

2013

## Real-Time High-Quality Image to Mesh Conversion for Finite Element Simulations

Panagiotis Foteinos  
*College of William & Mary - Arts & Sciences*

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Foteinos, Panagiotis, "Real-Time High-Quality Image to Mesh Conversion for Finite Element Simulations" (2013). *Dissertations, Theses, and Masters Projects*. Paper 1539623633.

<https://dx.doi.org/doi:10.21220/s2-aba0-9v40>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

Real-Time High-Quality Image to Mesh Conversion for Finite Element Simulations

Panagiotis Foteinos

Lefkada, Greece

Diploma from the Department of Electrical and Computer Engineering, University  
of Thessaly, Greece, 2006.

A Dissertation Presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Doctor of Philosophy

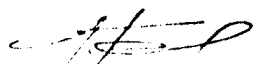
Department of Computer Science

The College of William and Mary  
January 2014

## APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

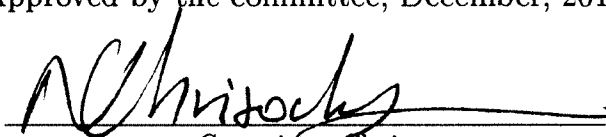
Doctor of Philosophy



---

Panagiotis A. Foteinos

Approved by the committee, December, 2013



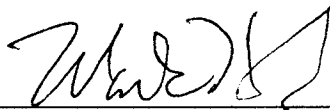
Committee Chair

Professor Nikos Chrisochoides, Computer Science  
Old Dominion University



---

Professor Weizhen Mao, Computer Science  
The College of William & Mary



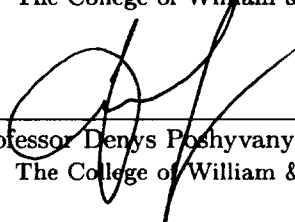
---

Professor Mark Hinders, Applied Science  
The College of William & Mary



---

Assistant Professor Pieter Peers, Computer Science  
The College of William & Mary



---

Assistant Professor Denys Poshyvanyk, Computer Science  
The College of William & Mary

## ABSTRACT

Technological Advances in Medical Imaging have enabled the acquisition of images accurately describing biological tissues. Finite Element (FE) methods on these images provide the means to simulate biological phenomena such as brain shift registration, respiratory organ motion, blood flow pressure in vessels, etc. FE methods require the domain of tissues be discretized by simpler geometric elements, such as triangles in two dimensions, tetrahedra in three, and pentatopes in four. This exact discretization is called a *mesh*. The accuracy and speed of FE methods depend on the *quality* and *fidelity* of the mesh used to describe the biological object. Elements with bad quality introduce numerical errors and slower solver convergence. Also, analysis based on poor fidelity meshes do not yield accurate results specially near the surface. In this dissertation, we present the theory and the implementation of both a sequential and a parallel Delaunay meshing technique for 3D and —for the first time— 4D space-time domains. Our method provably guarantees that the mesh is a faithful representation of the multi-tissue domain in topological and geometric sense. Moreover, we show that our method generates graded elements of bounded radius-edge and aspect ratio, which renders our technique suitable for Finite Element analysis. A notable feature of our implementation is speed and scalability. The single-threaded performance of our 3D code is faster than the state of the art open source meshing tools. Experimental evaluation shows a more than 82% weak scaling efficiency for up to 144 cores, reaching a rate of more than 14.3 million elements per second. This is the first 3D parallel Delaunay refinement method to achieve such a performance, on either distributed or shared-memory architectures. Lastly, this dissertation is the first to develop and examine the sequential and parallel high-quality and fidelity meshing of general space-time 4D multi-tissue domains.

## Table of Contents

Acknowledgments . . . . .	iv
List of Tables . . . . .	v
List of Figures . . . . .	vii
1 Introduction . . . . .	2
1.1 Motivation and Related Work . . . . .	2
1.2 Contributions . . . . .	12
2 Guaranteed Quality Tetrahedral Delaunay Meshing for Medical Images . . .	14
2.1 Preliminaries . . . . .	15
2.2 Algorithm . . . . .	17
2.3 Proof of Quality . . . . .	20
2.4 Proof of Good Grading . . . . .	30
2.5 Proof of Fidelity . . . . .	35
2.6 Implementation details . . . . .	40
2.6.1 Medial Axis Approximation . . . . .	41
2.6.2 Dihedral angle improvement . . . . .	43
2.7 Experimental Evaluation . . . . .	46
3 High Quality Real-Time Image-to-Mesh Conversion for Finite Element Simulations	56
3.1 Background: Delaunay Refinement for Smooth Surfaces . . . . .	57

3.2	Parallel Delaunay Refinement for Smooth Surfaces . . . . .	60
3.2.1	Poor Element List (PEL) . . . . .	61
3.2.2	Operation . . . . .	61
3.2.3	Update new and deleted cells . . . . .	62
3.2.4	Load Balancer . . . . .	62
3.2.5	Contention Manager (CM) . . . . .	63
3.3	Contention Manager . . . . .	64
3.3.1	Aggressive-CM . . . . .	65
3.3.2	Random-CM . . . . .	65
3.3.3	Global-CM . . . . .	66
3.3.4	Local-CM . . . . .	67
3.3.5	Comparison . . . . .	72
3.4	Performance . . . . .	75
3.4.1	Hierarchical Work Stealing (HWS) . . . . .	75
3.4.2	Strong Scaling Results . . . . .	76
3.4.3	Weak Scaling Results . . . . .	77
3.4.3.1	Hyper-threading . . . . .	81
3.5	Single-threaded evaluation . . . . .	83
4	4D Space-Time Delaunay Meshing for Medical Images. . . . .	90
4.1	Preliminaries . . . . .	91
4.2	Algorithm . . . . .	93
4.3	Termination and Quality . . . . .	95
4.4	Accuracy . . . . .	99
4.5	Experimental Evaluation . . . . .	103
4.6	Real-Time 4D Meshing . . . . .	105
4.6.1	Complexity . . . . .	105
4.6.2	Parallelization . . . . .	106
5	Conclusions and Future Work . . . . .	109

A Installing and Using the Software . . . . .	113
Bibliography . . . . .	116

## ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my advisor Professor Nikos Chrisochoides for his guidance and support the last six years. Nothing would be possible without his insight and encouragement over these years. Special thanks to the dissertation committee members Professors Mark Hinders, Weizhen Mao, Pieter Peers and Denys Poshyvanyk for their constructive suggestions and discussions. I would like to thank all my research collaborators and (ex) colleagues for fruitful discussions, specially Dr. Andrey Chernikov, Dr. Yixun Liu, and Dr. Andriy Kot for their cooperation on research projects and their friendship. I thank the professors of the College of William and Mary for making their teaching and coursework an inspiring and precious experience for my career. I thank the main office of the Computer Science Department, and specially Mrs. Vanessa Godwin and Mrs. Jacquelyn Johnson for their advice and help throughout these years as a PhD student. Special thanks to Reves Center team for its support to international affairs. I thank the Computer Science Department of Old Dominion University for providing everything necessary for my research during my stay there as a visitor researcher. I thank the Electrical and Computer Engineering Department of University of Thessaly and specially Professor Panayiotis Bozanis for the academic support and help throughout my undergrad years. Last but not least, I would like to thank my parents for always being there for me. This work is supported in part by NSF grants: CCF-1139864, CCF-1136538, and CSI-1136536 and by the John Simon Guggenheim Foundation and the Richard T. Cheng Endowment.



## List of Tables

2.1	Performance achieved by our algorithm and CGAL. . . . .	46
2.2	Information about the input images. . . . .	47
3.1	Comparison among Contention Managers (CM). . . . .	73
3.2	The specifications of the cc-NUMA machines we used. . . . .	75
3.3	Information about the input images. . . . .	77
3.4	Weak scaling performance. . . . .	78
3.5	Hyper-threaded performance. . . . .	81
3.6	Comparison between PI2M and CGAL. . . . .	84
4.1	Information about the images of the five patients. . . . .	103
4.2	Statistics of the output meshes generated for each patient. . . . .	103
4.3	The performance of the parallel 4D method. . . . .	107
A.1	The list and descriptions of influential macros. . . . .	115

## List of Figures

1.1	Image to Mesh Conversion on the BigBrain data. . . . .	3
1.2	Image to Mesh Conversion on a micro-CT bone structure. . . . .	4
2.1	The projection rule. . . . .	20
2.2	Illustration to the proof of Lemma 2.3. . . . .	23
2.3	Illustration to the proof of Lemma 2.5. . . . .	24
2.4	Flow diagram. . . . .	29
2.5	Illustration to the proof of Lemma 2.12. . . . .	36
2.6	The point rejection strategies. . . . .	45
2.7	Demonstrating the use of size functions. . . . .	51
2.8	Meshes produced by our algorithm on the spheres and the torus. . . . .	52
2.9	Meshes produced by our algorithm on the brain and the stomach. . . . .	53
2.10	Meshes produced by our algorithm on the skeleton and the colon. . . . .	54
2.11	Meshes produced by our algorithm on the knee and the head-neck. . . . .	55
3.1	Delaunay mesh generation and refinement. . . . .	57
3.2	Implementation of the local Contention Manager (local-CM). . . . .	67
3.3	Illustration of the local Contention Manager (local-CM). . . . .	69
3.4	A possible livelock and how to avoid it. . . . .	71
3.5	Strong scaling performance. . . . .	76
3.6	Degree of available parallelism. . . . .	80
3.7	Meshes generated by PI2M on the knee and the head-neck atlases. . . . .	87
3.8	Meshes generated by CGAL on the knee and the head-neck atlases. . . . .	88
3.9	Meshes generated by TetGen on the knee and the head-neck atlases. . . . .	89
4.1	Picking region on surfaces. . . . .	92

4.2	Proof of Lemma 4.3, a 3D illustration. . . . .	96
4.3	Flow diagram depicting the relationship among the rules. . . . .	98
4.4	Proof of Lemma 4.8. . . . .	100
4.5	Normalized volume histogram of the output mesh. . . . .	104
4.6	Complexity of the 4D code. . . . .	105

Real-Time High-Quality Image to Mesh Conversion for Finite Element  
Simulations

# Chapter 1

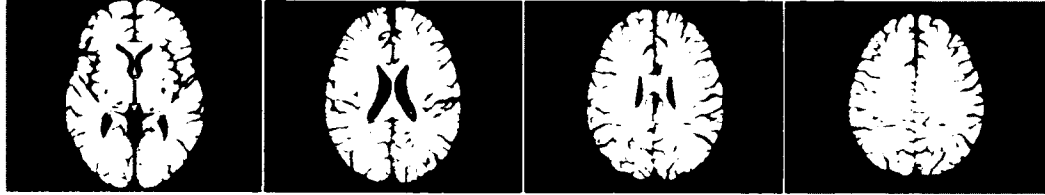
## Introduction

### 1.1 Motivation and Related Work

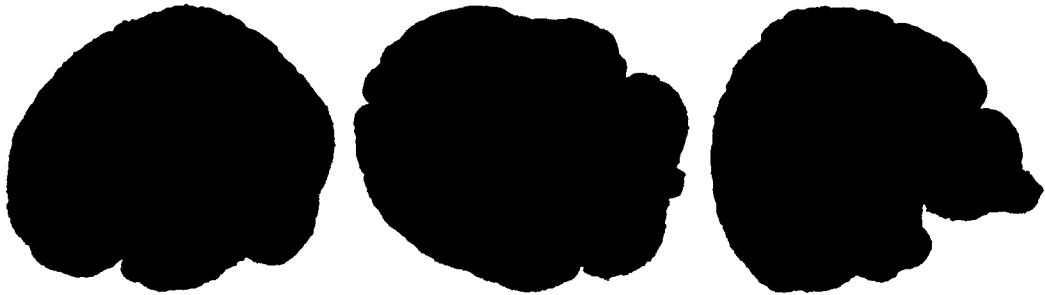
Image-to-mesh (I2M) conversion enables patient-specific Finite Element (FE) modeling in image guided diagnosis and therapy [15, 94]. See Figure 1.1 and Figure 1.2 for a couple of examples. This has significant implications in many areas, such as imaged-guided therapy, development of advanced patient-specific blood flow simulations for the prevention and treatment of stroke, patient-specific interactive surgery simulation for training young clinicians, and study of bio-mechanical properties of collagen nano-straws of patients with chest wall deformities, to name just a few.

Delaunay meshing is a popular technique for generating tetrahedral meshes, since it is able to mesh various domains such as: polyhedral domains [38, 117], domains bounded by surfaces [109, 113], or multi-labeled images [29, 110], offering at the same time mathematical guarantees on the quality and the fidelity of the final mesh.

In the literature, Delaunay refinement techniques have been employed to mesh objects whose surface is already meshed as a *Piecewise Linear Complex* (PLC) [34, 35, 38, 40, 42, 45, 69, 90, 99, 101, 117, 121]. The challenge in this category of techniques is that the quality of the input PLC affects the quality of the final volume mesh. For example, if the input angles of the PLC are small, then even termination might be compromised [120]. For images, one way to alleviate this challenge is to consider the



(a) A few slices of the grayscale BigBrain image (courtesy of BigBrain project).

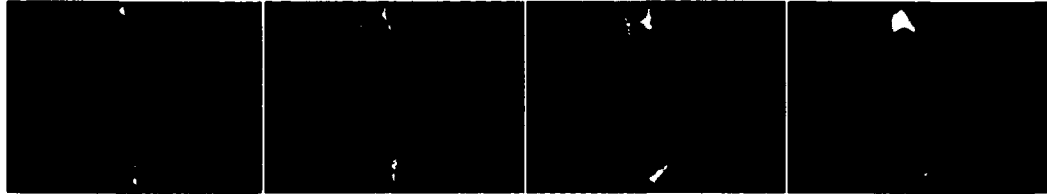


(b) Views of the resulted tetrahedral mesh.

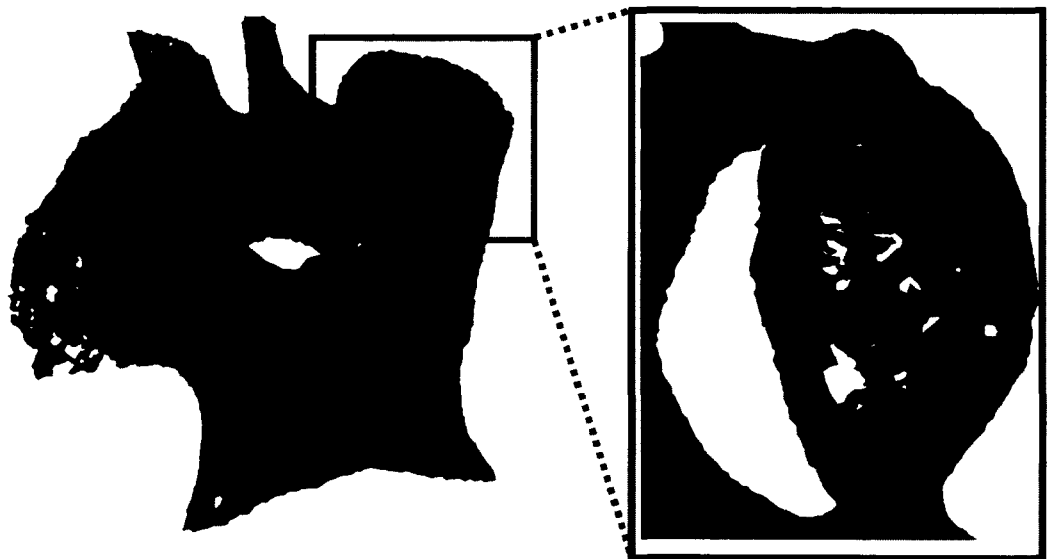
**Figure 1.1:** Image to Mesh Conversion on the high resolution BigBrain data [12] for subsequent FE bio-mechanical modeling.

faces of each outer voxel as the input PLC, since these faces meet at large angles ( $90^\circ$  or  $180^\circ$ ). However, this would result in an unnecessarily large final mesh.

Another approach is to assume that the object  $\Omega$  to be meshed is known only through an implicit function  $f : \mathbb{R}^3 \rightarrow \mathbb{Z}$  such that points in different regions of interest evaluate  $f$  differently. This assumption covers a wide range of inputs used in modeling and simulation, such as parametric surfaces/volumes [109], level-sets, and segmented multi-labeled images [29, 89, 110], the focus of this thesis. If the subsequent simulation permits sharp features of the domain to be rounded-off, such functions can be used to represent PLCs as well [89], a fact that renders this approach quite general. It should be noted that these methods do not suffer from any small input angle artifacts introduced by the initial conversion to PLCs, since the isosurface  $\partial\Omega$  of the object  $\Omega$  is recovered and meshed during refinement. In this work, we deal with objects whose surface is a smooth manifold (see Section 2.1 and Section 4.1). It is the algorithm’s responsibility to mesh both the surface and the interior of the object such that the mesh boundary describes the object surface in a way that meets the predefined fidelity and quality requirements.



(a) A few slices of the grayscale bone image (courtesy of Dr. Xenios Papademetris, Dr. Steven Tommasini, and Dr. Joshua Van Houten, Yale University).



(b) Views of the resulted tetrahedral mesh.

**Figure 1.2:** Image to Mesh Conversion on the micro-CT vertebral body of a mouse for subsequent FE bone modeling and compression analysis.

The quality of an element is traditionally measured in terms of its *circumradius-to-shortest-edge* ratio or *radius-edge* ratio for short. It is desirable that the mesh elements have radius-edge ratio bounded from above. Meshes satisfying that bounded ratio property are called *almost-good* meshes in the literature [91]. Miller *et al.* [99] show that almost-good meshes guarantee optimal convergence rates for approximate solutions of Poisson’s equation.

3D Delaunay volume meshing algorithms extend the popular Delaunay surface meshing and reconstruction algorithms described in [10, 44], and they offer quality and fidelity guarantees [109, 113] under the assumption that the surface of the object is smooth [10, 109] or does not form input angles less than  $90^\circ$  [113]. However, the

quality achieved by these algorithms is somewhat weak: the upper bound for the elements' radius-edge ratio is larger than 4. In contrast, the upper bound guaranteed by our algorithm is  $\sqrt{\sqrt{3} + 2}$  ( $\approx 1.93$ ). To our knowledge, our algorithm is the first volume Delaunay mesher for surfaces achieving such a small radius-edge ratio with these fidelity guarantees.

Almost-good meshes, however, might contain nearly flat elements, the so called *slivers*. The reason is that slivers can have a very small radius-edge ratio and at the same time a very small dihedral angle. In the literature, there are post-processing techniques that given an almost-good mesh, they are able to remove slivers. See for example the work of Li and Teng [91], the *exudation technique* of Cheng *et al.* [34], and the *sliver perturbation* of Tournois *et al.* [81]. In fact, the sliver removal technique of Li and Teng [91] requires a low radius-edge ratio, since the lower the radius-edge ratio, the larger the guaranteed bound on the minimum dihedral angles. This is another motivation for achieving low radius-edge ratio.

The success of Delaunay techniques to approximate the surface relies on the notion of  $\varepsilon$ -samples, first introduced by Amenta and Bern [9]. The construction of  $\varepsilon$ -samples directly from the surface is a challenging task. In the literature, however, it is assumed that either such a sample is known [9–11] or that an initial sparse sample is given on every connected component [28, 109, 113]. In this work, we propose a method that starts directly from labeled images and computes the appropriate sample on the fly.

In the literature, there are also non-Delaunay surface and volume meshing algorithms for 3D images. *Marching Cubes* [96] is a very popular technique for surface meshing; it guarantees, however, neither good quality triangular facets nor faithful surface approximation. Furthermore, since the cubes have a very small size (close to the voxel size), Marching Cubes does not offer a way to control the size of the mesh. Molino *et al.* [102] develop the *Red-Green Mesh* (RGM) method. RGM starts by meshing an initial body-centric cubic (BCC) lattice which is then compressed such that its boundary fit on the surface. RGM gives, however, no quality or surface approximation guarantees. Another lattice-based method is the *Isosurface Stuffing* of Labelle and Shewchuk [89]. They prove that the graded version of the final mesh consists of elements with dihedral angles larger than  $1.66^\circ$ . The *Lattice Decimation*



method proposed by Chernikov and Chrisochoides [41] is guaranteed to produce a good geometric approximation of the underlying object. The meshes are also proved to consist of tetrahedra with good dihedral angles. However, topological faithfulness is not guaranteed. Alliez *et al.* [8] introduce a Delaunay-based optimization technique. Specifically, they iteratively compute the new locations of the points by minimizing a quadratic energy. The connectivity of these points is recalculated by finding their Delaunay triangulation each time. They show that this technique produces meshes that respect the boundary of the domain. Klingner and Shewchuk [85] extend the work of Freitag and Ollivier-Gooch [67] by proposing smoothing and topological transformations which improve the quality of the mesh substantially. The execution time, however, can be very high, even for small mesh size problems.

In this thesis, building upon our 3D sequential code, we also present a 3D Delaunay parallel Image-to-Mesh conversion algorithm (abbreviated as PI2M) that (a) recovers the isosurface of the biological object with geometric and topological guarantees and (b) meshes the underlying volume with tetrahedra of high quality. These two characteristics render our method suitable for subsequent FE analysis, since the robustness and accuracy of the solver rely on the quality of the mesh [69, 71, 119].

PI2M recovers the tissues' boundaries and generates quality meshes through a sequence of dynamic insertion and deletion of points which is computed on the fly and in parallel during the course of refinement. To the best of our knowledge, none of the parallel Delaunay refinement algorithms support point removals. Point removal, however, offers new and rich refinement schemes which are shown in the sequential meshing literature [62, 85] to be very effective in practice.

Our implementation employs low level locking mechanisms, carefully designed contention managers, and well-suited load balancing schemes that not only boost the parallel performance, but they exhibit very little overhead: our single threaded performance is more than 10 times faster than our previous sequential prototype [62, 64] and it is faster than CGAL [6] and TetGen [121], the state of the art optimized sequential open source meshing tools. Specifically, PI2M is consistently 40% faster than CGAL. We also compare PI2M with TetGen [121] and show that PI2M is faster on generating large meshes (i.e., meshes consisting of more than 900,000 tetrahedra) by

35%. Considering the fact that both CGAL and TetGen perform insertions via the Bowyer-Watson kernel [30, 128], as is the case of PI2M, such a comparison is quite insightful.

Parallel Delaunay refinement is a highly irregular and data-intensive application and as such, it is very dynamic in terms of resource management. Implementing an efficient parallel Delaunay refinement would help the community gain insight into a whole family of problems characterized by unpredictable communication patterns [16]. We test and show the effectiveness of PI2M on the cc-NUMA architecture. Demonstrating the performance of mesh refinement on cc-NUMA architectures illuminates the characteristic challenges of irregular applications on the many-core chips featuring dozens of cores. But even the biggest distributed-memory machines consist of groups of cores that, from our application’s point of view and supporting software, can be treated as cc-NUMA. The efficient utilization of such deep architectures can be achieved by employing a tightly-coupled approach inside each group (i.e., the ideas of this thesis), and by being less explorative in the other layers, as we stated in more detail in [46].

Specifically, we used the Pittsburgh Supercomputing Center’s Blacklight, employing BoostC++ threads. Although the ideas of this thesis could be programmed using the more general MPI programming model, we chose threads, since the maintenance of threads is typically faster in shared-memory machines [84].

Experimental evaluation shows a more than 82% strong scaling efficiency for up to 64 cores, and a more than 82% weak scaling efficiency for up to 144 cores, reaching a rate of more than 14.3 million elements per second. We are not aware of any 3D parallel Delaunay refinement method achieving such a performance, on either distributed or shared-memory architectures. However, for a higher core count, our method exhibits considerable performance degradation. We argue that this deterioration is not because of load imbalance or high thread contention, but because of the intensive and hop-wise slower communication traffic involved in increased problem sizes, large memories, and cache coherency protocols. This problem could be potentially alleviated by using hybrid approaches to explore network hierarchies [46, 65]. However, this is outside the scope of the thesis. Our goal is to develop the most efficient and

scalable method on a moderate number ( $\sim 100$ ) of cores. Our long term goal is to increase scalability by exploiting concurrency at different levels [46].

In the parallel mesh generation literature, only PLC-based methods have been considered. That is, either  $\Omega$  is given as an initial mesh [31, 50, 78, 126] or  $\partial\Omega$  is already represented as a polyhedral domain [68, 83, 92, 105]. We, on the contrary, mesh both the volume and the isosurface directly from an image and not from a polyhedral domain. This flexibility offers great control over the trade-off between quality and fidelity: parts of the isosurface of high curvature can be meshed with more elements of better quality. Moreover, our method is able to satisfy both surface and volume custom element densities, as dictated by the user-specified size functions. This is not the case of algorithms that treat the surface voxels as the PLC of the domain [40, 73, 76], since the size of the elements is determined by the voxel spacing, a fact that offers little control over the mesh density. In the future, we also plan to incorporate in our parallel framework the computational intensive smoothing of the mesh boundary for CFD applications, e.g. lung modeling [57, 87, 88].

In our previous work [63], we implemented a parallel *Triangulator* able to support fully dynamic insertions and removals. Our parallel Triangulator, however, has one major limitation: as is the case with all Triangulators [20, 23, 24, 63], it tessellates only the convex hull of a set of points, and it is not concerned with any quality or fidelity constraints imposed by the input geometry and the user. Also, in parallel triangulation literature [20, 23, 24], the pointset, whose convex hull is to be constructed, is static and given before the algorithm starts. In this thesis, we extend our previous work [63], such that the discovery of the dynamically changing set of points, which are being inserted or removed in order to satisfy the quality and fidelity constraints, is performed in parallel as well: a very dynamic process that increases parallel complexity even more. This is neither incremental nor a trivial extension.

