front and octree-based methods typically mesh each region separately while constraining the boundary meshes of adjacent regions to match up.

There has been some work done on generating tetrahedral meshes from images in two and three dimensions. In [8], every interior voxel is split into 5 or 6 tetrahedra. Voxels straddling the boundary are meshed using a lookup table. The mesh generated by this technique is uniformly dense. In [9], the voxels are dubbed together into larger cells in homogeneous regions. Cells which are surrounded by cells of the same size are split into five tetrahedra. All other cells are meshed using Delaunay tessellation. Hale [10] sprinkles points and adjusts their positions by minimizing atomic energy
forces. He then runs a Delaunay triangulation algorithm on these points. However, there is no guarantee that the resulting mesh will conform to the domain boundary. Langer et al. [11] start with a uniform triangulation and then use simulated annealing and refinement procedures to conform to the input 2D image. This technique will probably be too computationally expensive when extended to three dimensions.

In Section 2, we describe the mesh generation algorithm, which primarily is a node-placement algorithm for conformal Delaunay tetrahedralization. Section 3 gives a few implementation details including unique data structures and acceleration techniques used. We list a few application areas for this algorithm and show some example meshes in Section 4. We conclude and suggest some improvements in Section 5.

2. THE ALGORITHM

The algorithm can be considered to be a constrained Delaunay tetrahedralization. We first generate an octree (referred to as a FEA mesh tree) from the segmented voxel data. Each node of the FEA mesh tree contributes one node to the tetrahedral mesh (referred to as a Delaunay node). We adjust the levels of the nodes of the FEA mesh tree so that they satisfy what we term as surface integrity conditions. We then input the nodes generated by the FEA mesh tree into a Delaunay tetrahedralization algorithm. The surface integrity conditions guarantee that the resulting tetrahedralization conforms to the boundaries of the domain.

2.1 Generation of FEA mesh tree

The input into our algorithm is voxels with volume of fluid information i.e. the percentage by volume of the different regions present in the voxel is stored. An FEA mesh tree is an octree and is comprised of two types of nodes: filled and interface. Filled nodes have only one material in them while interface nodes have two or more materials. The topology of the geometry in a cell is restricted. The boundaries in a cell must form one $C_0$ component. Note that the boundaries inside a cell need not be manifold and non-manifold boundaries occur in cells having multiple materials. This is illustrated further in Figure 1. The FEA mesh tree is built in a bottom-up manner by merging voxels. Interface nodes are merged only if the boundary of every region inside the interface node can be approximated by a plane up to a user-specified error threshold (Note that this condition is relaxed in interface nodes containing features as described in Section 2.1.1). The boundary in an interface FEA node is represented by a list of points (called interface points), which are the midpoints of all interface voxels under that FEA node.

![Figure 1: Interface Nodes: (a) Invalid - boundary has 2 components (b) Valid - The boundary is $C_0$ continuous although it is non-manifold.](image)

2.1.1 Assigning Delaunay points to FEA nodes

An interior FEA node is assigned a Delaunay point at its center. Many classes of man-made geometry such as those generated with CAD packages have sharp features like edges and corners. Placing a Delaunay point on these features improves the conformity of the mesh to the geometry. Moreover, mesh density is also reduced since the error tolerance check described in Section 2.1 need not be done at edges and corners. We run a feature-detection algorithm and assign a feature-strength to each interface point. For natural data like CT and MRI scans, this step is skipped and the feature-strength of all interface points are nullified. Feature detection is performed on interface points using an adaptation of a corner detection image processing algorithm [12]. Spatial image gradients for every point in a 3x3x3 neighborhood of every interface point is calculated. The image gradients $E_x$, $E_y$, $E_z$ are computed using central differences. The scalar values used for computing these central differences is as follows: Each interface point, say $P$ is assigned a reference region, say $m$, which is the region which occupies the largest fraction by volume of the corresponding interface voxel. The scalar values used while computing image gradients for the feature-strength of $P$ is the fraction of $m$ in the voxels in its immediate neighborhood. The feature-strength of $P$ is computed as follows:

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y & \sum E_x E_z \\ \sum E_x E_y & \sum E_y^2 & \sum E_y E_z \\ \sum E_x E_z & \sum E_y E_z & \sum E_z^2 \end{bmatrix}$$

The summation of the elements in matrix $C$ is taken over a 3x3x3 neighborhood of $P$. Since $C$ is symmetric it can be diagonalized by a rotation of the coordinate axis. Therefore $C$ can be thought of as a diagonal matrix:
Assume $\lambda_1 \geq \lambda_2 \geq \lambda_3$. If $\lambda_3$ is greater than a threshold, that interface node is tagged as a corner and its feature-strength tagged as $\lambda_1$. Else if $\lambda_2$ is greater than a threshold, the node is marked as an edge and its feature strength set to $\lambda_1$.