There is extensive previous work on parallel mesh generation, including various techniques, such as: Delaunay, Octree, or Advancing Front meshing. Parallel mesh generation/refinement should not be confused with parallel triangulation [20, 23, 24, 63]. Triangulation tessellates the convex hull of a given, static set of points. Mesh generation focuses on element quality and the conformity to the tissues' boundary.

which necessitates the parallel insertion or removal of points which are gradually and concurrently discovered through refinement.

One of the main differences between our method and previous work is that in the literature the surface of the domain is either given as a polyhedron, or the extraction of the polyhedron is done sequentially, or refinement starts from an initial background octree. As explained in this Section above, our method constructs the polyhedral representation of the object’s surface from scratch, and therefore, it adds extra functionality. This surface recovery is also performed in parallel, together with the volume meshing, thus taking advantage of another degree of parallelism.

Given an initial mesh, de Cougny and Shephard [50] dynamically repartition the domain such that every processor has equal work. They also describe “vertex snapping”, a method that can be used for the representation of curved boundaries, but they give no guarantees about the achieved fidelity (both geometrically and topologically).

In our past work [105], we implemented a tightly-coupled method like ours. However, in this thesis, we take extra care to greatly reduce the number of rollbacks (see Section 3.3), and thus achieve scalability for a higher core count. In [39] and [92], our group devised a partially-coupled and a decoupled method for distributed-memory systems based on Medial Axis decomposition. However, Medial Axis decomposition for general 3D domains is a challenging problem and still open. In contrast, the method presented in this thesis does not rely on any domain decomposition, and as such, it is flexible enough to be extended to arbitrary dimensions, a goal that is left for future work. In [43], our group presented a method which allows for safe insertion of points independently without synchronization. Although the method in [43] improves data locality and decreases communication, it exhibits little scalability on more than 8 cores because the initial bootstrapping, needed as a pre-processing step, is performed sequentially and not in parallel.

Kadow [83] starts from a polygonal surface (PSLG) and offers tightly coupled refinement schemes in 2D only. In our case, the polyhedral representation of the object’s surface is performed in parallel, which adds extra functionality and available parallelism. Galtier and George [68] compute a smooth separator and distribute the subdomains to distinct processors. However, the separators they create might not be

Delaunay-admissible and thus they need to restart the process from the beginning. Weatherill *et al.* [114] subdivide the domain into decoupled blocks. Each block then is meshed with considerably less communication and synchronization. Nevertheless, the generated mesh is not Delaunay, a property that is critical to applications like large scale electro-magnetics [116]. A decoupled Delaunay method was also developed by Ivanov *et al.* [79]. The reported speedup is superlinear, but only on very small (eight) core counts and on simplistic geometries.

Tu *et al.* [126] describe a parallel octree method that interacts with the solver in parallel and efficiently, but the fidelity and conformity of the meshes to complex multi-material junctions/interfaces (one of this thesis’s goals) was not their main focus. The work of Zhou *et al.* [131], and the *Forest-of-octrees* method of Burstedde *et al.* [31] offer techniques for fair and efficient data migration and partitioning in parallel. In our application, however, we show that the main bottleneck that hampers scalability is not load imbalance (see Subsection 3.4.1), but the rollbacks (see Section 3.3) and the memory pressure in the switches (see Section 3.4.3). Load balancing and data migration is also used by Okusanya and Peraire [107] to distribute bad elements across processors, but the performance reported is rather low, as the speedup achieved on 8 cores is shown to be less than 2.4. Dawes *et al.* [53] presented a scalable octree-based technique with grading and quality guarantees. Nevertheless, more than 48 cores are needed to surpass the single-core performance of our method.

Ito *et al.* [78] start from an initial mesh and Löhner [95] from a PLC for subsequent parallel mesh generation in advancing front fashion. It should be noted, however, that advancing front methods guarantee neither termination nor good quality meshes. Also, both methods show little scalability for even a small number of cores. Zagaris *et al.* [130] developed a parallel divide and conquer advancing front domain decomposition and volume meshing technique. The reported scalability, however, is limited, because there is no much parallelism available in the top levels of the divide and conquer tree.

Oliker and Biswas [108] employ three different architectures to test the applicability of 2D adaptive mesh refinement. They conclude that unstructured mesh refinement is not suitable for cc-NUMA architectures: irregular communication patterns and lack

of data locality deteriorate performance sometimes even on just 4 cores. In this thesis, we show that this becomes a problem on a much higher core count (more than 144 cores); i.e., with this work, we push the envelop even further. Clearly, this approach has its own limitations, but a highly scalable and efficient NUMA implementation combined with the decoupled and partially coupled approaches we developed in the past can allow us to explore concurrency levels in the order of at least  $10^8$  to  $10^{10}$  [46].

Technological advances in imaging have made the acquisition of 4D medical images feasible [103, 125, 127, 129]. At the same time, pentatope capable FEM solvers [21, 106] operating directly on 4D data have been shown to be effective for advection-diffusion and Navier-Stokes formulations.

In this work, we also describe a 4-dimensional Delaunay mesh algorithm which operates directly on a 4-dimensional image  $\mathcal{I}$ .  $\mathcal{I}$  represents the domain  $\Omega$  to be meshed as the temporal evolution of a 3D object. That is,  $\Omega = \bigcup_{t_i} \Omega_{t_i}$ , where  $\Omega_{t_i}$  is the 3D object at time  $t_i$  (i.e., the  $i^{\text{th}}$  slice of  $\Omega$ ).

We show that the resulting mesh is sliver free consisting of pentatopes whose boundary is a correct approximation of the underlying isosurface  $\partial\Omega = \bigcup_{t_i} \partial\Omega_{t_i}$ . Note that space-time meshing is different from dynamic surface simulations (see [82] and the references therein for example). In those simulations, the isosurface is not known; instead, a tetrahedral mesh is adapted on each time step that describes accurately the free surface dynamics.

One way to solve the space-time 4D problem is to mesh separately each 3D object  $\Omega_{t_i}$  and then connect the elements between two consecutive objects to obtain space-time elements. However, finding such correspondence—which also has to satisfy the quality criteria—is not intuitive, especially when the topology and the geometry of the two objects varies drastically. Alternatively, one could mesh a single object  $\Omega_{t_i}$  and then deform the mesh to match the shape of the other temporal instances. The limitation of this approach is twofold. First, the quality of the deformed mesh might be much worse than the original; second, there is no control over the mesh density across both the spatial and the temporal direction [21], since the mesh size of the original instance determines the size of the rest of the instances.

Space-time meshing methods have already been proposed in the literature [60, 124]. They assume, however, that the evolving object  $\Omega_t$  has the same spatial space across time. Furthermore, the implementation of these techniques is confined to only the  $2D+t$  case (i.e., the space-time elements are tetrahedra). The more general  $3D+t$  meshing has been the focus in [21, 106], but they consider only convex hyper-surfaces such as hyper-cubes or hyper-cylinders. To our knowledge, the method presented in this thesis is the first to address the  $3D+t$  problem where the topology and the geometry of the evolving object may differ substantially through time, and hence, it is allowed to form complex hyper-surfaces.

In the literature [9, 17, 27, 32, 35, 36], it is shown that given a sufficiently dense sample on a surface  $\partial\Omega$ , the restriction of its Delaunay triangulation to  $\partial\Omega$  is a topologically good approximation, or, alternatively, it satisfies the closed-topological-ball property [59]. Their focus, however, was not on volume meshing, but rather, on surface reconstruction. In this thesis, we fill the space-time volume  $\Omega$  with sliver-free pentatopes, such that  $\partial\Omega$  is approximated correctly.

Our algorithm guarantees that the resulted pentatopes are of bounded aspect ratio. We achieve that by generating elements of low radius-edge ratio and by proving the absence of slivers. We clean the mesh from slivers by integrating into our framework the theory presented in [90]. In [90], the surface is given as an already meshed polyhedral domain (i.e., the method in [90] is a PLC-based method), a different problem than ours, since it is our algorithm’s responsibility to mesh both the underlying zero-surfaces and the bounded volume with topological and geometric guarantees.

## 1.2 Contributions

In summary, the contributions of this thesis are the following:

- Development of a 3D Delaunay meshing technique that operates directly on images, samples and meshes the surface and the volume of the represented biological object with quality and fidelity guarantees.

- Development of a high quality and fidelity 3D parallel Delaunay technique able to scale on up to 144 cores exhibiting at the same time the best single-threaded performance, to the best of our knowledge.
- Development of a 4D Delaunay meshing technique able to recover arbitrary space-time isosurfaces and investigation of ways and directions towards a parallel 4D Delaunay meshing refinement.



## Chapter 2

# Guaranteed Quality Tetrahedral Delaunay Meshing for Medical Images

In this chapter, we present a Delaunay refinement algorithm for meshing 3D medical images. Given that the surface of the represented object is a smooth 2-manifold without boundary, we prove that (a) all the tetrahedra of the output mesh have radius-edge ratio less than  $\sqrt{\sqrt{3} + 2}$  ( $\approx 1.93$ ), (b) all the boundary facets have planar angles larger than 30 degrees, (c) the symmetric (2-sided) Hausdorff distance between the object surface and mesh boundary is bounded from above by a user-specified parameter, and (d) the mesh boundary is ambient isotopic to the object surface. The first two guarantees assure that our algorithm produces elements of bounded radius-edge ratio. The last two guarantees assure that the mesh boundary is a good geometric and topological approximation of the object surface. Our method also offers control over the size of tetrahedra in the final mesh. Experimental evaluation of our algorithm on synthetic and real medical data illustrates the theory and shows the effectiveness of our method.

## 2.1 Preliminaries

Let  $\mathcal{I} \subset \mathcal{R}^3$  be the (spatial) domain of a multi-tissue segmented image.  $\mathcal{I}$  is the input of our algorithm that contains the object  $\Omega \subset \mathcal{I}$  to be meshed. We assume that the object is partitioned into a finite number of  $n$  distinct tissues  $\Omega = \bigcup_i^n \Omega_i$ ,  $i = 1, \dots, n$ . Each  $\Omega_i$  defines an interface  $\partial\Omega_i$  that consists of the set of points that lie on the boundary between  $\Omega_i$  and at least one more tissue or the background of the image. The *isosurface*  $\partial\Omega$  of  $\Omega$  is then the collection of all interfaces; that is,  $\partial\Omega = \bigcup_i^n \partial\Omega_i$ ,  $i = 1, \dots, n$ . We assume that we are given a function  $f : \mathcal{I} \rightarrow \{-1, 0, 1, \dots, n\}$ , which classifies every point  $p \in \mathcal{I}$  appropriately. Specifically,  $p$  evaluates  $f$  to  $-1$  if it lies on  $\partial\Omega$ , to  $0$  if it lies in the background (i.e., outside the object), or to a positive integer  $i$  if it belongs to the tissue  $\Omega_i$ . The existence of such a function is a quite reasonable assumption:  $f$  can be constructed or approximated from the image voxels quite well for any segmented image (see Section 2.6 for details on  $f$ 's implementation).

As is generally the case in the literature [10, 28, 109], we also assume that  $\partial\Omega$  is a smooth (twice differentiable) 2-manifold without boundary.

**Definition 2.1 (medial axis, Blum [25])** *The medial axis of  $\partial\Omega$  is the closure of the set of those points having more than one closest point on  $\partial\Omega$ .*

**Definition 2.2 (local feature size, Amenta and Bern [9])** *The local feature size of a point  $p \in \partial\Omega$ , denoted as  $\text{lfs}_{\partial\Omega}(p)$ , is the distance from  $p$  to the medial axis of  $\partial\Omega$ .*

We denote with  $\text{lfs}_{\partial\Omega}^{\text{inf}}$  and  $\text{lfs}_{\partial\Omega}^{\text{sup}}$  the infimum and the supremum of the local feature sizes of all the points on  $\partial\Omega$  respectively, that is:  $\text{lfs}_{\partial\Omega}^{\text{inf}} = \inf\{\text{lfs}_{\partial\Omega}(p) : p \in \partial\Omega\}$  and  $\text{lfs}_{\partial\Omega}^{\text{sup}} = \sup\{\text{lfs}_{\partial\Omega}(p) : p \in \partial\Omega\}$ . Note that since  $\partial\Omega$  is assumed to be a smooth manifold, both  $\text{lfs}_{\partial\Omega}^{\text{inf}}$  and  $\text{lfs}_{\partial\Omega}^{\text{sup}}$  are positive real constants. Another useful property is that the local feature size is 1-Lipschitz, that is,

$$\text{lfs}_{\partial\Omega}(p) \leq |pq| + \text{lfs}_{\partial\Omega}(q). \quad (2.1)$$

**Definition 2.3** ( $\varepsilon$ -sample, Amenta *et al.* [10]) *A point set  $P \subset \partial\Omega$  is called an  $\varepsilon$ -sample of  $\partial\Omega$ , if for every point  $p \in \partial\Omega$  there is a sample point  $q \in P$ , such that  $|pq| \leq \varepsilon \cdot \text{lf}_{\partial\Omega}(p)$ .*

Next, we define a special *restriction*:

**Definition 2.4** (*restricted Delaunay triangulation*, Boissonnat *et al.* [28]) *Let  $\mathcal{D}(P)$  be the Delaunay triangulation of the point set  $P$ . The restriction of  $\mathcal{D}(P)$  to  $\partial\Omega$ , denoted as  $\mathcal{D}_{|\partial\Omega}(P)$ , contains the facets in  $\mathcal{D}(P)$  whose dual Voronoi edges intersect  $\partial\Omega$ .*

We shall refer to a facet whose dual Voronoi edge intersects  $\partial\Omega$  as a *restricted facet*. We denote the Voronoi edge of a facet  $f$  with  $\text{Vor}(f)$ .

In [28], the following useful theorem is proved:

**Theorem 2.1** (Boissonnat *et al.* [28]) *If  $P$  is an  $\varepsilon$ -sample of  $\partial\Omega$  with  $\varepsilon < 0.09$ , then:*

- $\mathcal{D}_{|\partial\Omega}(P)$  is a 2-manifold ambient isotopic to  $\partial\Omega$  and
- the 2-sided Hausdorff distance between  $\mathcal{D}_{|\partial\Omega}(P)$  and  $\partial\Omega$  is  $O(\varepsilon^2)$ .

We next define the *surface ball* of a restricted facet:

**Definition 2.5** (*surface ball*, Oudot *et al.* [109]) *Let  $f$  be a restricted facet and  $e$  be  $f$ 's dual Voronoi edge. The surface ball  $B_{\text{surf}}(f)$  of  $f$  is a closed ball which is centered at a point  $p \in e \cap \partial\Omega$  and passes through  $f$ 's vertices.*

In the rest of the chapter, the center and radius of restricted facet  $f$ 's surface ball  $B_{\text{surf}}(f)$  are denoted by  $c_{\text{surf}}(f)$  and  $r_{\text{surf}}(f)$ , respectively.

The following Remark follows directly from the fact that the center of restricted facet  $f$ 's surface ball lies on its Voronoi edge:

**Remark 2.1** *The surface ball of  $f$  contains no vertices in its interior.*

A real point  $p$  is called a *vertex*, if it has been already inserted into the mesh. Point  $p$  is called a *feature point* (or a *feature vertex*, if  $p$  is inserted into the mesh).

if it is a surface point, i.e.,  $p \in \partial\Omega$ . In the rest of the chapter,  $\text{cfp}(p)$  denotes the *Closest Feature Point* to  $p$ .

An *element*  $t$  is a tetrahedron, a (triangular) facet, or an edge. The *diametral ball*  $B(t)$  of  $t$  is the set of points that lie inside or on  $t$ 's smallest circumscribing sphere. The smallest circumscribing sphere of an element  $t$  will be sometimes called its *diametral sphere* and symbolized by  $S(t)$ . The center of  $t$ 's diametral ball/sphere and the radius of  $t$ 's diametral sphere are denoted by  $c(t)$  and  $r(t)$ , respectively. The shortest edge of element  $t$  is denoted by  $l_{\min}(t)$ . Finally, the radius-edge ratio  $\rho(t)$  of a tetrahedron or facet  $t$  is defined as  $\rho(t) = \frac{|r(t)|}{|l_{\min}(t)|}$ .

## 2.2 Algorithm

The user specifies as input the target upper radius-edge ratio  $\bar{\rho}_t$  for the mesh tetrahedra, the target upper radius-edge ratio  $\bar{\rho}_f$  for the mesh boundary facets, and parameter  $\delta$ . It will be clear in Section 2.5 that the lower  $\delta$  is, the better the mesh boundary will approximate  $\partial\Omega$ . For brevity, the quantity  $\delta \cdot \text{lfs}_{\partial\Omega}(z)$  is denoted by  $\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point.

Our algorithm initially inserts the 8 corners of a cubical box  $B$  that contains the object  $\Omega$ , such that the distance between a point  $p$  on the box and any feature point  $z$  is at least  $2\Delta_{\partial\Omega}(z)$ . Since  $\text{lfs}_{\partial\Omega}(z) \leq \text{lfs}_{\partial\Omega}^{\text{sup}}$ , it suffices to construct  $B$  such that it is separated from the minimum bounding box of  $\Omega$  by a distance of at least  $2 \cdot \delta \cdot \text{lfs}_{\partial\Omega}^{\text{sup}}$ . Let  $d$  be the diagonal of the minimum bounding box of  $\Omega$ . Clearly, constructing box  $B$  to be separated from the minimum bounding box by a distance of at least  $\delta \cdot d$  fulfills the requirement, since  $\text{lfs}_{\partial\Omega}^{\text{sup}}$  cannot be larger than  $\frac{d}{2}$ .

After the computation of this initial triangulation, the refinement starts dictating which extra points (also known as *Steiner* points) are inserted or which vertices are deleted. At any time, the Delaunay triangulation  $\mathcal{D}(V)$  of the current vertices  $V$  is maintained. Note that by construction,  $\mathcal{D}(V)$  always covers the entire object and that any point on the box is separated from  $\partial\Omega$  by a distance of at least  $2\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point.

The users can also define their own customized *Size Function*  $sf : \Omega \mapsto \mathbb{R}^+$  and pass it as input to our mesher. The size function sets an upper bound on the radii of the circumballs of the tetrahedra, and thus offers the flexibility of controlling which parts of the domain need a denser representation.

During the refinement, some vertices are inserted exactly on the box; these vertices are called *box vertices*. The edges that lie precisely on one of the 12 edges of the bounding box are called *box edges*. We further divide the box vertices into two categories: *box-edge vertices* and *non-box-edge vertices*. The former vertices lie precisely on a box edge, while the latter do not. The facets that lie precisely on one of the 6 faces of the box are called *box facets*. For example, the initial triangulation contains just 8 box vertices (which are also box-edge vertices) and 12 box edges (among other edges). Note that the endpoints of a box edge are always box edge vertices, but the opposite is not always true. We shall refer to the vertices that are neither box vertices nor feature vertices as *free vertices*.

Next, we define two types of tetrahedra:

- **intersecting tetrahedra:** tetrahedra whose circumsphere intersects  $\partial\Omega$  (i.e., there is at least one feature point in their circumball), and
- **interior tetrahedra:** tetrahedra whose circumcenter lies (strictly) inside  $\Omega$ .

Note that a tetrahedron might be both intersecting and interior or might belong to neither type.

The algorithm inserts new vertices or removes existing ones for three reasons: to guarantee that the mesh boundary is close to the object surface, to remove tetrahedra or facets with large radius-edge ratio, and to satisfy the sizing requirements. Specifically, let  $t$  be a tetrahedron and  $f$  a facet in  $\mathcal{D}(V)$ ; the following five rules are checked in this order:

- **R1:** Let  $t$  be an intersecting tetrahedron and  $z$  be equal to the Closest Feature Point  $\text{cfp}(c(t))$  of  $t$ 's circumcenter  $c(t)$ . If  $z$  is at a distance not closer than  $\Delta_{\partial\Omega}(z)$  to any other feature vertex, then  $z$  is inserted and all the free vertices closer than  $2\Delta_{\partial\Omega}(z)$  to  $z$  are deleted.

- **R2:** Let  $t$  be an intersecting tetrahedron and  $z$  be equal to  $\text{cfp}(c(t))$ . If  $r(t) \geq 2 \cdot \Delta_{\partial\Omega}(z)$ , then  $c(t)$  is inserted.
- **R3:** Let  $f$  be a restricted facet. If either  $\rho(f) \geq \bar{\rho}_f$  or a vertex of  $f$  is not a feature vertex, then  $z = c_{\text{surf}}(f)$  is inserted. All the free vertices closer than  $2\Delta_{\partial\Omega}(z)$  to  $z$  are deleted.
- **R4:** If  $t$  is an interior tetrahedron whose radius-edge ratio is larger than or equal to  $\bar{\rho}_t$ , then  $c(t)$  is inserted.
- **R5:** Let  $t$  be an interior tetrahedron. If  $|r(t)| \geq \text{sf}(c(t))$ , where  $\text{sf}(\cdot)$  is the user-defined Size Function, then  $c(t)$  is inserted.

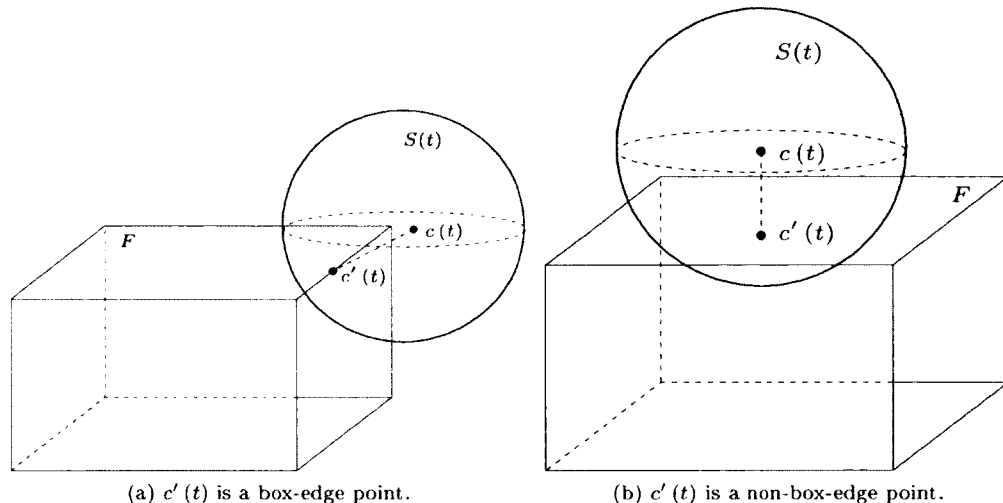
Whenever there is no simplex for which R1, R2, R3, R4, or R5 apply, the refinement process terminates. *The final mesh reported is the set of tetrahedra whose circumcenters lie inside  $\Omega$  (i.e., interior tetrahedra).* Thereafter, the final mesh is denoted by  $\mathcal{M}$ .

**Definition 2.6 (Mesh boundary)** *Let  $f$  be a facet of the final mesh  $\mathcal{M}$ . Consider its two incident tetrahedra. If one tetrahedron has a circumcenter lying inside a tissue  $\Omega_i$  and the other tetrahedron has a circumcenter lying either outside  $\Omega_i$  or on  $\partial\Omega_i$ , then  $f$  belongs to the mesh boundary  $\partial\mathcal{M}$ .*

In Section 2.5, we prove that  $\partial\mathcal{M}$  meshes all multi-tissue interfaces  $\bigcup_i^n \partial\Omega_i (= \partial\Omega)$ , see Section 2.1) accurately, in both geometric and topological sense.

To prove termination (see Section 2.3), no vertices should be inserted outside the bounding box. Notice, however, that vertices inserted due to R2 may lie outside the bounding box. To deal with such cases, we propose special **projections rules**. Their goal is to reject points lying outside the box and insert other points exactly on the box. They are simple to implement, computationally inexpensive, and do not compromise either quality or fidelity. Note that the projection rules are different than the traditional *encroachment rules* described in [38, 117, 118].

Specifically, assume that R2 is triggered for a (intersecting) tetrahedron  $t$  and  $c(t)$  lies outside the box. In that case,  $c(t)$  is rejected for insertion. Instead, its projection



**Figure 2.1:** The projection rule. The circumcenter  $c(t)$  of a tetrahedron  $t$  (not shown) does not lie inside the box.  $c(t)$  is rejected for insertion; rather, its projection  $c'(t)$  precisely on the box is computed and inserted into the triangulation.

$c'(t)$  on the box is inserted in the triangulation. That is,  $c'(t)$  is the closest to  $c(t)$  box point. Notice that  $c'(t)$  can either lie exactly on a box edge (see Figure 2.1a) or in the interior of a box facet (see Figure 2.1b).

Recall that tetrahedra with circumcenters on  $\partial\Omega$  or outside  $\Omega$  are not part of the final mesh, and that is why rules R4 and R5 do not check them.

Algorithm 1 summarizes our mesh generation algorithm. Observe that at line 10, we ask for the closest feature point  $\text{cfp}(c)$  of a given circumcenter  $c$ . Also, given a feature point  $z \in \partial\Omega$ , the algorithm asks for its distance  $\text{lfs}_{\partial\Omega}(z)$  from the medial axis. The computation of  $\text{cfp}(\cdot)$  and  $\text{lfs}_{\partial\Omega}(\cdot)$  is explained in detail in Section 2.6. In the next section, we will prove that *Intersecting*  $\cup$  *Interior* eventually will run out of elements, and the algorithm terminates.

## 2.3 Proof of Quality

In this section, we prove that if the target upper bound  $\bar{\rho}_t$  for the radius-edge ratio is no less than  $\sqrt{\sqrt{3} + 2}$ , then our algorithm terminates outputting tetrahedra with

---

**Algorithm 1: The mesh generation algorithm.**

---

```

1 Algorithm: Refine( $\mathcal{I}, \delta, \bar{\rho}_t, \bar{\rho}_f, sf(\cdot)$ )
2 Input :  $\mathcal{I}$  is the image containing  $\Omega$ ,
           $\delta$  is the parameter that determines how dense the surface sampling will be,
           $\bar{\rho}_t$  ( $\geq \sqrt{\sqrt{3}+2}$ ) is the target radius-edge ratio for the tetrahedra,
           $\bar{\rho}_f$  ( $\geq 1$ ) is the target radius-edge ratio for the facets,
           $sf(\cdot)$  is the size function.
3 Output: A Delaunay mesh  $\mathcal{M}$  that approximates  $\partial\Omega$  well and is composed of tetrahedra with radius-edge ratio less than  $\bar{\rho}_t$  and
          boundary facets with planar angles larger than  $30^\circ$ .
4 Let  $V$  be the set of vertices inserted into the triangulation;
5 Let  $\mathcal{D}(V)$  be the triangulation of the set  $V$ ;
6 Let Intersecting and Interior be the set of the intersecting and interior tetrahedra in  $\mathcal{D}(V)$ , respectively;
  /* At this point, all the above sets are equal to the empty set. */
7 Insert the 8 vertices of a cubical box which contains  $\Omega$  such that any point inserted on the box is separated from any point  $z \in \partial\Omega$  by a
  distance of at least  $2 \cdot \delta \cdot lfs_{\partial\Omega}(z)$ ;
8 Update  $V, \mathcal{D}(V),$  Intersecting, and Interior;
9 while Intersecting  $\cup$  Interior  $\neq \emptyset$  do
10   if Intersecting  $\neq \emptyset$  then
11     Pick a tetrahedron  $t \in$  Intersecting;
12     Compute steiner =  $cfp(c(t))$ ;
13     if there is no feature vertex closer than  $\delta \cdot lfs_{\partial\Omega}(\textit{steiner})$  to steiner then /* R1 applies. */
14       else
15         if  $r(t) \geq 2 \cdot \delta \cdot lfs_{\partial\Omega}(\textit{steiner})$  then
16           Compute steiner =  $c(t)$ ; /* R2 applies. */
17           if steiner lies outside the box then
18             Compute steiner =  $c^f(t)$ ; /* Projection rules apply. */
19           end
20         else
21           if  $t$  is adjacent to a restricted facet  $f$ , such that  $\rho(f) \geq \bar{\rho}_f$  or  $f$ 's vertices do not lie on  $\partial\Omega$  then /* Since  $f$  is a
22             restricted facet,  $f$  is necessarily incident to at least one intersecting tetrahedron  $t$ . */
23             Compute steiner =  $c_{surf}(f)$ ; /* R3 applies. */
24           else
25             Intersecting = Intersecting  $\setminus t$ ;
26             continue; /* No steiner point found. */
27           end
28         end
29       end
30     else
31       if  $\rho(t) \geq \bar{\rho}_t$  or  $r(t) \geq sf(c(t))$  then /* Interior cannot be empty. */
32         Compute steiner =  $c(t)$ ; /* R4 or R5 apply. */
33       else
34         Interior = Interior  $\setminus t$ ;
35         continue; /* No steiner point found. */
36       end
37     end
38   Insert steiner;
39   if steiner is a feature vertex then
40     Delete all the free vertices that are closer than  $2 \cdot \delta \cdot lfs_{\partial\Omega}(\textit{steiner})$  to steiner.
41   end
42   Update  $V, \mathcal{D}(V),$  Intersecting, and Interior;
43 end
44 Let the final mesh  $\mathcal{M}$  be equal to the set of the tetrahedra in  $\mathcal{D}(V)$  whose circumcenter lies inside  $\Omega$ ;

```

---

radius-edge ratio less than  $\bar{\rho}_t$  and boundary facets with planar angles larger than  $30^\circ$  (see Theorem 2.2). Note that termination and quality are not compromised by any positive value of  $\delta$ . Parameter  $\delta$  affects only the fidelity guarantees (see Section 2.5).

Suppose that an element (tetrahedron or facet)  $t$  violates a rule  $R_i$ , where  $i = 1, 2, 3, 4, 5, \text{proj}$ , where  $R_{\text{proj}}$  denotes the projection rules. That is, if  $t$  violates  $R_2$ , but its circumcenter lies on or outside the box, then we say that  $t$  violates  $R_{\text{proj}}$  instead.  $t$  is called an  $R_i$  element.  $R_i$  dictates the insertion of a point  $p$  (and possibly the removal of free points). Point  $p$  is called an  $R_i$  point. Although the initial 8 box corners inserted into the triangulation do not violate any rule, we shall refer to these corners as  $R_{\text{proj}}$  vertices as well.

Following similar terminology to [117, 118], we next define the *insertion radius* and the *parent* of a point  $p$ .



**Definition 2.7 (Insertion radius)** *Let  $v$  be a vertex inserted into the triangulation. Right after the insertion of  $v$  (i.e., before any potential vertex removals), the insertion radius  $R(v)$  of  $v$  is equal to  $|vq|$ , where  $q$  is:*

- *$v$ 's closest box vertex already inserted into the mesh, if  $v$  is a box vertex,*
- *$v$ 's closest feature vertex already inserted into the mesh, if  $v$  is an  $R1$  vertex,*
- *$v$ 's closest vertex already inserted into the mesh, otherwise.*

**Definition 2.8 (Parent)** *Let  $v$  be an  $Ri$  vertex inserted into the mesh because an element (tetrahedron or facet)  $t$  violated  $Ri$ . The parent  $Par(v)$  of  $v$  is:*

- *an arbitrary box vertex, if  $t$  is a facet incident to at least one box vertex,*
- *the most recently inserted vertex of  $t$ , if  $t$  is a facet with  $\rho(f) < \bar{\rho}_f$ ,*
- *the most recently inserted vertex of  $l_{\min}(t)$ , otherwise.*

The following two Lemmata relate the insertion radii of a vertex  $v$  with the distance between  $v$  and its neighbors.

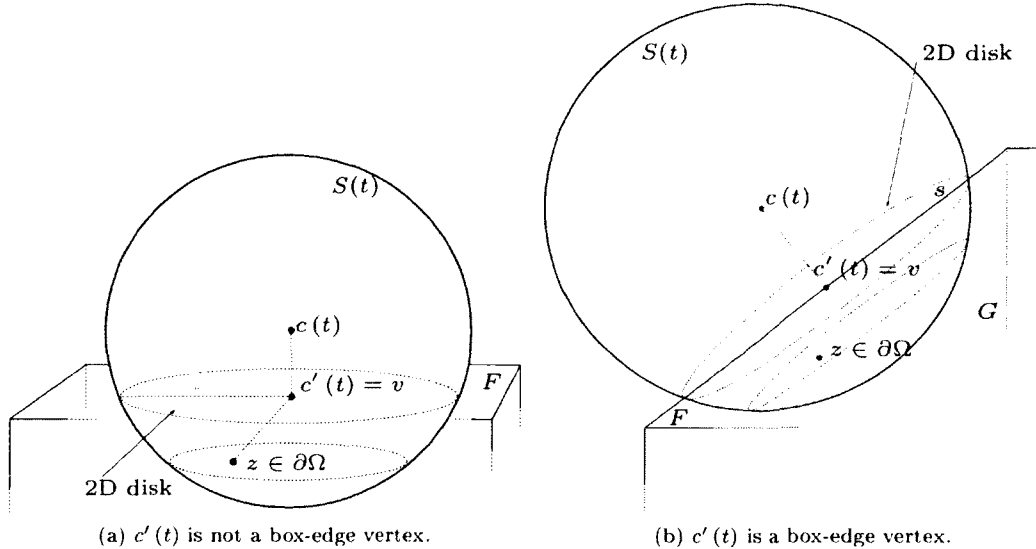
**Lemma 2.1** *Let  $w$  be an  $R2$ ,  $R3$ ,  $R4$ , or an  $R5$  vertex inserted into the triangulation and let  $x$  be an arbitrary vertex already in the triangulation. Then,  $R(w) \leq |wx|$ .*

**Proof:** According to Definition 2.7,  $R(w)$  is the distance between  $w$  and its closest neighbor, say  $q$ . Therefore,  $R(w) = |wq| \leq |wx|$ . ■

**Lemma 2.2** *Let  $w$  be an  $R1$  vertex inserted into the triangulation and let  $x$  be an arbitrary feature vertex already in the triangulation. Then,  $R(w) \leq |wx|$ .*

**Proof:** According to Definition 2.7,  $R(w)$  is the distance between  $w$  and its closest feature vertex, say  $q$ . Therefore,  $R(w) = |wq| \leq |wx|$ . ■

**Lemma 2.3** *Let  $v$  be a box vertex. Then,  $R(v) \geq 2\Delta_{\partial\Omega}(z)$ , where  $z$  is the closest feature point of  $v$ .*



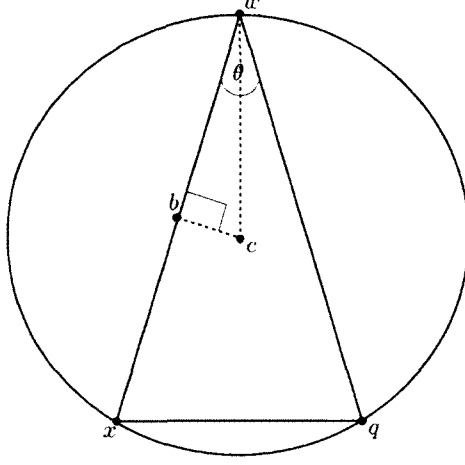
**Figure 2.2:** Illustration to the proof of Lemma 2.3.

**Proof:** According to Definition 2.7,  $R(v)$  is the distance between  $v$  and its closest box vertex.

Initially, only the 8 box vertices of the bounding box are inserted. By construction, no matter the order they are inserted, no box point is closer than  $2\delta\text{lfs}_{\partial\Omega}^{\text{sup}} \geq 2\delta\text{lfs}_{\partial\Omega}(z)$  for any  $z \in \partial\Omega$ . Therefore, the initial edges are definitely larger than  $4\Delta_{\partial\Omega}(z)$  for any  $z \in \partial\Omega$ , and the statement holds.

During the course of refinement, a box point  $v$  is inserted either because the circumcenter  $c(t)$  of an intersecting tetrahedron  $t$  lies on or outside the box. According to the projection rules,  $c(t)$  is ignored, and its projection  $c'(t)$  is inserted instead.

See Figure 2.2 for two examples illustrating the insertion of a non box-edge vertex and a box-edge vertex. In both cases, consider the 2D disk (drawn in both Figure 2.2a and Figure 2.2b) of the  $t$ 's sphere  $S(t)$  that contains  $c'(t)$  and is perpendicular to the segment  $c(t)c'(t)$ . This disk partitions  $t$ 's circumball in two parts: the *upper* part that contains  $c(t)$  and the *lower* part that intersects the interior of the box. From the empty ball property, we know that the insertion radius of  $c'(t)$  cannot be less than the radius of the 2D disk. Let  $z$  be the closest feature point to  $c'(t)$ . Since  $t$  is an intersecting tetrahedron,  $z$  has to lie in the lower part of  $t$ 's circumball, which



**Figure 2.3:** Illustration to the proof of Lemma 2.5. The facet  $f$  is shown in bold together with its circumcircle. The radius of the circumcircle of  $f$  is bounded from below by  $\frac{|wx|}{\sqrt{4 - \frac{1}{\bar{\rho}_f^2} + 2}}$ , where  $wx$  is the second smallest edge of  $f$ .

means that  $R(c'(t)) \geq |c'(t)z|$ . By construction, however,  $|c'(t)z|$  is larger than  $2\delta \text{lf}s_{\partial\Omega}^{\text{sup}} \geq 2\Delta_{\partial\Omega}(z)$ , and the proof is complete. ■

**Lemma 2.4** *Let  $v$  be a box vertex. Then  $|vcfp(v)| \leq R(v)\sqrt{3}$ .*

**Proof:** If  $v$  is a box vertex inserted because the circumcenter of an R2 element lies on or outside the box, then the statement holds, because the proof of Lemma 2.3 directly suggests that  $|cfp(v)v| \leq R(v)$ .

Consider the case where the box vertex  $v$  is one of the initially inserted 8 box corners. Note that the circumballs of all the resulting tetrahedra are the same with the circumscribed ball of the box. Let us denote with  $r$  the length of the radius of that ball. Since that ball contains the whole box, we have that  $|vcfp(v)| \leq 2r$ . It is easy to show that  $r = \frac{\sqrt{3}}{2}L$ , where  $L$  is the box's edge length edge. From Definition 2.7, we know that  $R(v) = L$ , and therefore, we obtain that  $|vcfp(v)| \leq 2r = L\sqrt{3} = R(v)\sqrt{3}$ . ■

**Lemma 2.5** *Let  $f$  be a facet and  $wx$  be its second smallest edge. If  $\rho(f) \leq \bar{\rho}_f$ , then*

$$|r(f)| \geq \frac{|wx|}{\sqrt{4 - \frac{1}{\bar{\rho}_f^2} - 2}}.$$

**Proof:** Let  $f$  be a facet defined by the vertices  $w, x$ , and  $q$ . Assume that  $xq = l_{\min}(f)$  and  $wq$  is the largest edge. If there are more than one smallest edges and/or largest edges, choose arbitrarily one edge as the smallest and/or one edge as the largest. In this example,  $wx$  is the second smallest edge of  $f$ .

Keeping the circumball fixed, move  $w$  along the circumcircle such that the length of  $wx$  becomes equal to the length of  $wq$ . In such a configuration (which is shown in Figure 2.3), the length of  $wx$  (i.e., the length of the second smallest edge) is maximized. Also observe, the radius  $wc$  bisects angle  $\theta$ . From the right triangle  $\triangle cbw$ , we get that  $\cos \frac{\theta}{2} = \frac{|wx|}{2|r(f)|}$ . Since  $\theta$  is still the angle opposite to  $f$ 's shortest edge,  $\theta$  and  $\rho(f)$  are related through the following equality:  $\sin \theta = \frac{1}{2\rho(f)}$  [118]. This

fact and basic trigonometry yield that  $\cos \frac{\theta}{2} = \frac{\sqrt{\sqrt{1 - \frac{1}{4\rho(f)^2}} + 1}}{\sqrt{2}}$ . Note that the right-hand side is maximized when  $\rho(f)$  gets its highest value. Since the Lemma assumes that

$\rho(f) \leq \bar{\rho}_f$ , we obtain that  $\cos \frac{\theta}{2} \leq \frac{\sqrt{\sqrt{1 - \frac{1}{4\bar{\rho}_f^2}} + 1}}{\sqrt{2}}$ . Therefore, we finally conclude that  $|r(f)| \geq \frac{|wx|}{\sqrt{\sqrt{4 - \frac{1}{\bar{\rho}_f^2}} + 2}}$ . ■

The following lemma sets a lower bound on the shortest edge introduced into the mesh after the insertion of a point according to the five rules.

**Lemma 2.6** *Let  $v$  be inserted as dictated by the five rules and  $w$  be its parent. Then,*

- $R(v) \geq \Delta_{\partial\Omega}(z)$ , if  $v$  is an R1 or an R2 vertex, where  $z$  is the closest feature point to  $v$ ,
- $R(v) \geq \min \left\{ \frac{R(w)}{\sqrt{\sqrt{4 - \frac{1}{\bar{\rho}_f^2}} + 2}}, \bar{\rho}_f R(w), \Delta_{\partial\Omega}(v) \right\}$ , if  $v$  is an R3 vertex and  $w$  is a free or a box vertex,
- $R(v) \geq \min \left\{ \bar{\rho}_f R(w), \Delta_{\partial\Omega}(w) \right\}$ , if  $v$  is an R3 vertex and  $w$  is a feature vertex,
- $R(v) \geq \bar{\rho}_t R(w)$ , if  $v$  is an R4 vertex,
- $R(v) \geq sf(v)$ , if  $v$  is an R5 vertex.

**Proof:** We separate cases according to the type of  $v$ .

- **Case 1:**  $v$  is an R1 or an R2 vertex.

If R1 is triggered, then  $v$  is equal to  $z$ . According to Definition 2.7,  $R(v)$  is the distance between  $v$  and its closest feature vertex. Vertex  $v$ , however, is inserted only if  $v$  is separated from any other feature vertex by a distance of at least  $\Delta_{\partial\Omega}(v) = R(v)$ , and the statement holds.

Otherwise, R2 applies for a tetrahedron  $t$  and  $v$  is equal to  $c(t)$ . According to Definition 2.7,  $R(v)$  is the distance between  $v$  and its closest neighbor. Because of the empty ball property,  $R(v)$  is at least  $r(t) \geq 2\Delta_{\partial\Omega}(\text{cfp}(v))$ , and the statement holds.

- **Case 2:**  $v$  is an R3 vertex.

In this case,  $v$  is equal to the center  $c_{\text{surf}}(f)$  of  $f$ 's surface ball, where  $f$  is the restricted facet that violates R3. According to Definition 2.7,  $R(v)$  is the distance between  $v$  and its closest neighbor. Since any surface ball is empty of vertices in its interior (Remark 2.1), we know that  $R(v) = |r_{\text{surf}}(f)|$ . The rest of this proof attempts to bound  $|r_{\text{surf}}(f)|$  from below. We separate three scenarios:

(a) First, consider the case where  $f$  is incident to at least on box vertex. According to Definition 2.8, this box vertex can be the parent  $w$  of  $v$ . By construction, the distance between  $c_{\text{surf}}(f)$  and  $w$  is at least  $2\Delta_{\partial\Omega}(z)$  for any feature point  $z$ . Therefore, the surface radius is at least  $2\Delta_{\partial\Omega}(v) > \Delta_{\partial\Omega}(v)$ , and the statement holds.

(b) Second, consider the case where  $\rho(f) \geq \bar{\rho}_f$ . According to Definition 2.8,  $w$  is the most recently inserted vertex incident to  $wq = l_{\min}(f)$ . Since  $\rho(f)$  is no less than  $\bar{\rho}_f$ ,  $|r(f)| = \rho(f) |l_{\min}(f)| \geq \bar{\rho}_f |l_{\min}(f)|$ .

If  $w$  is not an R1 vertex, from Lemma 2.1, we get that  $|r(f)| \geq \bar{\rho}_f |l_{\min}(f)| \geq \bar{\rho}_f R(w)$ .

If  $w$  is an R1 vertex and  $q$  is a feature vertex (that is,  $q$  is either an R1 or an R3 vertex), then from Lemma 2.2, we get that  $|r(f)| \geq \bar{\rho}_f |l_{\min}(f)| \geq \bar{\rho}_f R(w)$ .

If  $w$  is an R1 vertex and  $q$  is a free vertex then  $q$  has to be separated from  $w$  by a distance of at least  $2\Delta_{\partial\Omega}(w)$ , because R1 deleted all the free points closer

than  $2\Delta_{\partial\Omega}(w)$  to  $w$ . That means that  $|wq| \geq 2\Delta_{\partial\Omega}(w)$ , a fact that also bounds  $|r(t)|$  from below by  $\frac{|wq|}{2} \geq \Delta_{\partial\Omega}(w)$ .

(c) Lastly, consider the case where  $f$ 's radius-edge ratio is less than  $\bar{\rho}_f$ . Since R3 is triggered,  $f$  has to be incident to at least one free vertex  $q$ . According to Definition 2.8,  $w$  is the most recently inserted vertex of  $f$ . If  $w$  is a feature vertex (i.e.,  $w$  is either an R1 or an R3 vertex), then  $q$  must be separated from  $w$  by a distance of at least  $2\Delta_{\partial\Omega}(w)$ , because  $w$  was inserted after  $q$ , and by R1 and R3, all the free points closer than  $2\Delta_{\partial\Omega}(w)$  to  $w$  were deleted. Since  $wq$  is an edge of  $f$ , the radius of any surface ball of  $f$  has to be at least  $\frac{2\Delta_{\partial\Omega}(w)}{2} = \Delta_{\partial\Omega}(w)$  in length. Otherwise,  $w$  is in fact a free vertex (i.e., it is an R2, R4, or R5 vertex).

Any vertex  $w$  of  $f$  is incident to  $f$ 's shortest edge (say  $L_1$ ), or  $f$ 's second shortest edge (say  $L_2$ ), or both. From Lemma 2.5, we have that  $|r(f)| \geq \frac{|L_2|}{\sqrt{\sqrt{4-\frac{1}{\bar{\rho}_f^2}}+2}} \geq \frac{|L_1|}{\sqrt{\sqrt{4-\frac{1}{\bar{\rho}_f^2}}+2}}$ . From Lemma 2.1, we finally get that:  $|r(f)| \geq \frac{R(w)}{\sqrt{\sqrt{4-\frac{1}{\bar{\rho}_f^2}}+2}}$ .

- **Case 3:**  $v$  is an R4 vertex.

There has to be a tetrahedron  $t$  that violates R4, and therefore,  $|r(t)| \geq \bar{\rho}_t |l_{\min}(t)|$ . According to Definition 2.7,  $R(v)$  is the distance between  $v$  and its closest neighbor. Because of the empty ball property,  $R(v) = |r(t)| \geq \bar{\rho}_t |l_{\min}(t)|$ . According to Definition 2.8, the parent  $w$  of  $v$  is the most recently inserted vertex of  $wq = l_{\min}(t)$ .

If  $w$  is not an R1 vertex, from Lemma 2.1, we get that  $\bar{\rho}_t |l_{\min}(t)| \geq \bar{\rho}_t R(w)$ .

If  $w$  is an R1 vertex and  $q$  is a feature vertex (that is,  $q$  is either an R1 or an R3 vertex), then from Lemma 2.2, we get that  $\bar{\rho}_t |l_{\min}(t)| \geq \bar{\rho}_t R(w)$ .

If  $w$  is an R1 vertex and  $q$  is a free vertex then  $q$  has to be separated from  $w$  by a distance of at least  $2\Delta_{\partial\Omega}(w)$ , because R1 deleted all the free points closer than  $2\Delta_{\partial\Omega}(w)$  to  $w$ . That means that  $|wq| \geq 2\Delta_{\partial\Omega}(w)$ , a fact that also bounds  $|r(t)|$  from below by  $\frac{|wq|}{2} \geq \Delta_{\partial\Omega}(w)$ .

- **Case 4:**  $v$  is an R5 vertex.

$v$  is the circumcenter of a tetrahedron  $t$  with radius no less than  $\text{sf}(c(t)) = \text{sf}(v)$ . According to Definition 2.7 and the empty ball property, however, the radius of  $t$  is equal to  $R(v)$ . ■

The next Lemma shows that the boundary facets of the output mesh are in fact restricted facets.

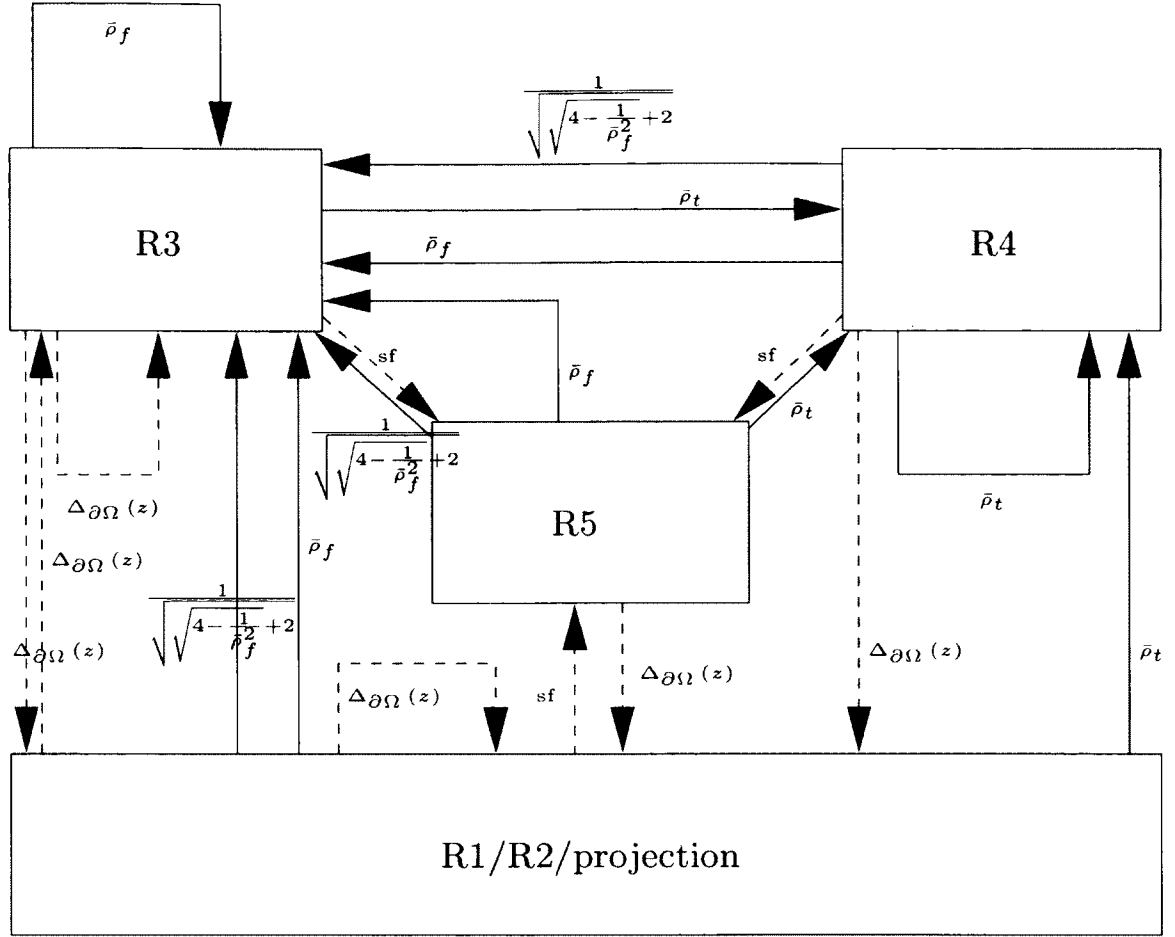
**Lemma 2.7** *Let  $V$  be the set of vertices inserted into the triangulation. The set  $\partial\mathcal{M}$  of the boundary facets of the final mesh  $\mathcal{M}$  is a subset of  $\mathcal{D}_{|\partial\Omega}(V)$ .*

**Proof:** It follows directly from Definition 2.6. A facet  $f$  is a facet of the mesh boundary, if it is incident upon a tetrahedron  $t_1$  whose circumcenter lies inside  $\Omega_i$  (see Section 2.1) and upon a tetrahedron  $t_2$  whose circumcenter lies either outside  $\Omega_i$  or on its surface  $\partial\Omega_i$ . However, this means that the dual Voronoi edge  $e$  of  $f$  intersects  $\partial\Omega_i$ , and as a subsequence,  $e$  also intersects  $\partial\Omega$  ( $\supseteq \partial\Omega_i$ ). Hence,  $f$  belongs to  $\mathcal{D}_{|\partial\Omega}(V)$ . ■

**Theorem 2.2 (Termination and quality)** *Let  $\bar{\rho}_f \geq 1$  and let  $\bar{\rho}_t \geq \sqrt{\sqrt{4 - \frac{1}{\bar{\rho}_f^2}} + 2}$  ( $\geq \sqrt{\sqrt{3} + 2} \approx 1.93$ ). The algorithm terminates producing tetrahedra of radius-edge ratio less than  $\bar{\rho}_t$  and boundary facets of planar angles larger than  $30^\circ$ .*

**Proof:** Figure 2.4 shows the insertion radius of the inserted point as a fraction of the insertion radius of its parent, as proved in Lemma 2.3 and Lemma 2.6. An arrow from  $R_i$  to  $R_j$  with label  $x$  implies that the insertion radius of an  $R_j$  point  $v$  is at least  $x$  times larger than the insertion radius of its  $R_i$  parent  $w$ . The label of the dashed arrows is the absolute value of  $R(v)$ , with  $\text{sf}$  denoting that the insertion radius of  $v$  is no less than  $\text{sf}(v)$ . Note that the labels of the dashed arrows depend on the local feature size of  $\partial\Omega$  and the size function  $\text{sf}$ , and as such are always positive constants.