In addition to feature strength, every interface point also stores the number of regions in a $3\times3\times3$ neighborhood of that point. This is called $r\text{Count}$.

Given this information, the points are ordered. The highest weightage is given to $r\text{Count}$. Therefore, nodes are placed at junctions where multiple materials meet. Then we look at the tag. Corner nodes are ordered before edge nodes. If two nodes have the same tag, they are ordered based on their feature-strength. The Delaunay point is the location of the first interface point using this ordering scheme.

### 2.2 Surface Integrity Conditions

The surface integrity conditions ensure that the Delaunay tetrahedrization conforms to the boundaries implicit in the voxel data and is the most important contribution of this paper. Additionally, they facilitate smooth mesh gradation. The surface integrity conditions are expressed as local conditions on the FEA mesh nodes. They are:

1. Two neighboring FEA nodes can differ by at most one level.
2. Two filled FEA nodes of different regions cannot be adjacent to one another. There must be an interface FEA node between them.
3. An interface FEA node can never be adjacent to an interior FEA node which is smaller (lower level in the FEA mesh tree) than it.
4. The Delaunay point of an FEA interface node must be in a cube which is generated by intersecting the boundary of that FEA mesh node by 16.7%.
5. The boundary inside an interface FEA node must be $C_0$ connected (may be non-manifold).
6. Two interface FEA nodes can be adjacent to one another only if they belong to the same surface. Else there must be a filled node between them.

The first condition is not needed to maintain surface integrity but facilitates a smooth gradation of the mesh from coarse to dense regions. The second condition validates the FEA mesh octree - there must be an interface node separating two regions.

The third and fourth conditions guarantee that there will never be a Delaunay edge between 2 filled nodes of different regions. For an input point set $S = \{P_i\}$ in $R^d$, a Delaunay edge can exist between $P_i$ and $P_k$ if and only if $\exists P \in R^d$ such that $\text{dist}(P, P_i) = \text{dist}(P, P_k) < \text{dist}(P, P_j) \forall P_j \in S, i \neq j, j \neq k$, where $\text{dist}$ is Euclidean distance in $R^d$. We will first explain this in two dimensions. Refer to Figure 2. Surface integrity condition 4 forces the blank (interface) nodes to be at the same size or smaller than its adjacent filled nodes. We consider the worst case when they have the same size. The Delaunay points contributed by the filled nodes will be at their center. The dotted line represents the perpendicular bisector of the edge $P_jP_k$. Now the edge $P_jP_k$ will be a Delaunay edge only if there is some point on the dotted line which is closer to $P_j (P_k)$ than any other input point. Each of the interface (blank) nodes also contributes one Delaunay point. Consider interface node $I$. Irrespective of where the Delaunay point, say $N$, lies in $I$, its distance to any point, say $P_h$ on the lower half of the perpendicular bisector will always be equal to or less than the $\text{dist}(P_i, P_h)$. The worst case is shown as node $N$ in Figure 2, where $\text{dist} (N, P_h) = \text{dist} (P_i, P_j)$. The other interface nodes will contribute Delaunay points which would be closer to points on the upper half of the perpendicular bisector. Therefore an invalid edge $(P_jP_k)$ between two filled nodes of different regions can never occur. Similarly one can enumerate all possible invalid edges between different filled regions and we find that these can never be Delaunay edges if the surface integrity conditions are enforced.

Now refer to Figure 3. There is a corner in the domain (shown as a lightly-shaded rectangle). Delaunay edges $P_1N$ and $P_2N$ would be invalid. However arguments similar to above reveal that these edges will never occur irrespective of the positions of $N_1$, $N_2$, or $N_3$ in their respective FEA mesh nodes.