Recall that during refinement, free vertices might be deleted (because of R1 or R3). Nevertheless, such deletions of vertices are always preceded by insertion of feature points. Considering the fact that feature vertices are never deleted from the



**Figure 2.4:** Flow diagram depicting the relationship among the insertion radii of the vertices inserted because of the rules, where the arrows point from parents to their offspring. A solid arrow from  $R_i$  to  $R_j$  with label  $x$  implies that the insertion radius of an  $R_j$  point  $v$  is at least  $x$  times larger than the insertion radius of its  $R_i$  parent  $w$ . The label of the dashed arrows is the absolute value of  $R(v)$ . No solid cycle should have a product less than 1. The dashed arrows break the cycle.

mesh, termination is guaranteed if we prove that the insertion radii of the inserted vertices cannot approach zero. Clearly [117, 118], it is enough to prove that Figure 2.4 contains no solid cycle of product less than 1. By requiring  $\bar{\rho}_f$  to be no less than 1 (cycle  $R3 \rightarrow R3$ ) and  $\bar{\rho}_t$  to be no less than  $\sqrt{4 - \frac{1}{\bar{\rho}_f^2} + 2} \geq \sqrt{\sqrt{3} + 2}$  (cycle  $R3 \rightarrow R4 \rightarrow R3$ ), no solid cycle of Figure 2.4 has product less than 1, and termination is guaranteed.

Upon termination, the tetrahedra reported as part of the mesh have circumcenters that lie inside  $\Omega$  and therefore they cannot be skinny, because otherwise R4 would apply. This implies that any mesh tetrahedron has radius-edge ratio less than  $\bar{\rho}_t$ .

Since a boundary facet  $f$  is a restricted facet (by Lemma 2.7), R3 guarantees that the radius-edge ratio  $\rho(f)$  of  $f$ 's diametral ball cannot be larger than or equal to  $\bar{\rho}_f$ . Also, note that  $\rho(f)$  is equal to  $\frac{1}{2\sin\theta}$ , where  $\theta$  is the smallest angle of  $f$  [118]. It



follows that the planar angles are larger than  $30^\circ$ . ■

## 2.4 Proof of Good Grading

In this Section, we show that the shortest edge connected to an inserted vertex  $v$  is proportional to  $v$ 's local feature size. Although good grading implies size optimality in 2D [100], the same does not hold in 3D. Nevertheless, it is useful to show that parts of the domain with large local feature sizes are meshed with larger and fewer elements than parts of lower local feature sizes. We also wish to show that dense size functions on certain parts will not affect considerably the density of vertices on other parts of the domain.

Following similar terminology to [109], we define the *general local feature size* and the *general size function* on a vertex  $v$  as follows:

**Definition 2.9 (General Local Feature Size)** *The general local feature size  $glfs_{\partial\Omega}(v)$  on a vertex  $v$  is defined as*

$$glfs_{\partial\Omega}(v) = \inf_{z \in \partial\Omega} \{ |vz| + lfs_{\partial\Omega}(z) \} \quad (2.2)$$

**Definition 2.10 (General Size Function)** *The general size function  $gsf(v)$  on a vertex  $v$  is defined as*

$$gsf(v) = \inf_{p \in \Omega} \{ |pv| + sf(p) \} \quad (2.3)$$

The definition of  $glfs_{\partial\Omega}(\cdot)$  implies that vertices far from  $\partial\Omega$  will tend to have large general local feature sizes. In the case where the vertex lies on the surface, the general local feature size coincides with the local feature size (see Definition 2.2) of the vertex, and it increases when the vertex lies far from the medial axis. The definition of  $gsf(\cdot)$  implies that vertices close to parts of the domain on which the user-defined size function is small will evaluate the general size function to a small number as well.

The following Remark states a few useful properties of the general local feature size and the general size function:

**Remark 2.2 (From [8, 98])**  $glfs_{\partial\Omega}(\cdot)$  and  $gsf(\cdot)$  are 1-Lipschitz. Moreover,  $glfs_{\partial\Omega}(z) = lfs_{\partial\Omega}(v)$ ,  $\forall z \in \partial\Omega$ .

Following the terminology of [117, 118], the density  $D(v)$  of a vertex  $v$  is defined as:

$$D(v) = \frac{\min\{glfs_{\partial\Omega}(v), gsf(v)\}}{R(v)} \quad (2.4)$$

Our goal is to bound from above the density of all inserted vertices by a constant depending only on  $\Omega$  and the input parameters. Notice that since  $D(v) \leq \frac{glfs_{\partial\Omega}(v)}{R(v)}$  and  $D(v) \leq \frac{gsf(v)}{R(v)}$ , it is enough to bound from above either  $\frac{glfs_{\partial\Omega}(v)}{R(v)}$  or  $\frac{gsf(v)}{R(v)}$ .

The following Lemma relates the insertion radius of a vertex with its distance from its parent:

**Lemma 2.8** *Let  $v$  be an R3 or an R4 vertex inserted into the mesh and  $w$  be its parent. Then,  $R(v) = |vw|$ .*

**Proof:** If  $v$  is an R3 or R4 vertex, then  $v$  is the center of an element  $t$ 's circumball or surface ball. According to Definition 2.8, the parent  $w$  of  $v$  is one vertex of  $t$ . Because of the empty circumball and surface ball property, Definition 2.7 implies that  $R(v) = |vw|$ . ■

The following Lemma relates the density of a vertex  $v$  with that of its parent:

**Lemma 2.9** *Let  $v$  be an R3 or R4 vertex and  $R(v) \geq c \cdot R(w)$ , where  $w = Par(v)$ . Then*

$$D(v) \leq 1 + \frac{D(w)}{c}. \quad (2.5)$$

**Proof:** The proof is similar to the proof of Lemma 6 in [118].

Let  $w = Par(v)$ . Since  $glfs_{\partial\Omega}(\cdot)$  and  $gsf(\cdot)$  are 1-Lipschitz (see Remark 2.2), we have that:

$$\begin{aligned} \min\{glfs_{\partial\Omega}(v), gsf(v)\} &\leq \min\{|vw| + glfs_{\partial\Omega}(w), |vw| + gsf(w)\} \\ &= |vw| + \min\{glfs_{\partial\Omega}(w), gsf(w)\} \\ &= R(v) + \min\{glfs_{\partial\Omega}(w), gsf(w)\} && \text{(from Lemma 2.8)} \\ &= R(v) + R(w)D(w) && \text{(from Equation (2.4))} \\ &\leq R(v) + \frac{R(w)}{c} D(w). \end{aligned}$$

and the result follows by dividing both sides by  $R(v)$ . ■

Before we proceed to the proof of good grading, we need two auxiliary Lemmata:

**Lemma 2.10** *Let  $v$  be an R2 vertex. Then,  $|vcfp(v)| \leq R(v)$ .*

**Proof:** Vertex  $v$  is an R2 vertex because of an intersecting tetrahedron  $t$ . Since  $t$  is an intersecting tetrahedron and  $v$  is the center of  $t$ , we have that  $|vcfp(v)| \leq |r(t)|$ . Definition 2.7, however, implies that  $|r(t)| = R(v)$ , and the statement holds.  $\blacksquare$

**Lemma 2.11** *Let  $v$  be a vertex inserted into the mesh. Then,*

- $D(v) \leq \frac{1+\delta\sqrt{3}}{\delta}$ , if  $v$  is an R1, R2, or a box vertex,
- $D(v) \leq \frac{1+\delta}{\delta}$ , if  $v$  is an R3 vertex and  $R(v) \geq \min\{\Delta_{\partial\Omega}(w), \Delta_{\partial\Omega}(v)\}$ , where  $w = Par(v) \in \partial\Omega$ , and
- $D(v) \leq 1$ , if  $v$  is an R5 vertex.

**Proof:** We separate cases.

Let  $v$  be an R1 vertex. According to the flow diagram of Figure 2.4, we have that  $R(v) \geq \Delta_{\partial\Omega}(cfp(v)) = \Delta_{\partial\Omega}(v) = \delta \cdot lfs_{\partial\Omega}(z)$ . From Remark 2.2, we have that  $R(v) \geq \delta \cdot lfs_{\partial\Omega}(z) = \delta \cdot glfs_{\partial\Omega}(z)$ , giving that  $D(v) \leq \frac{1}{\delta} \leq \frac{1+\delta\sqrt{3}}{\delta}$ , and the statement holds.

Let  $v$  be an R2 or a box vertex. According to the flow diagram,  $R(v) \geq \Delta_{\partial\Omega}(cfp(v)) = \delta \cdot lfs_{\partial\Omega}(cfp(v)) = \delta \cdot glfs_{\partial\Omega}(cfp(v))$ , and from the fact that the general local feature size is 1-Lipschitz, we have that  $R(v) \geq \delta (glfs_{\partial\Omega}(v) - |vcfp(v)|)$ . From Lemma 2.4 and Lemma 2.10, we know that  $|vcfp(v)| \leq R(v)\sqrt{3}$ . Therefore, we obtain that  $R(v) \geq \delta (glfs_{\partial\Omega}(v) - R(v)\sqrt{3})$ . Dividing both sides by  $R(v)$  finally gives that  $D(v) \leq \frac{1+\delta\sqrt{3}}{\delta}$ , and the statement holds.

Let  $v$  be an R3 vertex and  $R(v) \geq \Delta_{\partial\Omega}(v) = \delta \cdot lfs_{\partial\Omega}(v)$ . It follows directly that  $D(v) \leq \frac{1}{\delta} < \frac{1+\delta}{\delta}$ .

Let  $v$  be an R3 vertex and  $R(v) \geq \Delta_{\partial\Omega}(w)$ , where  $w \in \partial\Omega$  is the parent of  $v$ . From Remark 2.2, we obtain that  $R(v) \geq \Delta_{\partial\Omega}(w) = \delta \cdot lfs_{\partial\Omega}(w) = \delta \cdot glfs_{\partial\Omega}(w) \geq \delta (glfs_{\partial\Omega}(v) - |vw|)$ . From Definition 2.8,  $w$  is one of the vertices of a restricted facet whose surface ball has  $v$  as the center. From the empty surface ball property and Definition 2.7, we know that  $R(v) = |vw|$ . Therefore,  $R(v) \geq \delta (glfs_{\partial\Omega}(v) - R(v))$ . Dividing both sides by  $R(v)$  finally gives that  $D(v) \leq \frac{1+\delta}{\delta}$ , and the statement holds.

Let  $v$  be an R5 vertex. According to the flow diagram, all the arrows pointing to R5 are dashed and labeled as sf. The label of dashed arrows is the absolute value

of  $R(v)$  and therefore,  $R(v) \geq \text{sf}(v)$ . Since, however,  $\text{gsf}(v) = \inf_{p \in \Omega} \{ |pv| + \text{sf}(p) \} \leq |vv| + \text{sf}(v) = \text{sf}(v)$ , we get that  $R(v) \geq \text{gsf}(v)$ , and the proof is complete.  $\blacksquare$

Finally, the following Theorem proves that our algorithm achieves good grading:

**Theorem 2.3 (Good Grading)** *Let  $\bar{\rho}_f$  be strictly larger than 1 and let  $\bar{\rho}_t$  be strictly larger than  $X = \sqrt{\sqrt{4 - \frac{1}{\bar{\rho}_f^2}} + 2} \left( \geq \sqrt{\sqrt{3} + 2} \approx 1.93 \right)$ . Let  $v$  be an  $R_i$  vertex inserted into the mesh,  $i = 1, 2, 3, 4, 5, \text{proj}$ . Then, right after its insertion, its density  $D(v)$  is bounded from above by a fixed constant  $D_i > 0$ .*

**Proof:** This theorem will be proved via induction.

Initially, only the 8 box corners are inserted into the triangulation. According to Lemma 2.11, the induction basis holds, if

$$D_{\text{proj}} = \frac{1 + \delta\sqrt{3}}{\delta}. \quad (2.6)$$

For the induction hypothesis, assume that the density  $D(w)$  of  $v$ 's parent  $R_j$  vertex  $w$  is bounded from above by  $D_j$ , where  $j = 1, 2, 3, 4, 5, \text{proj}$ . We need to show that one constant  $D_i$  bounds from above the density of  $R_i$  vertex  $v$ , where  $i = 1, 2, 3, 4, 5, \text{proj}$ .

We separate cases according to the type of  $v$ :

- **$v$  is an R1, R2, or a box vertex**

According to Lemma 2.11, the insertion radius of  $v$  is bounded from above by  $\frac{1 + \delta\sqrt{3}}{\delta}$ . Therefore, no matter what the parent of  $v$  is, the induction step holds, if

$$D_1 = D_2 = D_{\text{proj}} = \frac{1 + \delta\sqrt{3}}{\delta}. \quad (2.7)$$

- **$v$  is an R5 vertex**

Similarly to the case above, Lemma 2.11 suggests that no matter what the parent of  $v$  is, the induction step holds, if

$$D_5 = 1. \quad (2.8)$$

- **$v$  is an R4 vertex**

From the flow diagram, all the arrows pointing to R4 are labeled with  $\bar{\rho}_t$ . Therefore, from Lemma 2.9 and Lemma 2.6, we get that  $D(v) \leq 1 + \frac{D(w)}{c}$ , with  $c$  equal

to  $\bar{\rho}_t$  for any parent  $w$ . Thus, the induction step would be proved, if  $D_4$  was set to a value that satisfied all the following inequalities:

$$1 + \frac{D_1}{\bar{\rho}_t} \leq D_4 \quad (2.9)$$

$$1 + \frac{D_3}{\bar{\rho}_t} \leq D_4 \quad (2.10)$$

$$1 + \frac{D_4}{\bar{\rho}_t} \leq D_4 \quad (2.11)$$

$$1 + \frac{D_5}{\bar{\rho}_t} = 1 + \frac{1}{\bar{\rho}_t} \leq D_4 \quad (2.12)$$

Observe that the  $D_5$  term in Inequality (2.12) is replaced by 1, according to Equality (2.8).

- **$v$  is an R3 vertex**

According to Lemma 2.11,  $D(v)$  is bounded from above by  $\frac{1+\delta}{\delta}$  for the relationships of Figure 2.4 that are depicted by the dashed arrows pointing to R3. Therefore, for the induction step to be proved,  $D_3$  has to satisfy at least the following inequality:

$$\frac{1+\delta}{\delta} \leq D_3 \quad (2.13)$$

For the rest of the relationships (i.e., solid arrows), we know from Lemma 2.9 and Lemma 2.6 that  $D(v) \leq 1 + \frac{D(w)}{c}$ , where  $c$  is equal to  $\bar{\rho}_f$  if  $w$  is an R3 vertex or equal to  $\min\left\{\frac{1}{X}, \bar{\rho}_f\right\}$  if  $w$  is an R1, R2, Rproj, R4 or R5 vertex.

Therefore, the induction step would be proved, if  $D_3$  was set to a value that satisfied also the following inequalities:

$$1 + D_1 \max\left\{X, \frac{1}{\bar{\rho}_f}\right\} = 1 + D_1 X \leq D_3 \quad (2.14)$$

$$1 + \frac{D_3}{\bar{\rho}_f} \leq D_3 \quad (2.15)$$

$$1 + D_4 \max \left\{ X, \frac{1}{\bar{\rho}_f} \right\} = 1 + D_4 X \leq D_3 \quad (2.16)$$

$$1 + D_5 \max \left\{ X, \frac{1}{\bar{\rho}_f} \right\} = 1 + X \leq D_3 \quad (2.17)$$

Observe that  $X$  is always larger than  $\frac{1}{\bar{\rho}_f}$  when  $\bar{\rho}_f > 1$  and that is why the  $\frac{1}{\bar{\rho}_f}$  term is eliminated from Inequalities (2.14), (2.16), and (2.17). Also, the  $D_5$  term in Inequality (2.17) is replaced by 1, according to Equality (2.8).

Putting it all together and simplifying the results, Inequalities (2.9)- (2.17) above are simultaneously satisfied by choosing:

$$D_4 = \max \left\{ \frac{\bar{\rho}_t + 1}{\bar{\rho}_t - X}, \frac{\delta (\bar{\rho}_t + 1) + X (1 + \delta \sqrt{3})}{\delta \bar{\rho}_t}, \frac{\bar{\rho}_t (\bar{\rho}_f - 1) + \bar{\rho}_f}{\bar{\rho}_t (\bar{\rho}_f - 1)} \right\} \quad (2.18)$$

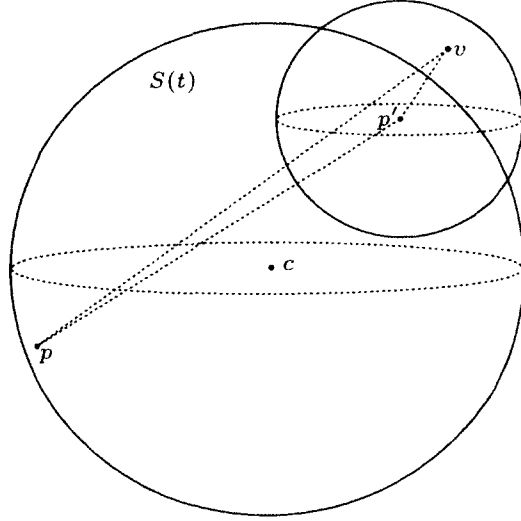
and

$$D_3 = \max \left\{ \frac{\delta + X (1 + \delta \sqrt{3})}{\delta}, \frac{\delta \bar{\rho}_t (1 + X) + X (1 + \delta \sqrt{3})}{\delta \bar{\rho}_t}, \frac{\bar{\rho}_f}{\bar{\rho}_f - 1}, \frac{\bar{\rho}_t (1 + X)}{\bar{\rho}_t - X} \right\} \quad (2.19)$$

Equalities (2.6), (2.7), (2.8), (2.18), and (2.19) satisfy both the induction basis and the induction step for any number and type of vertices, and therefore, the proof is complete. ■

## 2.5 Proof of Fidelity

In this section, we derive an upper bound for  $\delta$ , such that the boundary of the final mesh is a provably good topological and geometric approximation of  $\partial\Omega$ . Our goal



**Figure 2.5:** Illustration to the proof of Lemma 2.12.

is to prove that the mesh boundary  $\partial\mathcal{M}$  (see Definition 2.6) is equal to  $\mathcal{D}_{|\partial\Omega}(E)$  for  $E$  a 0.09-sample of  $\partial\Omega$  (see Theorem 2.4 of this section). To see why this is enough, recall that from Theorem 2.1, the restriction of a 0.09-sample of  $\partial\Omega$  to  $\partial\Omega$  is a good topological and geometric approximation of  $\partial\Omega$ .

First, we show that  $\delta$  directly controls the density of the feature vertices. Let  $V$  be the set of vertices in the triangulation and  $E$  be equal to  $V \cap \partial\Omega$ .

**Lemma 2.12** *Let  $\delta < \frac{1}{4}$ . Then  $E$  is a  $\frac{5\delta}{1-4\delta}$ -sample of  $\partial\Omega$ .*

**Proof:** Recall that upon termination, there is no tetrahedron for which R1, R2, R3, R4, or R5 apply.

See Figure 2.5. Let  $p$  be an arbitrary point on  $\partial\Omega$ . Since  $\mathcal{D}(V)$  covers all the domain, point  $p$  has to lie on or inside the circumsphere of a tetrahedron  $t$  (not shown). Hence,  $t$  is an intersecting tetrahedron. Let point  $p'$  be the feature point closest to  $c(t)$ . Note that  $|c(t)p| \geq |c(t)p'|$  and therefore  $p'$  lies on or inside  $t$ 's circumsphere. We also know that  $t$ 's circumradius has to be less than  $2\Delta_{\partial\Omega}(p')$ , since otherwise R2 would apply for  $t$ . Therefore, we have the following:

$$\begin{aligned}
|pp'| &< 2r(t) && \text{(because both } p \text{ and } p' \text{ lie on or inside } B(t)) \\
&< 4\Delta_{\partial\Omega}(p') && \text{(because of R2)} \\
&\leq 4\delta(|pp'| + \text{fs}_{\partial\Omega}(p)) && \text{(from Inequality (2.1)).}
\end{aligned}$$

and by reordering the terms, we obtain that:

$$|pp'| < \frac{4\delta}{1-4\delta} \text{lfs}_{\partial\Omega}(p), \text{ with } \delta < \frac{1}{4}. \quad (2.20)$$

Moreover, there must exist a feature vertex  $v$  in the triangulation closer than  $\Delta_{\partial\Omega}(p') = \delta \cdot \text{lfs}_{\partial\Omega}(p')$  to  $p'$ , since otherwise R1 would apply for  $t$ . Hence,  $|vp'| < \delta \cdot \text{lfs}_{\partial\Omega}(p')$ , and using Inequality (2.1), we have that:

$$|vp'| < \delta (|pp'| + \text{lfs}_{\partial\Omega}(p)) \quad (2.21)$$

Applying the triangle inequality for  $\Delta pvp'$  yields the following:

$$\begin{aligned} |pv| &\leq |pp'| + |vp'| \\ &< |pp'| + \delta (|pp'| + \text{lfs}_{\partial\Omega}(p)) && \text{(from Inequality (2.21))} \\ &= |pp'| (1 + \delta) + \delta \cdot \text{lfs}_{\partial\Omega}(p) \\ &< \frac{4\delta}{1-4\delta} \text{lfs}_{\partial\Omega}(p) (1 + \delta) + \delta \cdot \text{lfs}_{\partial\Omega}(p) && \text{(from Inequality (2.20))} \\ &= \left( \frac{4\delta(1+\delta)}{1-4\delta} + \delta \right) \text{lfs}_{\partial\Omega}(p) \\ &= \frac{5\delta}{1-4\delta} \text{lfs}_{\partial\Omega}(p), \end{aligned}$$

and the proof is complete. ■

Recall from Section 2.1 that the multi-tissue object  $\Omega$  could be described as a union of materials  $\Omega = \bigcup_i^n \Omega_i$ . Let us denote by  $\Omega_i^j$ , the  $j^{\text{th}}$  connected component of a specific tissue  $\Omega_i$ ,  $j = 1, \dots, m$ .

Similar to Definition 2.4,  $\mathcal{D}_{|\partial\Omega_i^j}(V)$  denotes the set of those facets in the Delaunay triangulation of the vertices in  $V$  whose dual Voronoi edge intersects the surface  $\partial\Omega_i^j$  of  $\Omega_i^j$ . Also, note that  $\partial\Omega = \bigcup_i^n \bigcup_j^m \partial\Omega_i^j$ .

From Lemma 2.12 and Definition 2.3, the following Corollary follows:

**Corollary 2.1** *Let  $\delta \leq \frac{0.09}{5.36} \approx 0.0168$  and let  $E_i^j = V \cap \partial\Omega_i^j$ . Then,  $E_i^j$  is a 0.09-sample of  $\partial\Omega_i^j$ .*

As we have already mentioned in Section 2.2, the final mesh  $\mathcal{M}$  reported consists of tetrahedra whose circumcenter lies inside  $\Omega$ . Let  $\mathcal{M}_i^j$  be the set of tetrahedra whose circumcenter lies inside  $\Omega_i^j$ .



Similar to Definition 2.6,  $\partial\mathcal{M}_i^j$  denotes the set of the boundary facets of submesh  $\mathcal{M}_i^j$ . That is,  $\partial\mathcal{M}_i^j$  contains the facets incident to two tetrahedra such that one tetrahedron has a circumcenter lying inside  $\Omega_i^j$  and the other has a circumcenter lying either outside  $\Omega_i^j$  or on  $\partial\Omega_i^j$ .

**Lemma 2.13** *Let  $t$  be an intersecting tetrahedron whose circumball  $B(t)$  contains a point  $m$  of  $\partial\Omega$ 's medial axis. Then,  $\delta > \frac{1}{4}$ .*

**Proof:** Upon termination, rule R2 cannot apply for any tetrahedron. Therefore, we have the following:

$$\begin{aligned}
2 \cdot \delta \cdot \text{fs}_{\partial\Omega}(\text{cfp}(c(t))) &> |r(t)| && \text{(from R2)} \\
&\geq \frac{|\text{cfp}(c(t))m|}{2} && \text{(since } m \text{ and } \text{cfp}(c(t)) \text{ lie inside } B(t)) \\
&\geq \frac{\text{fs}_{\partial\Omega}(\text{cfp}(c(t)))}{2} && \text{(since } m \text{ is on the medial axis)} \quad \Rightarrow \\
\delta &> \frac{1}{4}.
\end{aligned}$$

■

**Lemma 2.14** *Let  $\delta \leq \frac{1}{4}$ . Any facet  $f \in \partial\mathcal{M}_i^j$  belongs to  $\mathcal{D}_{|\partial\Omega_i^j}(V)$  and has its vertices on  $\partial\Omega_i^j$ .*

**Proof:** Since  $f$  belongs to  $\partial\mathcal{M}_i^j$ ,  $f$  is incident to two tetrahedra  $t_1, t_2 \in \mathcal{D}(V)$ , such that the circumcenter of  $t_1$  lies inside  $\Omega_i^j$  and the circumcenter of  $t_2$  lies outside  $\Omega_i^j$  or  $\partial\Omega_i^j$ . However, this means that the Voronoi edge of  $f$  intersects  $\partial\Omega_i^j$ , and therefore,  $f \in \mathcal{D}_{|\partial\Omega_i^j}(V)$ . This completes the first part.

For the second part and for the sake of contradiction, assume that there is at least one vertex  $v$  of  $f$  that does not lie on  $\partial\Omega_i^j$ , but on another  $\partial\Omega_i^{j'}$ . Consider the tetrahedron  $t_1$ , one of the two tetrahedra incident to  $f$  with circumcenter lying inside  $\Omega_i^j$ . Since  $v$  lies on  $\partial\Omega_i^{j'}$ , the circumball  $B(t_1)$  of  $t_1$  intersects  $\partial\Omega$  in more than one connected component. According to Lemma 7 of Amenta and Bern [9], this implies that  $B(t)$  contains a point  $m$  of the medial axis of  $\partial\Omega$ . Moreover, observe that  $t_1$  is in fact an intersecting tetrahedron. From Lemma 2.13, we finally get that  $\delta > \frac{1}{4}$ . However, this raises a contradiction, since  $\delta$  is assumed to be no larger than  $\frac{1}{4}$ . ■

The next two Lemmas prove a few useful properties for the mesh  $\mathcal{M}$  and its boundary  $\partial\mathcal{M}$ . Our goal is to show that  $\partial\mathcal{M}_i^j$  is always non-empty and does not have

boundary (Lemma 2.16), a fact that will be used for proving the fidelity guarantees (Theorem 2.4).

**Lemma 2.15** *Let  $\delta \leq \frac{1}{4}$ . Then,  $\mathcal{M}_i^j \neq \emptyset$ .*

**Proof:** For the sake of contradiction, assume that  $\mathcal{M}_i^j$  is empty. That means that there is no tetrahedron whose circumcenter lies inside  $\Omega_i^j$ . Since the triangulation  $\mathcal{D}(V)$  covers all the domain, the circumballs of the tetrahedra in  $\mathcal{D}(V)$  also cover the tissue  $\Omega_i^j$ . Therefore, there has to be a circumball  $B(t)$  ( $t \in \mathcal{D}(V)$ ) which contains a point  $m$  on the medial axis of  $\partial\Omega_i^j$ , such that  $m$  lies inside  $\Omega_i^j$ . By our assumption, the circumcenter  $c(t)$  cannot lie inside  $\Omega_i^j$ . Therefore,  $t$  is an intersecting tetrahedron. From Lemma 2.13, we finally get that  $\delta > \frac{1}{4}$ . However, this raises a contradiction, since  $\delta$  is assumed to be no larger than  $\frac{1}{4}$ . ■

**Lemma 2.16** *Let  $\delta \leq \frac{1}{4}$ . Then  $\partial\mathcal{M}_i^j$  is a non-empty set and does not have boundary.*

**Proof:** The fact that  $\partial\mathcal{M}_i^j$  is a non-empty set follows directly from Lemma 2.15: since  $\mathcal{M}_i^j$  cannot be empty, its boundary  $\partial\mathcal{M}_i^j$  cannot be empty too. For the other part, since  $\partial\mathcal{M}_i^j$  is the boundary of a set of tetrahedra, it cannot have boundary. ■

The following Theorem proves the fidelity guarantees achieved by our algorithm:

**Theorem 2.4** *Let  $\delta = 0.0168$ . Then the mesh boundary  $\partial\mathcal{M}$  is a 2-manifold ambient isotopic to  $\partial\Omega$  and the 2-sided Hausdorff distance between the mesh boundary and  $\partial\Omega$  is  $O(\delta^2)$ .*

**Proof:** By Theorem 2.1, it is enough to prove that  $\partial\mathcal{M}$  is the restriction to  $\partial\Omega$  of the Delaunay triangulation of a 0.09-sample. We will, in fact, show that the boundary  $\partial\mathcal{M}_i^j$  of the submesh  $\mathcal{M}_i^j$  is equal to  $\mathcal{D}_{|\partial\Omega_i^j}(E_i^j)$  (recall that  $E_i^j$  is equal to  $V \cap \partial\Omega_i^j$ ) which is the restriction to  $\partial\Omega_i^j$  of the Delaunay triangulation of a 0.09-sample of  $\partial\Omega_i^j$ , by Corollary 2.1. This is enough, since this would prove that the boundary of each submesh  $\mathcal{M}_i^j$  is an accurate representation of the interface  $\partial\Omega_i^j$ , for any  $i$  and  $j$ .

Let  $f$  be a facet in  $\partial\mathcal{M}_i^j$ . From Lemma 2.14, we know that  $f \in \mathcal{D}_{|\partial\Omega_i^j}(V)$  that  $f$ 's vertices lie on  $\partial\Omega_i^j$ . Let  $B$  be the surface ball of  $f$ . From Definition 2.5, the interior

$\text{int}(B)$  of  $B$  is empty of vertices in  $V$ . Therefore,  $\text{int}(B)$  is empty of vertices in  $V \cap \partial\Omega_i^j$  also. Without loss of generality, assume that the vertices in  $V$  are in general position. Since there is a ball  $B$  that circumscribes  $f$  and does not contain vertices of  $V \cap \partial\Omega_i^j$  in its interior,  $f$  has to appear as a simplex in  $\mathcal{D}(V \cap \partial\Omega_i^j)$ . Since the center of  $B$  lies on  $\partial\Omega_i^j$ , then the Voronoi dual of  $f$  intersects  $\partial\Omega_i^j$  in  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$ , as well. Hence,  $\partial\mathcal{M}_i^j \subseteq \mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$ .

For the other direction, we will prove that  $\partial\mathcal{M}_i^j$  cannot be a proper subset of  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$ , and therefore, equality between these two sets is forced. Toward this direction, we will prove that any proper non-empty subset of  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$  has boundary; this is enough, because we have proved in Lemma 2.16 that  $\partial\mathcal{M}_i^j$  is non-empty and does not have boundary.

Since  $V \cap \partial\Omega_i^j$  meets the requirements of Theorem 2.1,  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$  is a 2-manifold without boundary. Therefore, any edge in  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$  is incident to exactly two facets of  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$ . Since any proper non-empty subset  $\mathcal{A}$  of  $\mathcal{D}_{|\partial\Omega_i^j}(V \cap \partial\Omega_i^j)$  has fewer facets,  $\mathcal{A}$  contains at least an edge  $e$  incident to only one facet. However, this implies that  $e$  belongs to the boundary of  $\mathcal{A}$ , and the proof is complete. ■

## 2.6 Implementation details

We used the *Insight Toolkit* (ITK) for image processing [7]. ITK provides, among others, the implicit function  $f$  that describes object  $\Omega$  to be meshed (see Section 2.1). Specifically, given a real point  $p$ ,  $f$  returns 0 if the voxel enclosing  $p$  is in the background, or it returns the identifier  $i$  of the tissue  $\Omega_i$  if that voxel belongs to  $\Omega_i$ ,  $i = 1, \dots, n$ . In order to compute the closest feature point function  $\text{cfp}(p)$  and identify the cloud of points lying on  $\partial\Omega$ , we make use of the Euclidean Distance Transform (EDT) as implemented in ITK and presented in [97]. Specifically, the EDT returns the boundary voxel  $p'$  which is closest to  $p$ . Then, we traverse the ray  $\overrightarrow{pp'}$  and we compute the intersection between the ray and  $\partial\Omega$  by interpolating the positions where  $f$  changes value [96]. The actual mesh generator was built on top of the *Computational*

*Geometry Algorithms Library* (CGAL) [6]. CGAL offers flexible data structures for Delaunay point insertions and removals and robust exact geometric predicates.

The rest of this section describes important implementation aspects.

### 2.6.1 Medial Axis Approximation

Recall that rules R1 and R2 make an extensive use of  $\text{lfs}_{\partial\Omega}(\cdot)$ , and therefore, knowledge about the medial axis is needed.

Since the computation of the exact medial axis is a difficult problem [56, 70], we seek a good (for our purposes) approximation of it. Precisely, we are interested in computing  $\tilde{\text{lfs}}_{\partial\Omega}(p)$ : the approximation of  $\text{lfs}_{\partial\Omega}(p)$ , where  $p \in \partial\Omega$ .

**Remark 2.3** *In this subsection, we do not alter the fidelity guarantees of Theorem 2.4, since the theorem assumes that  $\text{lfs}_{\partial\Omega}(\cdot)$  is known and accurate; in this subsection, we attempt to provide a fast way to approximate  $\text{lfs}_{\partial\Omega}(\cdot)$ .*

For an excellent review of image-based medial axis approximation methods, see the work of Coeurjolly and Montanvert [49]. The authors also describe an optimal algorithm (MAEVA<sup>1</sup>) for the computation of the medial axis, which is a free implementation to download. We found out, however, that although the method is fast, the resulted discrete medial axis was not accurate enough for our purposes. We attribute this behavior to the fact that image-based methods do not realize the underlying shape; they compute the medial axis of volumetric data, which contains discontinuities and thus, renders the computation unstable.

Amenta *et al.* [11] and Dey and Zhao [56] (and the references therein) consider methods that given a set of sample points on the surface, they approximate the medial axis from their Voronoi diagram. Their key concept is the *Pole* of a feature vertex, a technique that we integrate into our algorithm in order to compute  $\tilde{\text{lfs}}_{\partial\Omega}(\cdot)$ . Boissonnat and Oudot [28] describe a two-phase algorithm that is able to approximate the medial axis based on the notion of the *Lambda-Medial Axis* [33]. The Lambda-Medial Axis makes weaker assumptions about the sample and as such, it is suitable

---

<sup>1</sup><http://liris.cnrs.fr/david.coeurjolly/doku/doku.php?id=code:maeva>

for noisy data. Nevertheless, we found that the Pole technique is easier to implement and quite robust for our purposes in all the input images we tried. It should also be mentioned that both the Pole and the Lambda-Medial Axis technique focus on surface recovery and not volume meshing. That means that they assume that only vertices on the isosurface are allowed (i.e., the sample). This is not the case in this work, since the quality criteria might dictate the insertion of vertices in the interior of the domain. As we explain below, this difference necessitates the simultaneous maintenance of a second triangulation.

Let  $E$  be a vertex set on  $\partial\Omega$  and consider the voronoi vertices of the voronoi cell of a feature vertex  $v \in E$ . The voronoi vertices inside  $\Omega$  (if any) are called *internal* and the rest (if any) *external*. Amenta *et al.* [11] shows that if  $E$  is dense, the internal pole (i.e., the furthest from  $v$  internal voronoi vertex) is close to the medial axis contained in  $\Omega$ , and the external pole (i.e., the furthest from  $v$  external voronoi vertex) is close to the medial axis contained in the complement of  $\Omega$ . Therefore, the poles of each sample point form a good discrete approximation of the medial axis.

The problem with the poles (as a good approximation of the medial axis) is that  $E$  has to be a dense sample of the surface; however, our algorithm needs the approximation of the medial axis, so it can create a graded sample  $E$ . Recall that we do not assume that a starting sample set is known a priori. In fact, when the algorithm starts, there is not even a single feature vertex inserted into the triangulation. In order to resolve this cyclic dependency, our algorithm alternates between two modes: a “uniform” and a “graded”.

Specifically, the algorithm maintains a second triangulation  $\mathcal{D}(Z)$  (together with the triangulation  $\mathcal{D}(V)$ , see Section 2.2) which contains only feature vertices. To compute  $\tilde{\text{lf}}_{\partial\Omega}(z)$ ,  $z$  is inserted into the current set of feature vertices  $Z$ , and  $\mathcal{D}(Z)$  is updated. Next, the poles of  $z$  are computed from  $\mathcal{D}(Z)$ , and the distance from  $z$  to its closest pole is returned as the approximation of the distance from  $z$  to the medial axis. Clearly, in the early stages of the refinement,  $Z$  is a very sparse sample set, and, therefore, the poles of  $z \in Z$  are not to be trusted as a good approximation of the medial axis. Note, however, that these poles can only be further from  $z$ , than the poles computed at a much denser sample set. In other words, when  $Z$  is sparse,

$\tilde{\text{lfs}}_{\partial\Omega}(z)$  gives a larger value than it should (i.e.,  $\tilde{\text{lfs}}_{\partial\Omega}(z)$  is larger than  $\text{lfs}_{\partial\Omega}(z)$ ). This has severe consequences, since it is possible for the resulting sample not to be as dense as it should.

For this reason, instead of returning just the value of  $\tilde{\text{lfs}}_{\partial\Omega}(z)$ , we choose to return the following quantity:  $\min\{\lambda, \tilde{\text{lfs}}_{\partial\Omega}(z)\}$ , where  $\lambda$  will be specified shortly. When  $\tilde{\text{lfs}}_{\partial\Omega}(z)$  is too large (i.e., larger than  $\lambda$ ), the value of  $\lambda$  is returned. Parameter  $\lambda$  acts as a safety net and simulates the uniform mode of the algorithm: in the worst case, a uniform sample set will be generated, whose density depends on  $\lambda$ . Note that the uniform mode is triggered mostly in the early stages of the algorithm. Later on, more and more feature vertices are inserted into the triangulation, and the medial axis is sufficiently described by the poles; and this is when the graded mode of the algorithm is activated.

Specifying a value for  $\lambda$  is not intuitive. If  $\lambda$  is small, then the approximation of the medial axis would be more accurate, but the graded mode would be activated fewer times, sacrificing in this way a well-graded surface mesh. On the other hand, if  $\lambda$  is large, then we would expect to see better grading, but it is likely for the approximation of medial axis to be so bad (i.e., it is likely that  $\tilde{\text{lfs}}_{\partial\Omega}(\cdot)$  is too large), such that the graded mode would fail to capture the curvature of  $\partial\Omega$ . Nevertheless, extensive experimental evaluation on both synthetic and real medical images has shown that in most cases, setting  $\lambda$  to a value 12 times the size of the voxel suffices.

Note that if the users are not interested in achieving grading along the surface, the second triangulation  $\mathcal{D}(Z)$  is not needed at all, since they could define  $\tilde{\text{lfs}}_{\partial\Omega}(p)$  to be simply equal to  $\lambda$ .

### 2.6.2 Dihedral angle improvement

Provable theoretical guarantees on the minimum and maximum dihedral angles are outside the scope of this thesis. Nevertheless, for practical purposes, we felt that the issue of sliver removal and dihedral angle improvement should be addressed.

We could apply the *sliver exudation* technique [34] in order to improve the dihedral angles. Edelsbrunner and Guoy [58], however, have shown that in most cases sliver

exudation does not remove all poor tetrahedra: elements with dihedral angles less than  $5^\circ$  survive. The *random perturbation* technique [91] offers very small guarantees and sometimes requires many (random) trials for the elimination of a single sliver as reported in [81].

A straightforward and inexpensive way to eliminate slivers is to try to split them by inserting their circumcenter. Shewchuk [117] shows that this technique works when the slivers are far away from the mesh boundary. However, when slivers are close to the mesh boundary, the newly inserted points alter the boundary triangles. In fact, the boundary triangles might not have their vertices on the surface any more, or might not even belong to the restricted triangulation. In this subsection, we propose *point rejection strategies* that prevent the insertion of points which hurt fidelity.

Our algorithm first tries to convert *illegal* facets to *legal* ones. We define legal facets to be those restricted facets whose vertices lie precisely on  $\partial\Omega$ . Conversely, a restricted facet with at least one vertex not lying on  $\partial\Omega$  is called an illegal facet.

Let  $f$  be an illegal facet and  $e$  its voronoi edge (see Figure 2.6a). Recall that  $e$  has to intersect  $\partial\Omega$  (see Section 2.1) at a point  $p$ . Any vertex  $v$  of  $f$  which do not lie precisely on  $\partial\Omega$  is deleted from the triangulation, while point  $p$  is inserted.

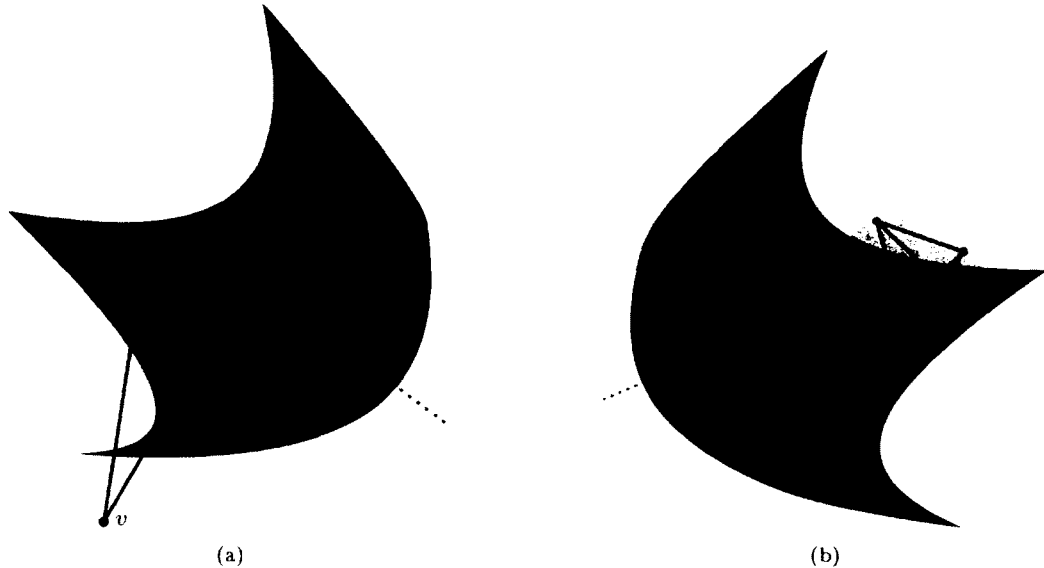
In addition, the algorithm tries to keep in the Delaunay triangulation as many legal facets as possible. Let  $c$  be the circumcenter of a sliver considered for insertion. If the insertion of  $c$  eliminates a legal facet  $f$  (see Figure 2.6b), then  $c$  is not inserted. Instead, a point  $p$  on the intersection of  $\partial\Omega$  and  $f$ 's voronoi edge  $e$  is inserted.

In summary, we cope with slivers by augmenting our algorithm (see Section 2.2) with the following two rules:

**R6:** If an illegal facet  $f$  appears, then all its vertices that do not lie on the surface are deleted and a point  $p$  on  $\text{Vor}(f) \cap \partial\Omega$  is inserted (Figure 2.6a).

**R7:** Let  $t$  be a sliver and  $c$  its circumcenter. If  $c$  eliminates a legal facet  $f$ , then  $c$  is rejected. Instead, a point on  $p$  on  $\text{Vor}(f) \cap \partial\Omega$  is inserted (Figure 2.6b).

We define slivers via the optimization metric  $\eta$ , as described by Liu and Joe [93]. Specifically, for a tetrahedron  $t$ ,  $\eta(t) = \frac{12(3v)^{\frac{2}{3}}}{\sum_{i=1}^6 l_i^2}$ , where  $v$  is the volume of  $t$ , and  $l_i$



**Figure 2.6:** The point rejection strategies. (a)  $f$  is an illegal facet. (b)  $f$  is a legal facet.

are the lengths of  $t$ 's edges. We chose  $\eta$ , because its computation is robust even when  $t$  is an almost flat element. In [93], it is proved that  $0 \leq \eta(t) \leq 1$ . Moreover,  $\eta$  is 0 for a flat element, and 1 for the regular tetrahedron.

We consider a tetrahedron  $t$  to be a sliver, if  $\eta(t)$  is less than 0.06. The reason we chose this value is because: (a) it introduces a small size increase (about 15%) over the mesh obtained without our sliver removal heuristic (i.e., without rules R6 and R7), and (b) it introduces a negligible time overhead. In the Experimental Section 2.7, we show that this 0.06 bound corresponds to meshes consisting of tetrahedra with dihedral angles between  $4.6^\circ$  and  $171^\circ$ .

Note that R6 and R7 never remove feature vertices; on the contrary, they might insert more to “protect” the surface. Hence, they do not violate Theorem 2.4: the mesh boundary continues being equal to the restricted Delaunay triangulation  $\mathcal{D}_{|\partial\Omega}(V \cap \partial\Omega)$ , and therefore a good approximation of the surface. In order not to compromise termination (and the guarantees we give for the radius-edge ratio and the boundary planar angles), if R6 or R7 introduce an edge shorter than the shortest edge already present in the mesh, then the operation is rejected and the sliver in question is ignored.



**Table 2.1:** Performance achieved by our algorithm and CGAL.

(a) Our algorithm.  $\lambda$  is set to 12 times the spacing length of the corresponding image,  $\delta$  to 2.  $\bar{\rho}_t$  to  $\sqrt{\sqrt{3} + 2}$ , and  $\bar{\rho}_f$  to 1.

	Input							
	Embedded Spheres	Torus	Brain	Stomach	Skeleton	Colon	Knee atlas	Head-neck atlas
achieved radius-edge ratio	1.93	1.93	1.93	1.84	1.93	1.93	1.93	1.93
achieved planar boundary angles (degrees)	30.0	30.0	30.0	30.0	30.0	30.0	30.0	30.0
# Vertices	2,428	2,299	6,023	4,712	50,759	5,917	102,330	36,174
# Boundary facets	3,626	3,898	10,080	8,254	95,778	11,040	146,460	74,768
# Elements	8,774	7,727	21,136	15,796	163,120	18,545	426,592	112,778
shortest mesh edge (mm)	0.45	0.61	2.83	4.78	3.9	3.19	2.2	8.2
achieved dihedral angles (degrees)	13.13 – 155.34	12.2 – 155.6	12.0 – 155.7	12.3 – 155.2	10.8 – 156.0	11.3 – 155.8	4.6 – 170.1	4.7 – 170.0
Time (secs)	1.4	1.3	4.8	2.6	41.8	5.3	43.9	20.3

(b) Performance achieved by CGAL on the same set of images.  $\bar{\rho}_t$  is set to  $\sqrt{\sqrt{3} + 2}$ , and  $\bar{\rho}_f$  to 1.

	Input							
	Embedded Spheres	Torus	Brain	Stomach	Skeleton	Colon	Knee atlas	Head-neck atlas
achieved radius-edge ratio	1.41	1.30	1.86	1.34	2.63	1.40	2.34	2.71
achieved planar boundary angles (degrees)	30.0	30.0	16.5	30.0	20.3	22.6	30.0	10.6
# Vertices	7,099	1,684	3,973	3,447	42,603	4,834	81,753	35,755
# Boundary facets	3,682	1,450	2,554	2,866	54,340	5,980	53,186	60,674
# Elements	37,718	7,937	20,271	16,442	173,858	19,524	430,827	127,684
shortest mesh edge (mm)	0.56	1.42	2.63	3.67	0.01	3.19	0.26	0.15
achieved dihedral angles (degrees)	11.87 – 161.30	14.3 – 159.3	11.3 – 161.5	11.0 – 163.8	10.0 – 165.7	12.0 – 160.5	2.1 – 176.1	6.5 – 169.8
Time (secs)	1.0	0.2	0.6	0.5	10.0	1.0	13.7	8.8

We experimentally found that the point rejection strategies were able to generate tetrahedra with angles more than  $5^\circ$  and less than  $170^\circ$ . We emphasize that neither fidelity (see Theorem 2.4) nor termination (see Theorem 2.2) is compromised with this heuristic.

## 2.7 Experimental Evaluation

This section presents the final meshes generated by our algorithm on synthetic and real medical data. All the experiments were conducted on a 64 bit machine equipped with a 2.8 GHz Intel Core i7 CPU and 8 GB of main memory. For the 3D visualization of the final meshes, we used ParaView [122], an open source visualization application.