In two dimensions surface integrity condition 4 is not needed. However in three dimensions invalid Delaunay edges are possible between two filled nodes of different regions, if the the Delaunay point contributed by an interface FEA node is allowed to be anywhere inside the FEA node. We have enumerated all cases and determined the worst case. This is shown in Figure 4. Assume that the the Delaunay point of interface FEA node $I$ is chosen on its back lower left corner $C$. Assume that all the FEA nodes are cubes with edges of length 1 unit. Note that $N$ is equidistant to $P_1$ and $P_2$. Now, $\text{dist}(P_1N) = \text{dist}(P_2N) = \sqrt{1^2 + 0.5^2 + 0.5^2} = 1.23$. However $\text{dist}(CN) = \sqrt{1^2 + 1^2 + 0.5^2} = 1.50$. Therefore, the Delaunay point of interface node $I$, does
not prevent the edge $P_1 P_2$ from occurring. However, if we force the Delaunay point of $I$ closer to the center of $I$, its distance to $N$ would reduce, thereby preventing this edge from occurring. Let $x$ be the distance of the Delaunay point of $I$ from $C$. Then, we should satisfy:

$$\sqrt{(0.5 - x)^2 + (1 - x)^2 + (1 - x)^2} \leq 1.23$$  \hspace{1cm} (3)

Solving Equation 3, we get $x \geq \frac{1}{2}$. Therefore, if the Delaunay point contributed by $I$ is in a box inset by $16.7\%$ (condition 4), it will prevent the invalid edge $P_1 P_2$ from occurring. Now, if using the criteria described in Section 2.1.1, the Delaunay point does end up being outside the box, there are three options: (1) Move the Delaunay point inside the box if it is within a user-specified error threshold. This adds artificial roughness to the surface defining the interface. (2) Refine the interface FEA node. (3) Leave the Delaunay node where it is. Check and fix invalid edges in a post-processing step by local edge flipping operations.

A Delaunay point can only lie on one surface. Therefore the boundary inside an interface FEA node must be connected. Surface integrity condition 5 enforces this. When three or more interface nodes occur side by side, there may be a Delaunay edge generated between two of these nodes which are not neighbors. This is illustrated in Figure 5. Surface integrity condition 6 avoids this. A chain of consecutive interface nodes must all belong to the same surface.

**Figure 2:** Invalid edge $(P_1 P_2)$ between two filled regions will never occur. The hashed nodes are filled nodes while the clear nodes are interface nodes.

**Figure 3:** Invalid edges $(P_1 N_1$ and $P_2 N_1)$ will never occur. The hashed nodes are filled nodes while the clear nodes are interface nodes. The domain is shown as a lightly-shaded rectangle.

### 2.3 Generating tetrahedra

The Delaunay points from a FEA mesh tree satisfying all the surface-integrity conditions are fed into a Delaunay Tetrahedralization routine [13]. The volumetric mesh of all regions is therefore generated simultaneously. The tetrahedra are traversed and are assigned region numbers using the following routine:

**ForEach tetrahedron T**

- If (any vertex of T is from a filled FEA node of region R)
  - Assign $R$ as region of T
- Else If (3 only one region R common to all the interface nodes)
  - Assign $R$ as the region of T
- Else
  - Sample T uniformly and assign it to maximally-populated region
**End If**
**End For**

Delaunay tetrahedralization may generate some sliver tetrahedra. These are removed using local edge flipping and optimization-based smoothing operations [13]. After smoothing on an average ANSYS® generates shape warnings for less than 1% of the tetrahedra, when assuming they are linear. Our algorithm does not provide any theoretical guarantees about the quality of tetrahedra, as in [5]. However the mesh quality is similar to some other commercial meshing codes we tested.
2.4 Local error control and refinement

An experienced analyst can determine which regions of the domain are important in a simulation and which are unimportant. In regions of importance, it is desirable to have a high-density mesh which conforms closely to the boundary of the domain. On the other hand, in regions of low importance, a coarse mesh which roughly approximates the domain boundary suffices. The error used to decide the level of subdivision of interface nodes described in Section 2.1 gives the user control over the mesh density. A low error in regions of importance results in a higher subdivision and therefore greater conformity to the domain boundary. Additionally, the user could explicitly refine the FEA mesh tree further in regions of interest. This is illustrated in Figure 6.

2.5 Robustness

Delaunay tetrahedralization has been very well studied and several robust and well-tested implementations exist [13, 14]. Therefore, if the enforcement of the surface integrity conditions is robustly implemented, our algorithm will always generate a consistent tetrahedral mesh irrespective of the complexity of the input data. This is in contrast to some algorithms which are based on heuristics.

3. IMPLEMENTATION

In this section, we give details about how the surface integrity conditions are implemented and then accelerated.

3.1 FEA Mesh Tree Data structure

We adapt the shared octree data structure [2] to store the FEA Mesh tree. We describe it here for reader's convenience. The shared octree is a complete octree. However, every unique node at every level in the octree is stored exactly once. There is only one filled leaf node for each region. Interface leaf FEA nodes differ by the volume of fraction of all materials they interface. The Delaunay point of an interface leaf node is also chosen at its center. We could locally estimate the shape of the surface and determine a better Delaunay point position. However, choosing the node at the center is sufficient if the dataset is at a high resolution. The uniqueness of the leaf nodes is maintained using a hash table. Every time a new FEA leaf node is created, a query is done into the hash table to determine if it already exists. If so, the parent simply points to the already created node and the newly created node is discarded. If not, the new node is inserted into the hash table. Higher level FEA nodes are also stored uniquely. The key into the hash table is the pointers to their 8 children. Note that if 2 interior FEA nodes have the same exact children then the Delaunay point they both generate will also have the same exact local coordinates. This is because the Delaunay point at an interior node is computed from only the interface points which are descendents of N. An example of a binary shared quadtree is shown in Figure 7 to illustrate the concept. In this case, there are only two types of leaf nodes, occupied (shaded) and vacant (blank).

The shared FEA mesh tree facilitates large savings in storage (up to 2 orders in magnitude) when dealing with geometries which have a lot of self-similarities, for
3.2 Implementing surface integrity conditions

The surface integrity conditions are enforced by making several passes over the FEA mesh tree. A decision about whether to subdivide a particular FEA mesh node can be made by looking at all its immediate siblings. For example, consider surface integrity condition 1. All FEA nodes which satisfy the error criteria described in Section 2.1 and their descendents are tagged as *squeezable*. A new FEA mesh tree is generated such that a node in the new FEA mesh tree remains *squeezable* only if the corresponding node in the old FEA mesh tree along with all its neighbors are also squeezable. Even though this procedure does not generate the optimally coarse tree, it is easy to implement, facilitates results matching (described in Section 3.2.1) and guarantees that the condition will be met. It is implemented using a recursive function shown below:

![Diagram](image-url)

**Figure 6**: An example of local refinement and local error control. The lower pad was given a lower error threshold than the upper pad. Therefore it conforms to the anchor better even though the two pads are geometrically the same.

**Figure 7**: A binary shared quad tree 3 levels high. The leaf nodes are not shown since only two types exist.

```c
FEANode *SC1(FEANode *in)
{
    FEANode *ret = new FEANode();
    ret->CopyData(in);
    if (!in->GetSqueezableFlag())
    {
        ret->SetSqueezableFlag(false);
        for(i=0 -> 8)
            ret->SetChild(i, SC1(in->GetChild(i)));
    }
    else
    {
        squeezable = true
        neighbors = GetNeighbors(in);
        foreach n in neighbors
            squeezable &= n->GetSqueezableFlag();
        ret->SetSqueezableFlag(squeezable);
    }
}

int Find(ret)
return ret
```

The function CopyData(in) copies the region numbers, interface points and Delaunay point position of in into the caller FEA node. The function GetNeighbors() fetches all the neighboring siblings by traversing the octree. The function Find() does a look up into the
FEA node hash table and is used to maintain uniqueness of all nodes in the FEA tree.

Now consider surface integrity condition 3. In this case we traverse the interface FEA nodes from top to bottom. When a squeezeable interface FEA mesh node, say \( n_1 \), is encountered, we look at all its immediate neighbors. If a neighbor, say \( n_2 \), is not squeezeable, we determine if any any child of \( n_2 \) which is adjacent to \( n_1 \) is a filled node. If this is so, then surface integrity condition 3 is violated and \( n_1 \) is refined. However, Delaunay points from filled children of \( n_1 \) are not added. This is because, this could violate surface integrity condition 3 for its neighbor and this could propagate further. This is illustrated in Figure 8.

![Figure 8](image)

**Figure 8:** Refinement for enforcing surface integrity condition 3 w.r.t one neighbor may cause it to be violated w.r.t another. Shaded nodes are filled FEA nodes while blank ones are interface. (a) Interface node 1 needs to be refined because it has a smaller filled node 2 adjacent to it. (b) When node 1 is refined, two if its children (4 and 5) may be filled interface nodes. Therefore we now have a smaller filled node (4 or 5) adjacent to a larger interface node (3), violating surface integrity condition 3. This is avoided by not adding Delaunay points of 4 and 5 into the Delaunay Tetrahedralization.

The surface integrity conditions are not independent of each other. Care must be taken not to violate one condition while enforcing another.