Although the fidelity guarantees we give hold for a very small value of  $\delta$  (see Theorem 2.4), we wanted to see if our algorithm works well for much larger values of  $\delta$ . Specifically, for all the experiments, we set  $\delta$  to 2, i.e., we set  $\delta$  to a value about 200

**Table 2.2:** Information about the input images.

Image	Resolution	Spacing (mm <sup>3</sup> )	Tissues
Single Sphere	416 × 416 × 416	0.04 × 0.04 × 0.04	1
Embedded Spheres	634 × 416 × 416	0.04 × 0.04 × 0.04	3
Torus	147 × 147 × 67	0.25 × 0.25 × 0.25	1
Brain	316 × 316 × 188	0.93 × 0.93 × 1.5	1
Stomach	140 × 186 × 86	0.96 × 0.96 × 2.4	1
Skeleton	359 × 265 × 218	0.96 × 0.96 × 2.4	1
Colon	296 × 167 × 117	0.96 × 0.96 × 1.8	1
Knee atlas	413 × 400 × 116	0.27 × 0.27 × 1	49
Head-neck atlas	241 × 216 × 228	0.97 × 0.97 × 1.4	60

times larger than Theorem 2.4 recommends. A larger value of  $\delta$  also implies that the size of the output mesh is smaller. Small-size meshes are desirable for two reasons: first, because the mesh generation execution time is considerably less (as it can be seen below, see Table 2.1a), and second, because finite element simulations [18, 19] on them run faster. We observed that even though the fidelity guarantees proved in Section 2.5 do not hold for large  $\delta$ , the results in fact are pretty good. (We should also note that in some applications fidelity is not that important. For instance, a study on the impact of  $\delta$  for the non-rigid registration problem [66] shows that the accuracy and speed of the solver is not very sensitive to fidelity.)

As mentioned in Section 2.6.1, in all the following experiments,  $\lambda$  is set to 12 times the voxel size (i.e., length of the image spacing). Recall that  $\lambda$  is used so that we can compute an approximation of  $\text{lfs}_{\partial\Omega}(p)$ , for  $p \in \partial\Omega$ .

For all the experiments, we set  $\bar{\rho}_t$  to  $\sqrt{\sqrt{3} + 2}$  and  $\bar{\rho}_f$  to 1, and therefore (from Theorem 2.2) termination is certain, all the output tetrahedra are guaranteed to have radius-edge ratio less than  $\sqrt{\sqrt{3} + 2}$ , and all the boundary facets are guaranteed to have planar angles larger than  $30^\circ$ . Recall that quality is not affected by any value of  $\delta$ . Although these parameters imply infinite grading constants (Theorem 2.3), grading is much better in practice, an observation that is also reported in [118] and demonstrated in this Section as well.

The first set of experiments demonstrates the use of custom size functions. Note that the use of any size function alters neither the quality nor the fidelity guarantees, since it is incorporated in Theorem 2.2 (see Section 2.3) and Theorem 2.4 (see

Section 2.5).

We synthetically created the image of a sphere of radius 10mm and center  $(0, 0, 0)$ . See Table 2.2 for information about this image. We ran our algorithm on the sphere image three times, each of which with a different size function:  $\text{sf}_1(\cdot)$ ,  $\text{sf}_2(\cdot)$ , and  $\text{sf}_3(\cdot)$ .  $\text{sf}_1(\cdot)$  restricts the radii of the elements to be smaller than 5mm, while  $\text{sf}_2(\cdot)$  restricts the radii of the elements to be smaller than 1mm.  $\text{sf}_3(\cdot)$  is a non-uniform size function. Specifically, it behaves as  $\text{sf}_1(\cdot)$  for  $z \geq 0$  and as  $\text{sf}_2(\cdot)$  for the other part of the sphere.

Figure 2.7 depicts the results. In all these three experiments, the achieved radius-edge ratio is less than  $\sqrt{\sqrt{3} + 2}$ , and the planar angles are larger than  $30^\circ$ , as theory dictates. Moreover, the dihedral angles of the output tetrahedra are between  $12.9^\circ$  and  $155.8^\circ$ .

Observe that although parameters  $\delta$  and  $\lambda$  (the ones directly responsible for the sampling density) are fixed for all three runs, the sample density varies. In fact, small size functions (i.e., size functions that take low values) make the boundary vertices denser (compare Figure 2.7a and Figure 2.7b for example). Figure 2.7c shows better exactly that: the surface is sampled more where the size function takes low values, and less otherwise. This indirect effect is expected and it is due to R3. Because of a small size function, more free vertices are inserted close to the surface. This, in turn, is likely to invalidate more restricted facets; that is, more restricted facets will not have their vertices on the surface, and thus, R3 is triggered dictating the insertion of more feature vertices to protect the restricted facets.

The next set of experiments shows the output of our method on difficult geometries both manifold and non manifold. Although the fidelity guarantees about the topology of the output mesh are proved only for manifold domains, in this Section we show that our method behaves fairly well for non-manifold cases (see Figure 2.11 for example) as well.

The first couple of images are the embedded spheres and a torus we synthetically created. The third is an MRI brain image obtained from Huashan Hospital<sup>2</sup>. The

---

<sup>2</sup>Huashan Hospital, 12 Wulumuqi Zhong Lu, Shanghai, China.

next three images are CT segmented scans of a skeleton, a colon, and a stomach, obtained from IRCAD Laparoscopic Center<sup>3</sup>. The last two images are the MRI knee atlas [112] and the CT head-neck [80] atlas obtained from the Surgical Planning Laboratory of Brigham and Women’s Hospital<sup>4</sup>. Information about the input images is shown in Table 2.2. Figure 2.8, Figure 2.9, Figure 2.10, and Figure 2.11 show the meshes produced by our algorithm on these input images.

Table 2.1a reports some statistics for the meshes generated by our algorithm. The observed largest radius-edge ratio in all the meshes is no more than  $\sqrt{\sqrt{3} + 2}$  and the observed planar angles of the boundary facets in all meshes is no smaller than  $30^\circ$  corroborating in this way the theory.

Also, for the meshes of Figure 2.8, Figure 2.9, Figure 2.10, and Figure 2.11, notice that: (a) the interior of the object (i.e. the part away from the surface) is meshed with fewer and bigger elements (*volume grading*), and (b) in most cases, more and smaller boundary triangles mesh parts of the surface close to the medial axis (*surface grading*). Graded meshes greatly reduce the total number of elements, representing, at the same time, difficult geometries (i.e., geometries with high curvature and/or non-manifold parts) accurately.

For comparison, Table 2.1b shows the meshes generated by CGAL [6], the state of the art mesh generation tool we are aware of, able to operate directly on images as well. We set the quality parameters to the same values with the ones used in our algorithm. Note, however, that CGAL does not offer surface grading according to the local feature size. Nevertheless, we were able to set an upper limit on the radii of all the tetrahedra, so that the resulting meshes have similar number of elements to the meshes produced by our algorithm.

Indeed, observe that both Table 2.1a and Table 2.1b report similar mesh sizes on the same input image, with one exception: the mesh size on the Embedded Spheres generated by CGAL is more than 4 times larger than the one generated by our method. The reason for this mismatch is the fact that CGAL found it difficult to recover the red ball (see Figure 2.8a) with a small number of elements. We had to considerably

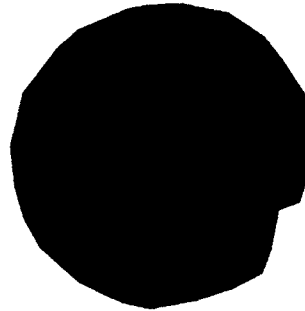
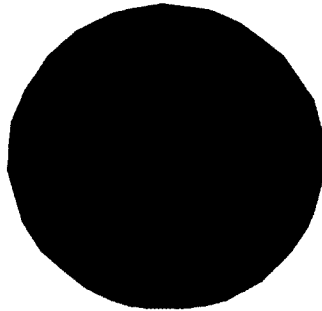
---

<sup>3</sup><http://www.ircad.fr>

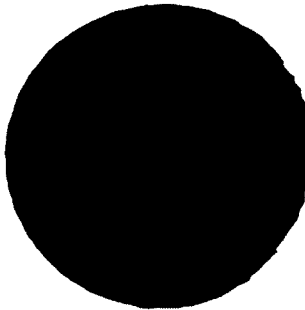
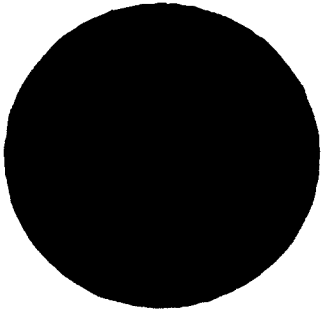
<sup>4</sup><http://www.spl.harvard.edu>

increase the size of the whole mesh so that CGAL could represent both connected components.

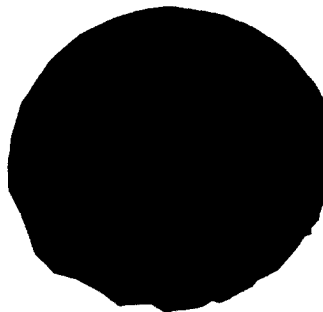
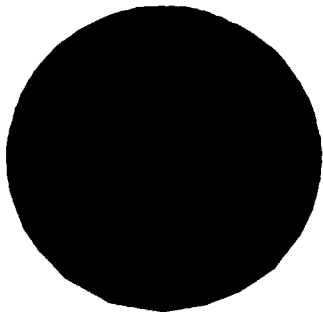
Table 2.1a and Table 2.1b suggest that the quality achieved by our method is comparable to CGAL's. The execution time of our method is much higher, but this is expected since the surface grading offered by our algorithm necessitates the computation of the poles and the maintenance of a second mesh, slowing down the overall meshing time. Improving the speed of our algorithm is the main focus of the next chapter.



(a)  $sf_1(\cdot)$ : the radii are smaller than 5mm.

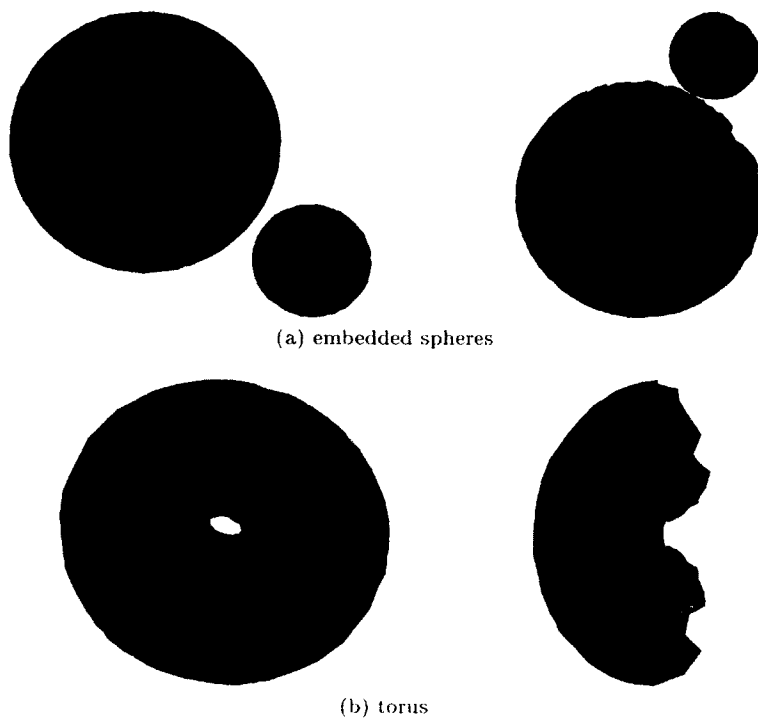


(b)  $sf_2(\cdot)$ : the radii are smaller than 1mm.

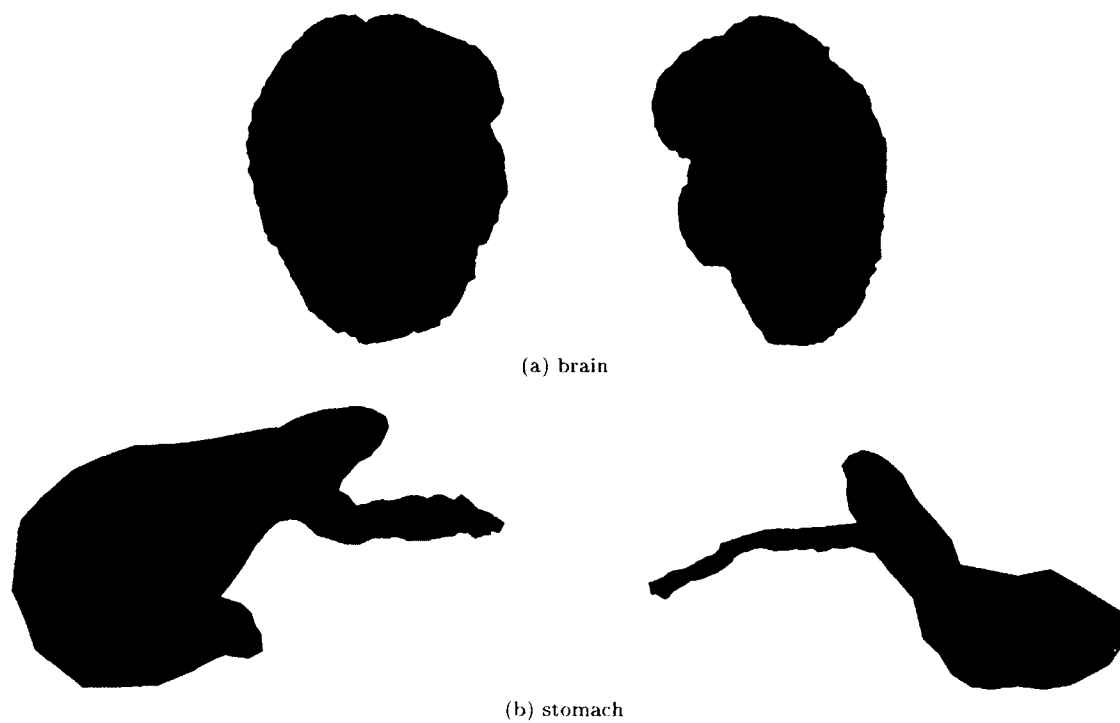


(c)  $sf_3(\cdot)$ : the radii are smaller than 5mm for  $z \geq 0$  and smaller than 1mm for  $z < 0$ .

**Figure 2.7:** Demonstrating the use of size functions. The whole mesh and a cross section of it is displayed.  $\lambda$  is set to about 12 times the spacing length of the image (i.e.,  $\lambda = 12\sqrt{3} \cdot 0.04^2 \approx 0.83$ ),  $\delta$  to 2,  $\bar{\rho}_t$  to  $\sqrt{\sqrt{3} + 2}$ , and  $\bar{\rho}_f$  to 1.



**Figure 2.8:** The final meshes produced by our algorithm for the embedded spheres and the torus. The first mesh of each row illustrates the whole mesh and the second a cross-section of it.



**Figure 2.9:** The final meshes produced by our algorithm for the brain and the stomach.



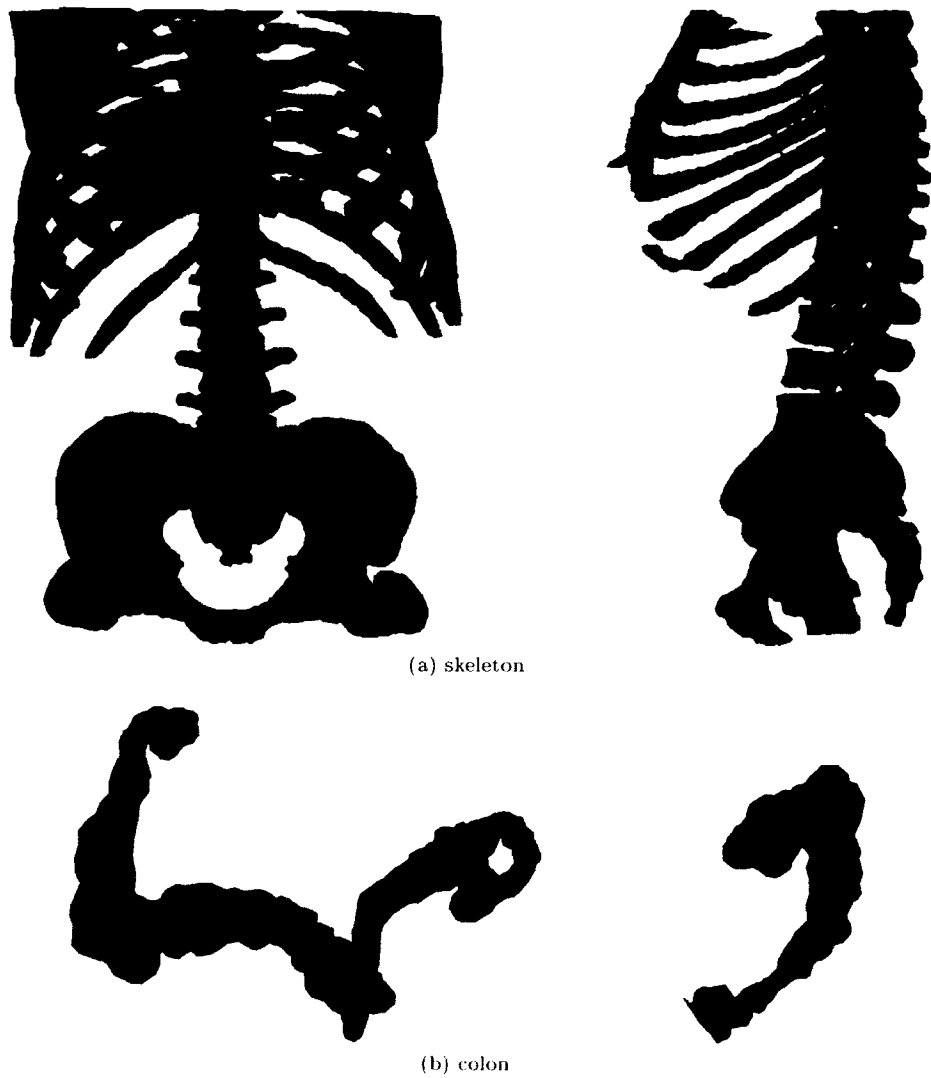
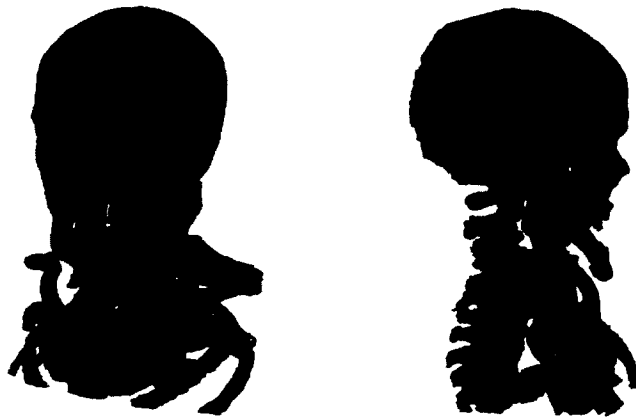


Figure 2.10: The final meshes produced by our algorithm for the skeleton and the colon.



(a) knee atlas



(b) head-neck atlas

**Figure 2.11:** The final meshes produced by our algorithm for the multi-tissue knee and head-neck atlases.

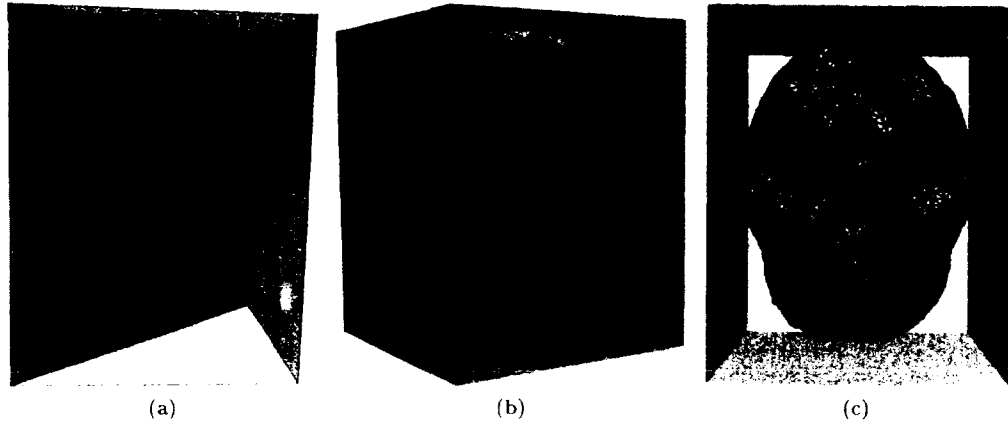
## Chapter 3

# High Quality Real-Time

# Image-to-Mesh Conversion for Finite

# Element Simulations

In this chapter, we present a parallel Image-to-Mesh Conversion (I2M) algorithm with quality and fidelity guarantees achieved by dynamic point insertions and removals. Starting directly from an image, its implementation is capable of recovering the isosurface and meshing the volume with tetrahedra of good shape. Our tightly-coupled shared-memory parallel speculative execution paradigm employs carefully designed contention managers, load balancing, synchronization and optimizations schemes. These techniques are shown to boost not only the parallel but also the single-threaded efficiency of our code. Specifically, our single-threaded performance is faster than both CGAL and TetGen, the state of the art sequential open source meshing tools we are aware of. The effectiveness of our method is demonstrated on Blacklight, the Pittsburgh Supercomputing Center's cache-coherent NUMA machine. We observe a more than 82% strong scaling efficiency for up to 64 cores, and a more than 82% weak scaling efficiency for up to 144 cores, reaching a rate of more than



**Figure 3.1:** (a) The virtual box is meshed into 6 tetrahedra. It encloses the volumetric object. (b) During refinement, the final mesh is gradually being carved according to the Rules. (c) At the end, the set of the tetrahedra whose circumcenter lies inside  $\Omega$  is the geometrically and topologically correct mesh  $\mathcal{M}$ .

14.3 million elements per second. This is the fastest 3D Delaunay mesh generation and refinement algorithm, to the best of our knowledge.

### 3.1 Background: Delaunay Refinement for Smooth Surfaces

Sequential Delaunay Refinement for smooth surfaces is presented in detail in the literature [109, 110] and in our previous work [62, 64]. In this Section, we briefly outline the main concepts.

As is usually the case in the literature [9, 89, 110], we assume that the surface of the object  $\partial\Omega$  to be meshed is a closed smooth 2-manifold. To prove that the boundary  $\partial\mathcal{M}$  of the final mesh  $\mathcal{M}$  is geometrically and topologically equivalent with  $\partial\Omega$ , we make use of the *sample theory* [9]. Omitting the details, it can be proved [9, 11] that the Delaunay triangulation of a dense pointset lying precisely on the isosurface  $\partial\Omega$  contains (as a subset) the correct mesh  $\mathcal{M}$ . That mesh consists of the tetrahedra  $t$  whose circumcenter  $c(t)$  lies inside  $\Omega$ . Formally, the sample theorem could be stated as follows [11, 28, 56]:

**Theorem 3.1** *Let  $V$  be samples of  $\partial\Omega$ . If for any point  $p \in \partial\Omega$ , there is a sample  $v \in V$  such that  $|v - p| \leq \delta$ , then the boundary triangles of  $\mathcal{D}_\Omega(V)$  is a topologically*

correct representation of  $\partial\Omega$ . Also, the 2-sided Hausdorff distance between the mesh and  $\partial\Omega$  is  $O(\delta^2)$ .

Typical values for  $\delta$  are usually fractions of the *local feature size* of  $\partial\Omega$ . See [11, 28, 56, 109] for well defined  $\delta$  parameters. In our application,  $\delta$  values equal to multiples of the voxel size is sufficient.

Therefore, one of the goals of the refinement is to sample the isosurface densely enough. To achieve that, our algorithm first constructs a *virtual box* which encloses  $\Omega$ . The box is then triangulated into 6 tetrahedra, as shown in Figure 3.1. This is the only sequential part of our method. Next, it dynamically computes new points to be inserted into or removed from the mesh maintaining the Delaunay property. This process continues, until certain fidelity and quality criteria are met. Specifically, the vertices removed or inserted are divided into 3 groups: *isosurface* vertices, circumcenters, and *surface-centers*.

The isosurface vertices will eventually form the sampling of the surface so that Theorem 3.1 holds together with its theoretical guarantees about the fidelity of the mesh boundary. Let  $c(t)$  be the circumcenter of a tetrahedron  $t$ . In order to guarantee termination, our algorithm inserts the isosurface vertex which is the closest to  $c(t)$ . In the sequel, we shall refer to the *Closest IsoSurface* vertex of a point  $p$  as  $\text{cfp}(p) \in \partial\Omega$ . The isosurface vertices (like the circumcenters) are computed during the refinement dynamically with the help of a parallel Euclidean Distance Transformation (EDT) presented and implemented in [123]. Specifically, the EDT returns the *surface voxel*  $q$  which is closest to  $p$ . A surface-voxel is a voxel that lies inside the foreground and has at least one neighbor of different label. Then, we traverse the ray  $\vec{pq}$  on small intervals and we compute  $\text{cfp}(p) \in \partial\Omega$  by interpolating the positions of different labels [96]. The density of the inserted isosurface vertices is defined by the user by a parameter  $\delta > 0$ . A low value for  $\delta$  implies a denser sampling of the surface, and therefore, according to Theorem 3.1, a better approximation of  $\partial\Omega$ .

The circumcenter  $c(t)$  of a tetrahedron  $t$  is inserted when  $t$  has low quality (in terms of its radius-edge ratio [117]) or because its circumradius  $r(t)$  is larger than a user-defined size function  $\text{sf}(\cdot)$ . Circumcenters might also be chosen to be removed, when they lie close to an isosurface vertex, because in this case termination is compromised.