### 3.2.1 Results Matching

Results matching [2] is a technique for accelerating algorithms operating on shared octree data structures. We describe the shared octree concept briefly here for the reader's convenience.

Results matching accelerates operations performed on shared octrees by caching intermediate results. It takes advantage of the self-similarity in the domain. The operations must be (1) deterministic i.e. should give the same answer when given the same set of inputs and (2) its parameters should be local. So if for example an operation depended upon the global position of the octree node, results matching cannot be used. Whenever an operation is performed on a group of octree nodes the result is stored in a hash table which we call results cache. The key into this hash table are all the octree nodes and other parameters of the operation. The result is the group of octree nodes affected by the operation.

Fortunately, all surface integrity conditions can be checked by only looking at immediate neighbors and therefore we can take advantage of results matching. Here is pseudocode for enforcing surface integrity condition 1 with results matching:

```cpp
FEANode *SCIR(FEANode *in)
{
    // Check for results
    neighbors = GetNeighbors(in)
    result = FindResult(in, neighbors)
    if (result)
        return result
    FEANode *ret = new FEANode()
    ret->CopyData(in)
    if (!in->GetSqueezeableFlag())
    {
        ret->SetSqueezeableFlag(false)
        for (i = 0; i < 8)
            ret->SetChild(i, SCIR(in->GetChild(i)))
    }
    else
    {
        squeezeable = true
        foreach n in neighbors
            squeezeable &= n->GetSqueezeableFlag()
        ret->SetSqueezeableFlag(squeezeable)
    }
    ret = Find(ret)
    // Store result
    StoreResult(in, neighbors, ret)
    return ret
}
```

The function FindResult() does a lookup into the results cache. If the operation SCIR has already been performed on the same node which has the same set of neighbors, the resulting FEANode which was previously computed and stored in the results cache is returned. Else FindResult() returns NULL. The func-
4. RESULTS AND APPLICATIONS

In this section we show some of the meshes generated by our algorithm as well as list various application areas it has been applied to. All the meshes were generated on a 1.0 GHz PC with 1 GB of RAM, running Windows® XP.

4.1 MEMS devices

The output of the MEMulator software is a shared octree which accurately represents MEMS geometry. We generate tetrahedral meshes from this geometry so that various analysis including electrothermal and structural can be performed. Figure 9 shows a closeup of a bidirectional actuator. The mesh has 9160 elements and is generated in 17.70 seconds.

![Figure 9: Mesh of a MEMS bidirectional actuator.](image)

4.2 Medical data

The finite element method has a large field of application in the engineering sciences domain. However, recently, this method has gained interest in the medical field, for example, modeling irradiation in tumor therapy [15]. The MRI or CT scan data can be segmented into various entities, for example, bone, tissue, skin etc. These regions can then be fed into our algorithm and meshed. The mesh in Figure 10 was generated after a crude segmentation (thresholding on the scalar values) of the head dataset which comes packaged with the Visualization Toolkit [16]. The model has a large number of disconnected components and is highly convoluted. The tetrahedral mesh has 208,350 elements and was generated in under half an hour. There is not much self-similarity in this data set and therefore results matching does not offer significant savings. Figure 11 shows a cross-section through the tetrahedral mesh generated from a segmented MRI scan of a human brain and has two regions. The complete mesh has nearly a million elements and took less than an hour to generate.

4.3 CAD model healing

Model stitching is the process of taking disjoint models and bringing them together so that they can be treated as one model. FEA requires a clean mesh as input. The mesh must be free of T-joints, cracks, self-intersections and holes. Unfortunately these flaws are all too common in CAD geometry and must be repaired. We perform model stitching and repair simultaneously and then generate a consistent volumetric mesh which can be used for analysis.

4.3.1 Converting CAD models to voxels

We use techniques described in [17] to do stitching and repair. We first scan-convert the CAD model into a shared octree. Our current implementation takes a polygonal model as input. However, any CAD format which can answer inside/ outside for an arbitrary point in space can be scan-converted. In our current implementation, we do the inside/ outside test for a query point P by shooting a ray from P in the positive direction and counting the number of polygons intersected. This count is called parity. An odd parity indicates that P is inside the model. This would work only for closed polygonal models. Nooruddin et. al. [17] extend the parity count method to open polygonal models by shooting rays in multiple directions and then using a voting scheme. Currently we only produce a digital tree i.e., a voxel can be fully filled or unoccupied. This can be modified to find the actual occupancy volume by taking several samples inside the voxel. We scan-convert all the models together into one shared octree. Note that once the models are converted to voxels, they are fixed. Voxels do not have any of the topological ambiguities of explicit surface representations.

4.3.2 Morphological Operations for Stitching

A dilation followed by an erosion of a distance, say $x$, is sufficient to stitch the models together, if $x$ is the maximum size of a crack. This is illustrated in
two dimensions in Figure 12. Dilation and erosion are equivalent to conformal deposition and etch operation in MEMS process emulation [1]. This technique smooths out features smaller than the dilation and erosion distance ($x$). It is equivalent to running a low pass filter over the model. In some applications this is desirable, for example model simplification. In FEA/CFD, the simplified geometry results in a coarser mesh which reduces computation time. If these small features are desirable, the dilation and erosion can be restricted to the junctions between the different CAD parts.

4.3.3 An example

We show the results for a Boeing 737. The fuselage, tail, wing, engine and the connector between the wing and the engine were separate CAD models and had several cracks and intersections as shown in Figure 13. These were voxelized, dilated and eroded as shown in Figure 14. Then the result was meshed as shown in Figure 15. The mesh has 109,488 nodes and 335,736 tetrahedral elements. The air surrounding the model is meshed too though it is not shown. ANSYS issues tetrahedral quality warnings for only 1088 elements (0.33%), when assuming they are linear elements.

5. CONCLUSIONS AND FUTURE WORK

We have presented a robust algorithm which generates a good-quality tetrahedral mesh directly from voxel data. Our technique is useful for applications where the original data is in voxel format such as CT scans and MEMS emulation. It is also useful where traditional boundary-based approaches fail because of inconsistencies in or high complexity of the boundary representation.

Our technique does not support any anisotropy. In general, since the octree cells are cubes, the tetrahedra generated have similar lengths in all three dimensions. In really thin regions, the tetrahedral mesh tends to get very dense. A mesh coarsening algorithm [18] could be applied to reduce the density in these regions.

Several improvements to the algorithm are possible. We would like better schemes for assigning regions to the tetrahedra (Section 2.3) and for detecting features
We would also like to investigate techniques of determining positions of Delaunay points inside interface voxels on the actual surface instead of at their center.

ACKNOWLEDGEMENTS

This work was done at Zyvex Corporation and was supported by NIST-ATP grant 70NANB1H3021. Zyvex Corporation is pursuing IP protection for these techniques. We would like to thank the Scientific Computing and Imaging Institute at the University of Utah for providing us with the segmented MRI dataset of a human brain.

References


Figure 11: Cross section of a 2-region mesh of a human brain.
Figure 12: Morphological stitching operations in 2D. (a) The original model. (b) Dilation (c) Erosion. The small gaps are filled up resulting in one model


Figure 13: CAD model of Boeing airplane.

Figure 14: (a) Binary-voxelized wing. Gaps and overlaps are visible between wing and fuselage (b) The stitched model.
Figure 15: (a) The surface of the volumetric tetrahedral mesh of a Boeing 737 model. Notice the smooth reconstruction and preservation of features. (b) A fairly complicated junction of the engine and wing. (c) Junction of the wing and fuselage. (d) Junction of the tail and fuselage.
Panel Discussion & Birds-of-a-feather Sessions
NOTES:
Index of Authors & Co-Authors