Consider a facet  $f$  of a tetrahedron. The *Voronoi* edge  $V(f)$  of  $f$  is the segment connecting the circumcenters of the two tetrahedra that contain  $f$ . The intersection  $V(f) \cap \partial\Omega$  is called a *surface-center* and is denoted by  $c_{\text{surf}}(f)$ . During refinement, surface-centers are computed similarly to the isosurfaces (i.e., by traversing  $V(f)$  on small intervals and interpolating positions of different labels) and inserted into the mesh to improve the planar angles of the boundary mesh triangles [118] and to ensure that the vertices of the boundary mesh triangles lie precisely on the isosurface [109].

In summary, tetrahedra and faces are refined according to the following *Refinement Rules*:

- **R1:** Let  $t$  be a tetrahedron whose circumball intersects  $\partial\Omega$ . Compute the closest isosurface point  $z = \text{cfp}(c(t))$ . If  $z$  is at a distance not closer than  $\delta$  to any other isosurface vertex, then  $z$  is inserted.
- **R2:** Let  $t$  be a tetrahedron whose circumball intersects  $\partial\Omega$ . If its radius  $r(t)$  is larger than  $2 \cdot \delta$ , then  $c(t)$  is inserted.
- **R3:** Let  $f$  be a facet whose Voronoi edge  $V(f)$  intersects  $\partial\Omega$  at  $c_{\text{surf}}(f)$ . If either its smallest planar angle is less than  $30^\circ$  or a vertex of  $f$  is not an isosurface vertex, then  $c_{\text{surf}}(f)$  is inserted.
- **R4:** Let  $t$  be a tetrahedron whose circumcenter lies inside  $\Omega$ . If its radius-edge ratio is larger than 2, then  $c(t)$  is inserted.
- **R5:** Let  $t$  be a tetrahedron whose circumcenter lies inside  $\Omega$ . If its radius  $r(t)$  is larger than  $\text{sf}(c(t))$ , then  $c(t)$  is inserted.
- **R6:** Let  $t$  be incident to an isosurface vertex  $z$ . All the already inserted circumcenters closer than  $2\delta$  to  $z$  are deleted.

Rules R1 and R2 are responsible for creating the appropriate dense sample so that the boundary triangles of the resulting mesh satisfies Theorem 3.1 and thus the fidelity guarantees. R3 and R4 deal with the quality guarantees, while R5 imposes the size constraints of the users. R6 is needed so termination can be guaranteed. See [62, 64, 109] for more details. When none of the above rules applies, then refinement is

complete. In our previous work [62, 64], we prove that termination is guaranteed, the radius-edge ratio of all elements in the mesh is less than 2, and the planar angles of the boundary mesh triangles is less than  $30^\circ$ .

### 3.2 Parallel Delaunay Refinement for Smooth Surfaces

---

**Algorithm 2:** The parallel mesh generation algorithm. It is executed by each thread.

---

```

1 Algorithm: GenerateMesh( $\mathcal{I}$ ,  $\delta$ ,  $\bar{\rho}$ ,  $sf(\cdot)$ ,  $tid$ )
Input :  $\mathcal{I}$  is the image containing  $\Omega$ ,
         $\delta$  is the parameter that determines the density of the surface sampling,
         $\bar{\rho} (> 2)$  is the target radius-edge ratio,
         $sf(\cdot)$  is the size function,
         $tid$  is the unique identifier of the thread.
Output: A Delaunay mesh  $\mathcal{M}$  that is guaranteed to (a) approximate  $\partial\Omega$  in a correct topological way with Hausdorff distance within  $O(\delta^2)$ , (b) be composed of elements with radius-edge ratio less than  $\bar{\rho}$  and (c) have boundary facets with planar angles larger than  $30^\circ$ .

2 if  $tid == 0$  then /* If it is the main thread */
3     /* At this moment, both the mesh and all PELs are empty. */
4     Insert the 8 vertices of a box which contains  $\Omega$ ;
5      $PEL_0 = PEL_0 \cup \text{NewElements}$ ;
6 end
7 while  $PEL_{tid} \neq \emptyset$  do
8      $t = PEL_{tid} \rightarrow \text{next}()$ ;
9     if locking  $t$ 's vertices is not successful then
10        Unlock related vertices; Invoke Contention Manager; continue;
11    end
12    if  $t$  is an intersecting tetrahedron then /* potential R1 element */
13        Compute  $z = cfp(c(t))$ ;
14        if there is an iso-surface vertex closer than  $\delta$  to  $z$  then
15            if  $r(t) \geq 2\delta$  then
16                Compute  $z = c(t)$ ; /* R2 element */
17            end
18        else
19            if  $t$  is adjacent to a restricted facet  $f$ , such that  $\rho(f) \geq 1$  or  $f$ 's vertices do not lie on  $\partial\Omega$  then /* R3 applies. */
20                Compute  $z = c_{\text{surf}}(f)$ ;
21            else
22                if  $c(t)$  lies inside  $\Omega$  and either  $\rho(t) \geq \bar{\rho}$  or  $r(t) \geq sf(c(t))$  then /* R4 or R5 apply. */
23                    Compute  $z = c(t)$ ;
24                else
25                     $PEL_{tid} = PEL_{tid} - t$ ; /*  $t$  is not a poor element */
26                    Unlock all the related vertices; continue;
27                end
28            end
29        end
30        if  $z$  is a isosurface vertex then
31            Prepare to delete all the free vertices that are closer than  $2\delta$  to  $z$ ;
32        end
33        if locking the vertices for the operation is not successful then
34            Rollback; Unlock related vertices; Invoke Contention Manager; continue;
35        end
36        Insert  $z$  and delete the vertices (if any); Unlock all the related vertices;
37        if  $BeggingList \neq \emptyset$  then
38             $other\_tid = BeggingList \rightarrow \text{first}()$ ;
39             $PEL_{other\_id} = PEL_{other\_id} \cup \text{NewElements}$ ; /* Give work to begging Thread other_id */
40            Wake Thread  $other\_id$ ; /* Notify Thread other_id that it can check its PEL again */
41             $BeggingList = BeggingList - \{other\_id\}$ ;
42        end
43    end
44    if  $BeggingList \rightarrow \text{size}() != \#Threads - 1$  then /* If I am NOT the last Thread to ask for work */
45         $BeggingList \rightarrow \text{push\_at\_end}(tid)$ ;
46        Wait;
47        continue; /* Now some other thread gave Thread  $tid$  work, so  $PEL_{tid}$  is not empty any more */
48    else /* The mesh is ready, all PELs are empty */
49        Let the final mesh  $\mathcal{M}$  be equal to the set of the tetrahedra whose circumcenter lies inside  $\Omega$ ;
50    end

```

---

As explained in Section 3.1, before the mesh generation starts, the Euclidean Distance Transform (EDT) of the image is needed for the on-the-fly computation of the appropriate iso-surface vertices. For this pre-processing step, we make use of the publicly available parallel Maurer filter presented and implemented by Staubs *et*

al. [123]. It can be shown [97, 123] that this parallel EDT scales linearly with the respect to the number of threads.

The rest of this section describes the main aspects of our parallel code. Algorithm 2 illustrates the basic building blocks of our multi-threaded mesh-generation design. Note that our tightly-coupled parallelization does not alter the fidelity (Theorem 3.1) and the quality guarantees described in the previous section.

### 3.2.1 Poor Element List (PEL)

Each thread  $T_i$  maintains its own *Poor Element List (PEL)*  $PEL_i$ .  $PEL_i$  contains the tetrahedra that violate the Refinement Rules and need to be refined by thread  $T_i$  accordingly.

### 3.2.2 Operation

An operation that refines an element can be either an insertion of a point  $p$  or the removal of a vertex  $p$ . In the case of insertion, the cavity  $\mathcal{C}(p)$  needs to be found and re-triangulated according to the well known Bowyer-Watson kernel [30, 128]. Specifically,  $\mathcal{C}(p)$  consists of the elements whose circumsphere contains  $p$ . These elements are deleted (because they violate the Delaunay property) and  $p$  is connected to the vertices of the boundary of  $\mathcal{C}(p)$ . In the case of a removal, the ball  $Bp$  needs to be re-triangulated. As explained in [55], this is a more challenging operation than insertion, because the re-triangulation of the ball in degenerate cases is not unique which implies the creation of illegal elements, i.e., elements that cannot be connected with the corresponding elements outside the ball. We overcome this difficulty by computing a *local Delaunay triangulation*  $\mathcal{D}_{\mathcal{B}(p)}$  (or  $\mathcal{D}_{\mathcal{B}}$  for brevity) of the vertices incident to  $p$ , such that the vertices inserted earlier in the shared triangulation are inserted into  $\mathcal{D}_{\mathcal{B}}$  first. In order to avoid races associated with writing, reading, and deleting vertices/cells from a PEL or the shared mesh, any vertex touched during the operation of cavity expansion, or ball filling needs to be locked. We utilize GCC's atomic built-in functions for this goal, since they perform faster than the conventional pthread `try_locks`. Indeed, replacing pthread locks (our first implementation) with



GCC’s atomic built-ins (current implementation) decreased the execution time by 3.6% on 1 core and by 4.2% on 12 cores.

In the case a vertex is already locked by another thread, then we have a *rollback*: the operation is stopped and the changes are discarded [105]. When a rollback occurs, the thread moves on to the next bad element in its PEL.

### 3.2.3 Update new and deleted cells

After a thread  $T_i$  completes an operation, new cells are created and some cells are invalidated. The new cells are those that re-triangulate the cavity (in case of an insertion) or the ball (in case of a removal) of a point  $p$  and the invalidated cells are those that used to form the cavity or the ball of  $p$  right before the operation.  $T_i$  determines whether a newly created element violates a rule. If it does, then  $T_i$  pushes it back to  $PEL_i$  (or to another thread’s PEL, see below) for future refinement. Also,  $T_i$  removes the invalidated elements from the PEL they have been residing in so far, which might be the PEL of another thread. To decrease the synchronization involved for the concurrent access to the PELs, if the invalidated cell  $c$  resides in another thread  $T_j$ ’s  $PEL_j$ , then  $T_i$  removes  $c$  from  $PEL_j$  only if  $T_j$  belongs to the same socket with  $T_i$ . Otherwise,  $T_i$  raises cell  $c$ ’s invalidation flag, so that  $T_j$  can remove it when  $T_j$  examines  $c$ .

As Line 49 of Algorithm 2 shows, the final mesh  $\mathcal{M}$  reported consists of the subset of tetrahedra whose circumcenter lies inside the object  $\Omega$ . To expedite the process of finding those elements, each thread maintains a linked list of those elements on the fly, i.e., from the beginning of mesh generation and refinement. Thus, collecting those elements at the end costs constant time  $O(\#Threads)$ . These linked lists are updated similarly to the update of the Poor Element Lists (PELs) described in the previous paragraph.

### 3.2.4 Load Balancer

Right after the triangulation of the virtual box and the sequential creation of the first 6 tetrahedra, only the main thread might have a non-empty PEL. Clearly, Load

Balancing is a fundamental aspect of our implementation. Our base (not optimized) Load Balancer is the classic *Random Work Stealing* (RHW) [26] technique, since it best fits our implementation design. In Section 3.4.1, we implement an optimized work stealing balancer that takes advantage of the NUMA architecture and achieves an excellent performance.

If the poor element list  $PEL_i$  of a thread  $T_i$  is empty of elements,  $T_i$  “pushes back” its ID to the *Begging List*, a global array that tracks down threads without work. Then,  $T_i$  is busy-waiting and can be awoken by a thread  $T_j$  right after  $T_j$  gives some work to  $T_i$ . A running thread  $T_j$ , every time it completes an *operation* (i.e., a Delaunay insertion or a Delaunay removal), it gathers the newly created elements and places the ones that are poor to the PEL of the first thread  $T_i$  found in the begging list. The classification of whether or not a newly created cell is poor or not is done by  $T_j$ .  $T_j$  also removes  $T_i$  from the Begging List.

To decrease unnecessary communication, a thread is not allowed to give work to threads, if it does not have enough poor elements in its PEL. Hence, each thread  $T_i$  maintains a counter that keeps track of all the poor and *valid* cells that reside in  $PEL_i$ .  $T_i$  is forbidden to give work to a thread, if the counter is less than a threshold. We set that threshold equal to 5, since it yielded the best results. When  $T_i$  invalidates an element  $c$  or when it makes a poor element  $c$  not to be poor anymore, it decreases accordingly the counter of the thread that contains  $c$  in its PEL. Similarly, when  $T_i$  gives extra poor elements to a thread,  $T_i$  increases the counter of the corresponding thread.

### 3.2.5 Contention Manager (CM)

In order to eliminate livelocks caused by repeated rollbacks, threads talk to a Contention Manager (CM). Its purpose is to pause on run-time the execution of some threads making sure that at least one will do useful work so that system throughput can never get stuck [115]. See Section 3.3 for approaches able to greatly reduce the number of rollbacks and yield a considerable speedup, even in the absence of enough parallelism. Contention managers avoid energy waste because of rollbacks and re-

duce dynamic power consumption, by throttling the number of threads that contend, thereby providing an opportunity for the runtime system to place some cores in deep low power states.

### 3.3 Contention Manager

The goal of the Contention Manager (CM) is to reduce the number of rollbacks and guarantee the absence of livelocks, if possible [74, 115].

We implemented and compared four contention techniques: the *Aggressive Contention Manager* (Aggressive-CM) [115], the *Random Contention Manager* (Random-CM), the *Global Contention Manager* (Global-CM), and the *Local Contention Manager* (Local-CM).

The Aggressive-CM and Random-CM are non-blocking schemes. As is usually the case for non-blocking schemes [14, 74, 86, 105, 115], we do not prove absence of livelocks for these techniques. Nevertheless, they are useful for comparison purposes as Aggressive-CM is the simplest to implement, and Random-CM has already been presented in the mesh generation literature [14, 86, 105].

The Global-CM is a blocking scheme and we prove that does not introduce any deadlock. (Blocking schemes are guaranteed not to introduce livelocks [22]).

The last one, Local-CM, is semi-blocking, that is, it has both blocking and non-blocking parts. Because of its (partial) non-blocking nature, we found it difficult to prove starvation-freedom [74, 75], but we could guarantee absence of deadlocks and livelocks. It should be noted, however, that we have never experience any thread starvation when using Local-CM: all threads in all case studies are making progress concurrently for about the same period of time.

Note that none of the earlier Transactional Memory techniques [74, 115] and the Random Contention Managers presented in the past [14, 86, 105] solve the livelock problem. In this section, we show that if livelocks are not provably eliminated in our application, then termination is compromised on high core counts.

For the next of this Section assume that (without loss of generality) each thread

always finds elements to refine in its Poor Element List (PEL). This assumption simplifies the presentation of this Section, since it hides several details that are mainly related to Load Balancing. The interaction between the Load Balancing and the Contention Manager techniques does not invalidate the proofs of this Section.

### 3.3.1 Aggressive-CM

The Aggressive-CM is a brute-force technique, since there is no special treatment. Threads greedily attempt to apply the operation, and in case of a rollback, they just discard the changes, and move on to the next poor element to refine (if there is any). The purpose of this technique is to show that reducing the number of rollbacks is not just a matter of performance, but a matter of correctness. Indeed, experimental evaluation (see Section 3.3.5) shows that Aggressive-CM very often suffers from livelocks.

### 3.3.2 Random-CM

Random-CM has already been presented (with minor differences) in the literature [14, 86, 104, 105] and worked fairly well, i.e, no livelocks were observed in practice. This scheme lets “randomness” choose the execution scenario that would eliminate livelocks. We implement this technique as well to show that our application needs considerably more elaborate CMs. Indeed, recall that in our case, there is no much parallelism in the beginning of refinement and therefore, there is no much randomness that can be used to break the livelock.

Each thread  $T_i$  counts the number of consecutive rollbacks  $r_i$ . If  $r_i$  exceeds a specified upper value  $r^+$ , then  $T_i$  sleeps for a random time interval  $t_i$ . If the consecutive rollbacks break because an operation was successfully finished then  $r_i$  is reset to 0. The time interval  $t_i$  is in milliseconds and is a randomly generated number between 1 and  $r^+$ . The value of  $r^+$  is set to 5. Other values yielded similar results. Note that lower values for  $r^+$  do not necessarily imply faster executions. A low  $r^+$  decreases the number of rollbacks much more, but increases the number of times that a contented thread goes to sleep (for  $t_i$  milliseconds). On the other hand, a high  $r^+$  increases the

number of rollbacks, but randomness is given more chance to avoid livelocks; that is, a contented thread has now more chances to find other elements to refine before it goes to sleep (for  $t_i$  milliseconds).

Random-CM cannot guarantee the absence of livelocks. As noted in [22], this randomness can rarely lead to livelocks, but it should be rejected as it is not a valid solution. We also experimentally verified that livelocks are not that rare (see Section 3.3.5).

### 3.3.3 Global-CM

Global-CM maintains a global *Contention List* (CL). If a thread  $T_i$  encounters a rollback, then it writes its id in CL and it busy waits (i.e., it blocks). Threads waiting in CL are potentially awoken (in FIFO order) by threads that have made a lot of progress, or in other words, by threads that have not recently encountered many rollbacks. Therefore, each thread  $T_i$  computes its “progress” by counting how many consecutive successful operations  $s_i$  have been performed without an interruption by a rollback. If  $s_i$  exceeds a upper value  $s^+$ , then  $T_i$  awakes the first thread in CL, if any. The value for  $s^+$  is set to 10. Experimentally, we found that this value yielded the best results.

Global-CM can never create livelocks, because it is a blocking mechanism as opposed to random-CM which does not block any thread. Nevertheless, the system might end up to a deadlock, because of the interaction with the Load Balancing’s Begging List BL (see the Load Balancer in Section 3.2).

Therefore, at any time, the number of *active threads* needs to be tracked down, that is, the number of threads that do not busy wait in either the CL or the Begging List. A thread is forbidden to enter CL and busy wait, if it sees that there is only one (i.e., itself) active thread; instead, it skips CL and attempts to refine the next element in its Poor Element List. Similarly, a thread about to enter the Begging List (because it has no work to do) checks whether or not it is the only active thread at this moment, in which case, it awakes a thread from the CL, before it starts idling for extra work. In this simple way, the absence of livelocks and deadlocks are

guaranteed, since threads always block in case of a rollback and there will always be at least one active thread. The disadvantage of this method is that there is global communication and synchronization: the CL, and the number of active threads are global structures/variables that are accessed by all threads.

### 3.3.4 Local-CM

<pre> 1 <b>Algorithm:</b> Initialization(<math>T, i</math>)   <b>Input</b> : <math>T</math> is the array of threads,            <math>i</math> (<math>\geq 0</math>) is the id of the running thread            <math>T_i</math>.   /* <math>s</math> tracks down the progress of <math>T_i</math>. It      counts the number of consecutive operations      that finished successfully without      rollback. */ 2 <math>T[i].s = 0</math>;   /* <math>conflicting\_id</math> establishes dependencies.      If <math>conflicting\_id</math> is not a negative number      that means <math>T_i</math> rollbacks because it      attempted to acquire a vertex already owned      by <math>T_{conflicting\_id}</math>. */ 3 <math>T[i].conflicting\_id = -1</math>;   /* <math>busy\_wait</math> implements the busy waiting. */ 4 <math>T[i].busy\_wait = false</math>; </pre> <p>(a) It is called by each thread, before refinement starts.</p>	<pre> 1 <b>Algorithm:</b> Rollback_Occurred(<math>T, i, conflicting\_id</math>)   <b>Input</b> : <math>T</math> is the array of threads,            <math>i</math> (<math>\geq 0</math>) is the id of the running thread <math>T_i</math>            which attempted to acquire a vertex already locked by            the thread <math>T_{conflicting\_id}</math>.   /* The number of consecutive successful operations      is reset to 0. */ 2 <math>T[i].s = 0</math>; 3 <math>T[i].conflicting\_id = conflicting\_id</math>; 4 <math>T[\min(i, conflicting\_id)].mutex.lock()</math>; 5 <math>T[\max(i, conflicting\_id)].mutex.lock()</math>; 6 if <math>T[conflicting\_id].busy\_wait</math> then    /* <math>T_{conflicting\_id}</math> is very likely to be busy       waiting; to avoid cyclic dependencies, <math>T_i</math> is       forbidden to busy wait. */ 7   <math>T[i].conflicting\_id = -1</math>; 8   <math>T[\max(i, conflicting\_id)].mutex.unlock()</math>; 9   <math>T[\min(i, conflicting\_id)].mutex.unlock()</math>; 10  return; 11 end   /* <math>T_{conflicting\_id}</math> is not busy waiting; atomically, <math>T_i</math>      will. */ 12 <math>T[i].busy\_wait = true</math>; 13 <math>T[\max(i, conflicting\_id)].mutex.unlock()</math>; 14 <math>T[\min(i, conflicting\_id)].mutex.unlock()</math>;   /* <math>T_i</math> writes its id in <math>T_{conflicting\_id}</math>'s      Contention_List (CL). */ 15 <math>T[conflicting\_id].mutex.lock()</math>; 16 <math>T[conflicting\_id].CL.push\_back(i)</math>; 17 <math>T[conflicting\_id].mutex.unlock()</math>; 18 while <math>T[i].busy\_wait</math> do   /* <math>T_i</math> is busy waiting until thread <math>T_{conflicting\_id}</math>      wakes it up. */ 19 end 20 <math>T[i].conflicting\_id = -1</math>; </pre> <p>(c) <math>T_i</math> did not complete the operation because it encountered a rollback.</p>
(b) $T_i$ completed the operation.	

**Figure 3.2:** Pseudocode elaborating on the implementation of the local Contention Manager (local-CM).

The local Contention Manager (local-CM) distributes the previously global Contention List (CL) across threads. The Contention List  $CL_i$  of a thread  $T_i$  contains the ids of threads that encountered a rollback because of  $T_i$  (i.e. they attempted to

acquire a vertex already acquired by  $T_i$ ) and now they busy wait. As above, if  $T_i$  is doing a lot of progress, i.e., the number of consecutive successful operations exceed  $s^+$ , then  $T_i$  awakes one thread from its local  $CL_i$ .

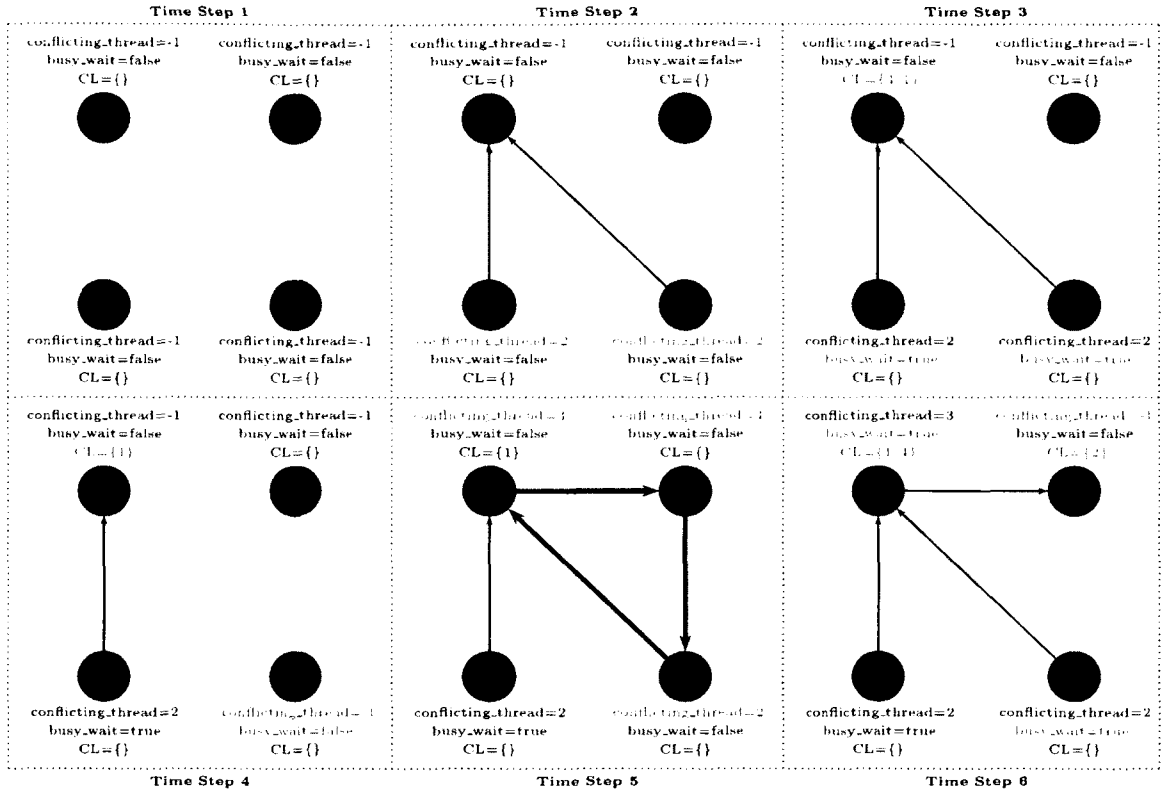
Extra care should be taken, however, to guarantee not only the absence of livelocks, but also, the absence of deadlocks. It is possible that  $T_1$  encounters a rollback because of  $T_2$  (and we symbolize this relationship by writing  $T_1 \rightarrow T_2$ ), and  $T_2$  encounters a rollback because of  $T_1$  (i.e.,  $T_2 \rightarrow T_1$ ): both threads write their ids to the other thread's CL, and no one else can wake them up. Clearly, this *dependency cycle* ( $T_1 \rightarrow T_2 \rightarrow T_1$ ) leads  $T_1$  and  $T_2$  to a deadlock, because under no circumstances these threads will ever be awoken again.