Alauzet Frederic............................... 335
Alliez, Pierre..................................... 215
Baker, Timothy................................... 55
Beall, Mark ......................................... 33
Berndt, Markos................................. 351
Blandford, Daniel ............................ 135
Blelloch, Guy.................................... 135
Borouchaki, Houman........................ 393
Branets, L. ....................................... 371
Brewer, Michael ........................139, 379
Bridson, Robert ................................ 103
Camarero, Ricardo........................... 293
Cabello, Jean ................................... 201
Cardoze, David. ............................... 135
Carey, Gary ................................79, 371
Chen Zhijian...............................307, 361
Crawford, Curran ................................ 21
Degener, P. ..................................... 227
Deister, Frank ................................... 43
Eiseman, Peter ............................... 277
Fedkiw, Ronald ............................... 103
Folwell, Nate .................................... 379
Freitag Daichin, Lori......................... 239
Frey, Pascal ..................................... 335
Gallier, J. ........................................ 159
Gee, J. ............................................ 159
George, Paul Lois. ........................... 393
Gotsman, Craig......................... 189, 215
Gumhold, Stefan .............................. 177
Guibault, Francois............................ 293
Haines, Robert .................................. 21
Hancq, Alfred. ............................... 307
Hassan, Oubay ................................. 43
Joshi, Bhautik................................. 251
Kadow, Clemens .............................. 135
Khachan, Mohammed ....................... 293
Klein, R. ......................................... 227
Knoop, Sarah .................................... 415
Knupp, Patrick ............................... 239, 379
Kocak, Suleyman ............................. 323
Kwok, Wa........................................ 307, 361
Leurent, Thomas.............................. 239
Melander, Darryl ................................ 239
Meseth, J. ....................................... 227
Miller, Gary L...................................... 91
Molino, Neil ....................................... 103
Ourselin, Sebastien ................................ 251
Owen, Steven J.................................. 121
Pav, Steven E. ................................. 91
Pekkan, Kerem .................................. 285
Plaza, Angel ..................................... 67, 79
Rajagopalan, Krishnakumar ............... 277
Ramaswami, S. ................................. 159
Rivara, Maria-Cecilia ......................... 67
Saigal, Sunil .................................... 121
Shashkov, Mikhail J. .......................... 351
Sheffer, Alla ..................................... 189
Shephard, Mark S. ............................. 33, 323
Shimada, Kenji .................................. 403
Shontz, Suzanne .............................. 147
Sifri, Oren ....................................... 189
Siqueira, M. ..................................... 159
Sondershaus, Ralf ................................ 177
Suarez, J.P. ...................................... 79
Sundaram, T. ...................................... 159
Surazhsky, Vitaly.............................. 215
Suresh, Krishnan ................................ 161
Tautges, Timothy J............................. 415
Tchon, Ko-Foa ................................... 293
Teran, Joseph ................................... 103
Tremel, Udo ....................................... 43
Tristano, Joseph.............................. 307, 361
Udeshi, Tushar .................................. 425
Vavasis, Stephen ............................. 147
Walkington, Noel J............................. 91
Walsh, Joe ......................................... 33
Wan, Jie ........................................... 323
Weatherill, Nigel .............................. 43
White, David ..................................... 121
Yamakawa, Soji ............................... 403
Zhu Jin ............................................. 315
## Index by Affiliation

<table>
<thead>
<tr>
<th>Affiliation</th>
<th>Authors</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argonne National Laboratory</td>
<td>Leurent, Thomas</td>
<td>239</td>
</tr>
<tr>
<td>ANSYS Incorporated</td>
<td>Chen, Zhijan</td>
<td>307, 361</td>
</tr>
<tr>
<td></td>
<td>Hancq, Alfred</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>Kwok, Wa</td>
<td>307, 361</td>
</tr>
<tr>
<td></td>
<td>Tristano, Joseph</td>
<td>307, 361</td>
</tr>
<tr>
<td>Carnegie Mellon University</td>
<td>Blandford, Daniel</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Blelloch, Guy</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Cardoze, David</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Kadow, Clemens</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Miller, Gary</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>Pav, Steven</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>Shimada, Kenji</td>
<td>403</td>
</tr>
<tr>
<td></td>
<td>Walkington, Noel J.</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>Yamakawa, Soji</td>
<td>403</td>
</tr>
<tr>
<td>Centre de Recherche en Calcul Applique</td>
<td>Camarero, Ricardo</td>
<td>293</td>
</tr>
<tr>
<td>(CERCA)</td>
<td>Guibault, Francois</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>Khachan, Mohammed</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>Tchon, Ko-Foa</td>
<td>293</td>
</tr>
<tr>
<td>Cornell University</td>
<td>Shontz, Suzanne</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>Vavasis, Stephen</td>
<td>147</td>
</tr>
<tr>
<td>EADS</td>
<td>Deister, Frank</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Tremel, Udo</td>
<td>43</td>
</tr>
<tr>
<td>EDS PLM Solutions</td>
<td>Cabello, Jean</td>
<td>201</td>
</tr>
<tr>
<td>Fluent Inc.</td>
<td>Zhu Jin</td>
<td>315</td>
</tr>
<tr>
<td>Georgia Institute of Technology</td>
<td>Pekkan, Kerem</td>
<td>285</td>
</tr>
<tr>
<td>INRIA</td>
<td>Alauzet, Frederic</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Alliez, Pierre</td>
<td>215</td>
</tr>
<tr>
<td></td>
<td>Frey, Pascal</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>George, Paul Louis</td>
<td>393</td>
</tr>
<tr>
<td>Israel Institute of Technology</td>
<td>Gotsman, Craig</td>
<td>189, 215</td>
</tr>
<tr>
<td></td>
<td>Sheffer, Alla</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>Sifri, Oren</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td>Surazhsky, Vitaly</td>
<td>215</td>
</tr>
<tr>
<td>Los Alamos National Laboratory</td>
<td>Berndt, Markos</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>Shashkov, Mikhail J.</td>
<td>351</td>
</tr>
<tr>
<td>Massachusetts Institute of Technology</td>
<td>Crawford, Curran</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Haimes, Robert</td>
<td>21</td>
</tr>
<tr>
<td>Program Development Company</td>
<td>Eiseman, Peter</td>
<td>277</td>
</tr>
<tr>
<td></td>
<td>Rajagopalan, Krishnakumar</td>
<td>277</td>
</tr>
<tr>
<td>Princeton University</td>
<td>Baker, Timothy</td>
<td>55</td>
</tr>
<tr>
<td>Rensselaer Polytechnic Institute</td>
<td>Kocak, Suleyman</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>Shephard, Mark</td>
<td>33, 323</td>
</tr>
<tr>
<td></td>
<td>Wan, Jie</td>
<td>323</td>
</tr>
<tr>
<td>Rutgers University</td>
<td>Ramaswami, S</td>
<td>159</td>
</tr>
<tr>
<td>Sandia National Laboratories</td>
<td>Brewer, Michael</td>
<td>239, 379</td>
</tr>
<tr>
<td></td>
<td>Freitag Diachin, Lori</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Knupp, Patrick</td>
<td>239, 379</td>
</tr>
<tr>
<td></td>
<td>Melander, Darryl</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Owen, Steven J</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>Tautges, Timothy</td>
<td>415</td>
</tr>
<tr>
<td></td>
<td>White, David</td>
<td>121</td>
</tr>
<tr>
<td>Affiliation</td>
<td>Authors</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Simmetrix, Inc.</td>
<td>Beall, Mark, Walsh, Joe</td>
<td>33, 33</td>
</tr>
<tr>
<td>Stanford University</td>
<td>Bridson, Robert, Fedkiw, Ronald, Molino, Neil, Teran, Joseph</td>
<td>103, 103, 103, 103</td>
</tr>
<tr>
<td>Stanford Linear Accelerator Center</td>
<td>Folwell, Nate</td>
<td>379</td>
</tr>
<tr>
<td>Universite de Technologie de Troyes</td>
<td>Borouchaki, Houman</td>
<td>393</td>
</tr>
<tr>
<td>University of Bonn</td>
<td>Degener, P, Klein, R, Meseth, J</td>
<td>227, 227, 227</td>
</tr>
<tr>
<td>University of Chile</td>
<td>Rivara, Maria-Cecilia</td>
<td>67</td>
</tr>
<tr>
<td>University of Las Palmas de Gran Canaria</td>
<td>Plaza, Angel, Suarez, J.P</td>
<td>67, 79, 79</td>
</tr>
<tr>
<td>University of Pennsylvania</td>
<td>Gallier, J, Gee, J, Siqueira, M, Sundaram, T</td>
<td>159, 159, 159, 159</td>
</tr>
<tr>
<td>University of South Florida</td>
<td>Saigal, Sunil</td>
<td>121</td>
</tr>
<tr>
<td>University of South Wales</td>
<td>Joshi, Bhautik, Ourselin, Sebastien</td>
<td>251, 251</td>
</tr>
<tr>
<td>University of Texas at Austin</td>
<td>Branets, L, Carey, Gary</td>
<td>371, 371</td>
</tr>
<tr>
<td>University of Tubingen</td>
<td>Gumhold, Stefan, Sondershaus, Ralf</td>
<td>177, 177</td>
</tr>
<tr>
<td>University of Wales Swansea</td>
<td>Hassan, Oubay, Weatherill, Nigel</td>
<td>43, 43</td>
</tr>
<tr>
<td>University of Wisconsin-Madison</td>
<td>Knoop, Sarah, Suresh, Krishnan</td>
<td>415, 261</td>
</tr>
<tr>
<td>Zyvex Corporation</td>
<td>Udeshi, Tushar</td>
<td>425</td>
</tr>
</tbody>
</table>
Distribution:

MS9018  Central Technical Files, 8945-1
MS0612  Review & Approval Desk, 9612

MS0321  9226 (200 copies)
To be distributed at the
12th International Meshing Roundtable
Santa Fe, New Mexico
September 14-17, 2003

For additional copies:
Sandia National Laboratories
PO Box 5800, MS0847
Albuquerque, NM 87185-0847
(505) 844-1309