To solve these issues, each thread is now equipped with two extra variables: `conflicting_id` and `busy_wait`. See Figure 3.2 for a detailed pseudo-code of local-CM.

The algorithm in Figure 3.2c is called by a  $T_i$  every time it does not finish the operation successfully (i.e., it encounters a rollback). Suppose  $T_i$  attempts to acquire a vertex already locked by  $T_j$  ( $T_i \rightarrow T_j$ ). In this case,  $T_i$  does not complete the operation, but rather, it rolls back by disregarding the so far changes, unlocking all the associated vertices, and finally executing the `Rollback_Occurred` function, with `conflicting_id` equal to  $j$ . In other words, the `conflicting_id` variables represent dependencies among threads:  $T_i \rightarrow T_j \Leftrightarrow T_i.\text{conflicting\_id} = j$ .

For example, if  $T_i$  encounters a rollback because of  $T_j$  and  $T_j$  encounters a rollback because of  $T_k$ , then the dependency path from  $T_i$  is  $T_i \rightarrow T_j \rightarrow T_k$ , which corresponds to the following values:  $T_i.\text{conflicting\_id} = j, T_j.\text{conflicting\_id} = k, T_k.\text{conflicting\_id} = -1$  (where -1 denotes the absence of dependency).

Lines 4-14 of `Rollback_Occurred` decide whether or not  $T_i$  should block (via busy-waiting).  $T_i$  is not allowed to block if  $T_{\text{conflicting\_id}}$  has already decided to block (Lines 6-10). Threads communicate their decision to block by setting their `busy_wait` flags to true. If  $T_{\text{conflicting\_id}}.\text{busy\_wait}$  has already been set to true, it is imperative that  $T_i$  is not allowed to block, because it might be the case that the dependency of  $T_i$  forms a cycle. By not letting  $T_i$  to block, the dependency cycle “breaks”. Otherwise,  $T_i$  writes its id to  $CL_{\text{conflicting\_id}}$  (Lines 15-17) and loops around its `busy_wait` flag (Line 18).



**Figure 3.3:** Illustration of the local Contention Manager (local-CM). Six Time Steps demonstrate the interaction among four threads. The contents of their Contention List (CL), the value of the `conflicting_thread` variable, and the value of the `busy_wait` flag are shown.

The algorithm in Figure 3.2b is called by a  $T_i$  every time it completes an operation, i.e., every time  $T_i$  does not encounter a rollback. If  $T_i$  has done a lot of progress (Lines 2-5 of `Rollback_Not_Occurred`), then it awakes a thread  $T_j$  from its Contention List  $CL_i$  by setting  $T_j$ 's `busy_wait` flag to false. Therefore,  $T_j$  escapes from the loop of Line 18 in `Rollback_Occurred` and is free to attempt the next operation.

Figure 3.3 illustrates possible execution scenarios for local-CM during six Time Steps. Below, we describe in detail what might happen in each step:

- **Time Step 1:** All four threads are making progress without any rollbacks.
- **Time Step 2:**  $T_1$  and  $T_4$  attempted to acquire a vertex already owned by  $T_2$ . Both  $T_1$  and  $T_4$  call the code of Figure 3.2c. Their `conflicting_id` variables represent those exact dependencies (Line 3 of `Rollback_Occurred`).



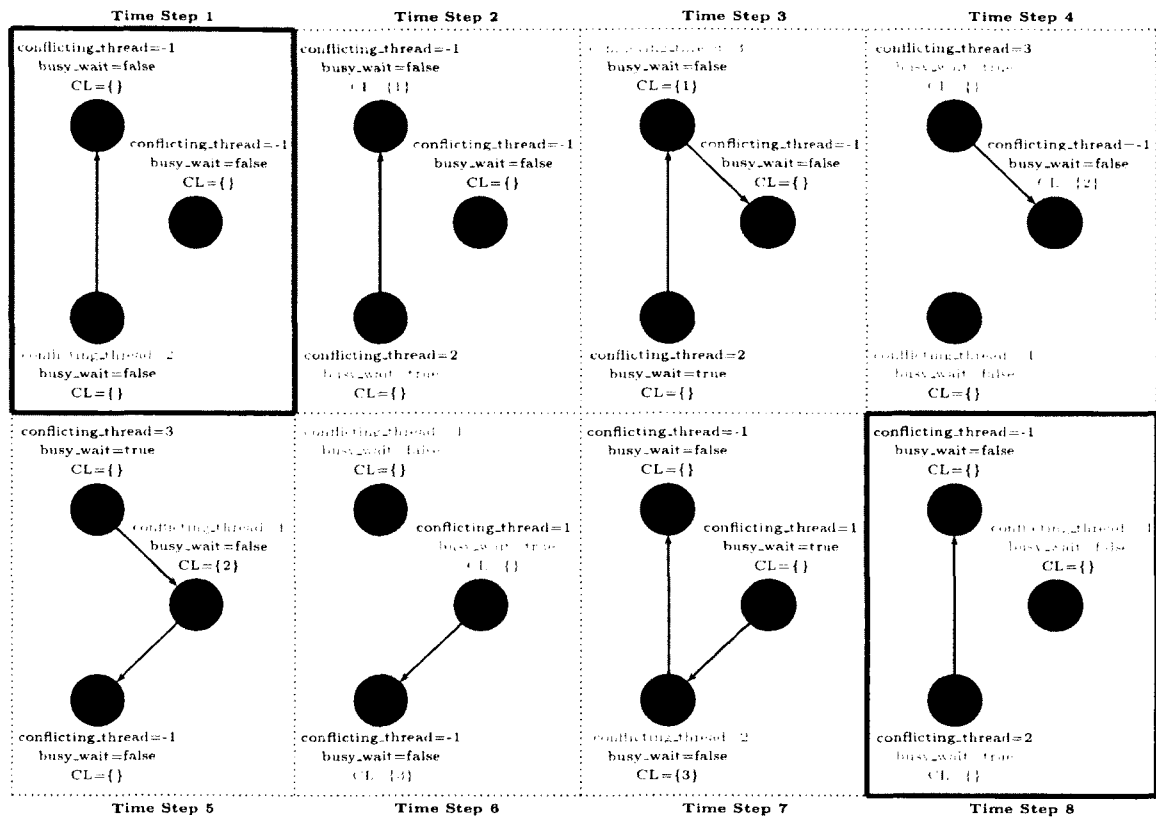
- **Time Step 3:**  $T_1$  and  $T_4$  set their `busy_wait` flag to true (Line 12 of `Rollback_Occurred`), they write their ids to `CL2` (Lines 15-17), and they block via a busy wait (Line 18).
- **Time Step 4:**  $T_2$  has done lots of progress and executes the Lines 6-9 of `Rollback_Not_Occurred`, awaking in this way  $T_4$ .
- **Time Step 5:** A dependency cycle is formed:  $T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$ . Lines 4-14 of `Rollback_Occurred` will determine which threads block and which ones do not. Note that the mutex locking of Lines 4-5 cannot be executed at the same time by these 3 threads. Only one thread can enter its critical section (Lines 6-14) at a time.
- **Time Step 6:** Here it is shown that  $T_4$  executed its critical section first,  $T_2$  executed its critical section second, and  $T_3$  was last. Therefore,  $T_4$  and  $T_2$  block, since the condition in Line 6 was false: their conflicting threads at that time had not set their `busy_wait` to true. The last thread  $T_3$  realized that its conflicting thread  $T_4$  has already decided to block, and therefore,  $T_3$  returns at Line 10, without blocking.

Note that in Time Step 6,  $T_2$  blocks without awaking the threads in its CL, and that is why both `CL2` and `CL3` are not empty. It might be tempting to instruct a thread  $T_i$  to awake all the threads in `CLi`, when  $T_i$  is about to block. This could clearly expedite things. Nevertheless, such an approach could easily cause a livelock as shown in Figure 3.4.

Local-CM is substantially more complex than global-CM, and the deadlock-free/livelock-free guarantees are not very intuitive. The rest of this Subsection is devoted to prove that local-CM indeed can never introduce deadlocks or livelocks.

The following two Remarks follow directly from the definition of deadlock and livelock [22].

**Remark 3.1** *If a deadlock arises, then there has to be a dependency cycle where all the participant threads block. Only then these blocked threads will never be awoken again.*



**Figure 3.4:** A thread about to busy-wait on another thread’s Contention List (CL) should not awake the threads already in its own CL. Otherwise, a livelock might happen, as illustrated in this Figure. Time Step 8 leads the system to the same situation of Time Step 1: this can be taking place for an undefined period of time with none of the threads making progress.

**Remark 3.2** *If a livelock arises, then there has to be a dependency cycle where all the participant threads are not blocked. Since all the participant threads break the cycle without making any progress, this “cycle breaking” might be happening indefinitely without completing any operations. In the only case where the rest threads of the system are blocked waiting on these participant threads’ Contention Lists (or all the system’s threads participate in such a cycle), then system-wide progress is indefinitely postponed.*

The next Lemmas prove that in a dependency cycle, at least one thread will block and at least one thread will not block. This is enough to prove absence of deadlocks and livelocks.

**Lemma 3.1 (Absence of deadlocks)** *In a dependency cycle at least one thread will not block.*

**Proof:** For the sake of contradiction, assume that the threads  $T_{i_1}, T_{i_2}, \dots, T_{i_n}$  participate in a cycle, that is,  $T_{i_1} \rightarrow T_{i_2} \rightarrow \dots \rightarrow T_{i_n} \rightarrow T_{i_1}$ , such that all threads block. This means that all threads evaluated Line 6 of Figure 3.2c to false. Therefore, since  $T_{i_1}$ 's `conflicting_id` is  $T_{i_2}$ , right before  $T_{i_1}$  decides to block (Line 12),  $T_{i_2}$ 's `busy_wait` flag was false. The same argument applies for all the pairs of consecutive threads:  $\{T_{i_2}, T_{i_3}\}, \{T_{i_3}, T_{i_4}\}, \dots, \{T_{i_n}, T_{i_1}\}$ . But  $T_{i_n}$  could not have evaluated Line 6 to false, because, by our assumption,  $T_{i_1}$  had already decided to block and  $T_{i_1}$ .`busy_wait` had been already set to true when  $T_{i_n}$  acquired  $T_{i_1}$ 's mutex. A contradiction:  $T_{i_n}$  returns from `Rollback_Occurred` without blocking. ■

**Lemma 3.2 (Absence of livelocks)** *In a dependency cycle at least one thread will block.*

**Proof:** For the sake of contradiction, assume that the threads  $T_{i_1}, T_{i_2}, \dots, T_{i_n}$  participate in a cycle, that is,  $T_{i_1} \rightarrow T_{i_2} \rightarrow \dots \rightarrow T_{i_n} \rightarrow T_{i_1}$ , such that all threads do not block. This means that all threads evaluated Line 6 of Figure 3.2c to true. Consider for example  $T_{i_1}$ . When  $T_{i_1}$  acquired  $T_{i_2}$ 's mutex, it evaluated Line 6 to true. That means that  $T_{i_2}$  had already acquired and released its mutex having executed Line 12: a contradiction because  $T_{i_2}$  blocks. ■

### 3.3.5 Comparison

For this case study, we evaluated each CM on the CT abdominal atlas of IRCAD Laparoscopic Center (<http://www.ircad.fr/>) using 128 and 256 Blacklight cores (see Table 3.2 for its specification). The final mesh consists of about  $150 \times 10^6$  tetrahedra. The single-threaded execution time on Blacklight was 1.080 seconds. See Table 3.1.

There are three direct sources of wasted cycles in our algorithm, and all of them are shown in Table 3.1:

**Table 3.1:** Comparison among Contention Managers (CM). A 150 Million element mesh is generated.

(a) 128 cores

	Aggressive-CM	Random-CM	Global-CM	Local-CM
time (secs)	n/a	64.2	23.7	19.3
rollbacks	n/a	$2.48251 \times 10^7$	728,087	680,338
contention overhead (secs)	n/a	4330.9	1081.4	545.80
load balance overhead (secs)	n/a	872.48	134.62	126.22
rollback overhead (secs)	n/a	516.81	3.0	2.9
total overhead (secs)	n/a	5720.9	1219.6	675.11
speedup	n/a	16.8	45.6	56.0
livelock	yes	no	not possible	not possible
deadlock	not possible	not possible	not possible	not possible

(b) 256 cores

	Aggressive-CM	Random-CM	Global-CM	Local-CM
time (secs)	n/a	n/a	22.3	14.1
rollbacks	n/a	n/a	882,768	$1.71197 \times 10^6$
contention overhead (secs)	n/a	n/a	3095.9	1377.1
load balance overhead (secs)	n/a	n/a	285.44	239.98
rollback overhead (secs)	n/a	n/a	3.6	7.6
total overhead (secs)	n/a	n/a	3385.1	1624.9
speedup	n/a	n/a	48.4	76.6
livelock	yes	yes	not possible	not possible
deadlock	not possible	not possible	not possible	not possible

- **contention overhead time:** it is the total time that threads spent busy-waiting on a Contention List (or busy-waiting for a random number of seconds as is the case of Random-CM) and accessing the Contention List (in case of Global-CM).
- **load balance overhead time:** it is the total time that threads spent busy-waiting on the Begging List waiting for more work to arrive (see Section 3.2) and accessing the Begging List, and
- **rollback overhead time:** it is the total time that threads had spent for the partial completion of an operation right before they decided that they had to

discard the changes and roll back.

Observe that Aggressive-CM was stuck in a livelock on both 128 and 256 cores. We know for sure that these were livelocks because we found out that no tetrahedron was refined, i.e., no thread actually made any progress, in the time period of an hour.

Random-CM terminated successfully on 128 cores, but it was very slow compared to Global-CM and Local-CM. Indeed, Random-CM exhibits a large number of rollbacks that directly increases both the contention overhead and the rollback overhead. Also, since threads' progress is much slower, threads wait for extra work for much longer, a fact that also increases the load balance overhead considerably. As we have already explained above, Random-CM does not eliminate livelocks, and this is manifested on the 256 core experiment, where a livelock did occur.

On both 128 and 256 cores, Local-CM performed better. Indeed, observe that the total overhead time is approximately twice as small as Global-CM's overhead time. This is mainly due to the little contention overhead achieved by Local-CM. Since Global-CM maintains a global Contention List, a thread  $T_i$  waits for more time before it gets awoken from another thread for two reasons: (a) because there are more threads in front of  $T_i$  that need to be awoken first, and (b) because the Contention List and the number of active threads are accessed by all threads which causes longer communication latencies.

Although Local-CM is the fastest scheme, observe that it introduces higher number of rollbacks on 256 cores than Global-CM. This also justifies the increased rollback overhead (see Table 3.1b). In other words, fewer rollbacks do not always imply faster executions, a fact that renders the optimization of our application a challenging task. This result can be explained by the following observation: the number of rollbacks (and subsequently, the rollback overhead) and the contention overhead constitute a tradeoff. The more a thread waits in a Contention List, the more its contention overhead is, but the fewer the rollbacks it encounters are, since it does not attempt to perform any operation. Conversely, the less a thread waits in a Contention List, the less its contention overhead is, but since it is given more chances to apply an operation, it might encounter more rollbacks. Nevertheless, Table 3.1 suggests that Local-CM does a very good job balancing this tradeoff on runtime.

**Table 3.2:** The specifications of the cc-NUMA machines we used.

	Model	cores per socket	sockets per blade	blades	memory per socket	max hops
Blacklight	Intel Xeon X7560	8	2	128	64GB	5
CRTC	Intel Xeon X5690	6	2	1	48GB	0

Although there are other elaborate and hybrid contention techniques [74, 115], none of them guarantees the absence of livelocks. Therefore, we chose Local-CM because of its efficiency and correctness.

### 3.4 Performance

In this Section, we describe a load balancing optimization and present the strong and weak scaling performance on Blacklight. See Table 3.2 for its specifications.

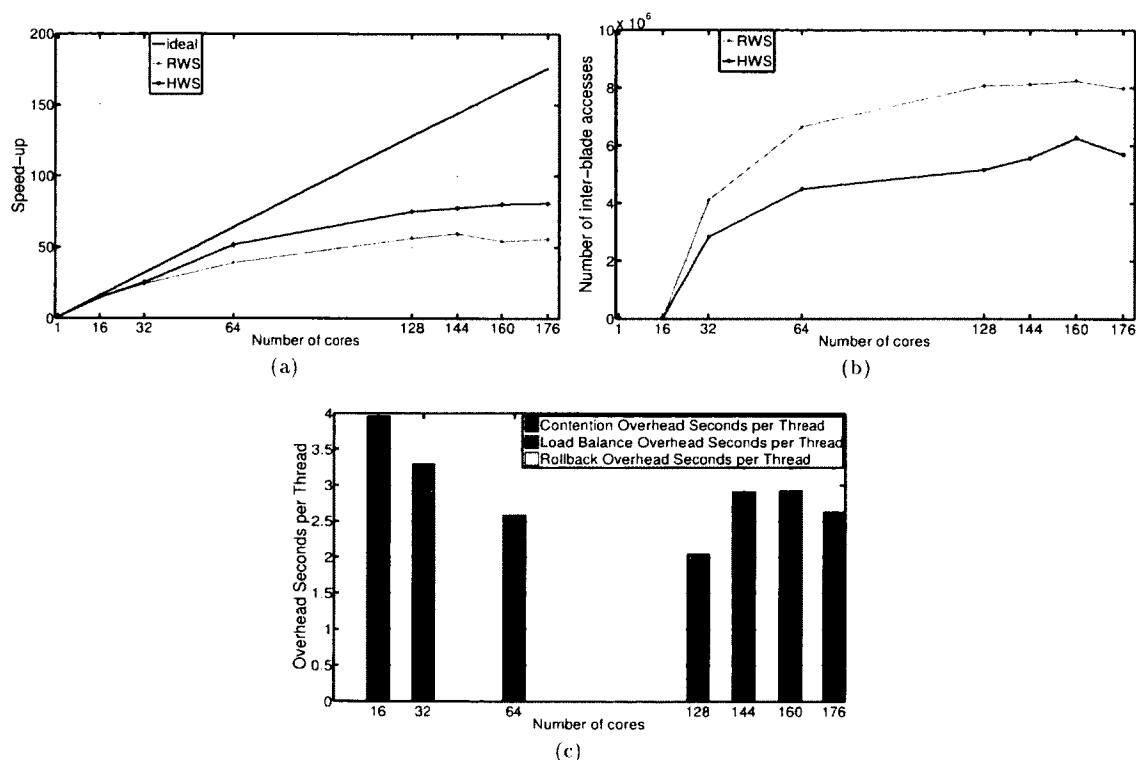
#### 3.4.1 Hierarchical Work Stealing (HWS)

In order to further decrease the communication overhead associated with remote memory accesses, we implemented a *Hierarchical Work Stealing* scheme (HWS) by taking advantage of the cc-NUMA architecture.

We re-organized the Begging List into three levels: BL1, BL2, and BL3. Threads of a single socket that run out of work place themselves into the first level begging list BL1 which is shared among threads of a single socket. If the thread realizes that all the other socket threads wait on BL1, it skips BL1, and places itself to BL2, which is shared among threads of a single blade. Similarly, if the thread realizes that BL2 already accommodates a thread from the other socket in its blade, it asks work by placing itself into the last level begging list BL3. When a thread completes an operation and is about to send extra work to an idle thread, it gives priority to BL1 threads first, then to BL2, and lastly to BL3 threads. In other words, BL1 is shared among the threads of a single socket and is able to accommodate up to *number\_of\_threads\_per\_socket* - 1 idle threads (in Blacklight, that is 7 threads). BL2 is shared among the sockets of a single blade and is able to accommodate up

to  $number\_of\_sockets\_per\_blade - 1$  idle threads (in Blacklight, that is 1 thread). Lastly, BL3 is shared among all the allocated blades and can accommodate at most one thread per blade. In this way, an idle thread  $T_i$  tends to take work first from threads inside its socket. If there is none,  $T_i$  takes work from a thread of the other socket inside its blade, if any. Finally, if all the other threads inside  $T_i$ 's blade are idling for extra work,  $T_i$  places its id to BL3, asking work from a thread of another blade.

### 3.4.2 Strong Scaling Results



**Figure 3.5:** Strong scaling performance achieved by the classic Random Work Stealing (RWS) and Hierarchical Work Stealing (HWS). (a)-(b) Comparison between RWS and HWS on speed-up ( $\frac{time_1}{time \times Threads}$ ) and on the number of inter-blade accesses. (c) Breakdown of the overhead time for HWS.

Figure 3.5 shows the strong scaling experiment demonstrating both the Random Work Stealing (RWS) load balance and the Hierarchical Work Stealing (HWS). The

**Table 3.3:** Information about the three input images used for the scaling results of Section 3.4 and the single-threaded performance comparison of Section 3.5.

	voxels	spacing (mm <sup>3</sup> )	tissues	download from
abdominal atlas	512 × 512 × 219	0.96 × 0.96 × 2.4	23	<a href="http://www.ircad.fr/softwares/3Dircadb/3Dircadb2/3Dircadb2.2.zip">http://www.ircad.fr/softwares/3Dircadb/3Dircadb2/3Dircadb2.2.zip</a>
knee atlas	512 × 512 × 119	0.27 × 0.27 × 1	49	<a href="http://www.spl.harvard.edu/publications/item/view/1953">http://www.spl.harvard.edu/publications/item/view/1953</a>
head-neck atlas	255 × 255 × 229	0.97 × 0.97 × 1.4	60	<a href="http://www.spl.harvard.edu/publications/item/view/2271">http://www.spl.harvard.edu/publications/item/view/2271</a>

input image we used is the CT abdominal atlas obtained from IRCAD Laparoscopic Center. Information about this input image is shown in Table 3.3. The final mesh generated consists of  $124 \times 10^6$  elements. On a single Blacklight core, the execution time was 1100 seconds.

Observe that the speed-up of RWS deteriorates by a lot for more than 64 cores (see the green line in Figure 3.5a). In contrast, HWS manages to achieve a (slight) improvement even on 176 cores. This could be attributed to the fact that the number of inter-blade (i.e., remote) accesses are greatly reduced by HWS (see Figure 3.5b), since begging threads are more likely to get poor elements created by threads of their own socket and blade first. Clearly, this reduces the communication involved when a thread reads memory residing in a remote memory bank. Indeed, on 176 cores, 98.9% of all the number of times threads asked for work, they received it from a thread of their own blade, yielding a 28.8% reduction in inter-blade accesses, as Figure 3.5b shows.

Figure 3.5c shows the breakdown of the overhead time per thread for HWS across runs. Note that since this is a strong scaling case study, the ideal behavior is a linear increase of the height of the bars with the respect to the number of threads. Observe, however, that the overhead time per thread is always below the overhead time measured on 16 threads. This means that Local-CM and the Hierarchical Work Stealing method (HWS) are able to serve threads fast and tolerate congestion efficiently on runtime.

### 3.4.3 Weak Scaling Results

In this section, we present the weak scaling performance of PI2M on two inputs, the information of which is presented in Table 3.3. The first is the same CT abdominal atlas already used in the previous strong scaling Section. The second input image



**Table 3.4:** Weak scaling performance. Across runs, the number of elements per thread remains approximately constant.

(a) abdominal atlas								
#Threads	1	16	32	64	128	144	160	176
#Elements	1.07E 07	1.72E 08	3.49E 08	7.44E 08	1.32E 09	1.51E 09	1.67E 09	1.85E 09
Time (secs)	90.37	80.03	87.50	99.23	93.00	103.26	150.03	181.10
Elements per second	1.18E 05	2.15E 06	3.99E 06	7.50E 06	1.42E 07	<b>1.46E+07</b>	1.11E 07	1.02E 07
Speedup	1.00	18.19	33.71	63.33	119.56	123.67	94.10	86.36
Efficiency	1.00	1.14	1.05	0.99	<b>0.93</b>	0.86	0.59	0.49
Overhead secs per thread	0.90	1.60	2.41	2.98	4.42	4.76	8.74	10.55

(b) knee-atlas								
#Threads	1	16	32	64	128	144	160	176
#Elements	1.06E 07	1.66E 08	3.70E 08	8.06E 08	1.31E 09	1.58E 09	1.70E 09	1.91E 09
Time (secs)	87.26	80.67	98.36	110.72	97.79	110.00	167.08	190.00
Elements per second	1.22E 05	2.05E 06	3.76E 06	7.28E 06	1.34E 07	<b>1.43E+07</b>	1.02E 07	1.01E 07
Speedup	1.00	16.89	30.92	59.90	110.61	117.92	83.77	82.81
Efficiency	1.00	1.06	0.97	0.94	<b>0.86</b>	0.82	0.52	0.47
Overhead secs per thread	0.87	1.46	2.77	3.41	5.47	6.58	8.90	11.07

is the knee atlas obtained from Brigham & Women’s Hospital Surgical Planning Laboratory [112]. Other inputs exhibit very similar results on comparable mesh sizes.

We measure the number of tetrahedra created per second across the runs. Specifically, let us define with  $\text{Elements}(n)$  and  $\text{Time}(n)$ , the number of elements created and the time elapsed, when  $n$  threads are employed. Then, the speedup is defined as  $\frac{\text{Elements}(n) \times \text{Time}(1)}{\text{Time}(n) \times \text{Elements}(1)}$ . With  $n$  threads, a perfect speedup would be equal to  $n$  [72].

We can directly control the size of the problem (i.e., the number of generated tetrahedra) via the parameter  $\delta$  (see Section 3.1). This parameter sets an upper limit on the volume of the tetrahedra generated. With a simple volume argument, we can show that a decrease of  $\delta$  by a factor of  $x$  results in an  $x^3$  times increase of the mesh size, approximately.

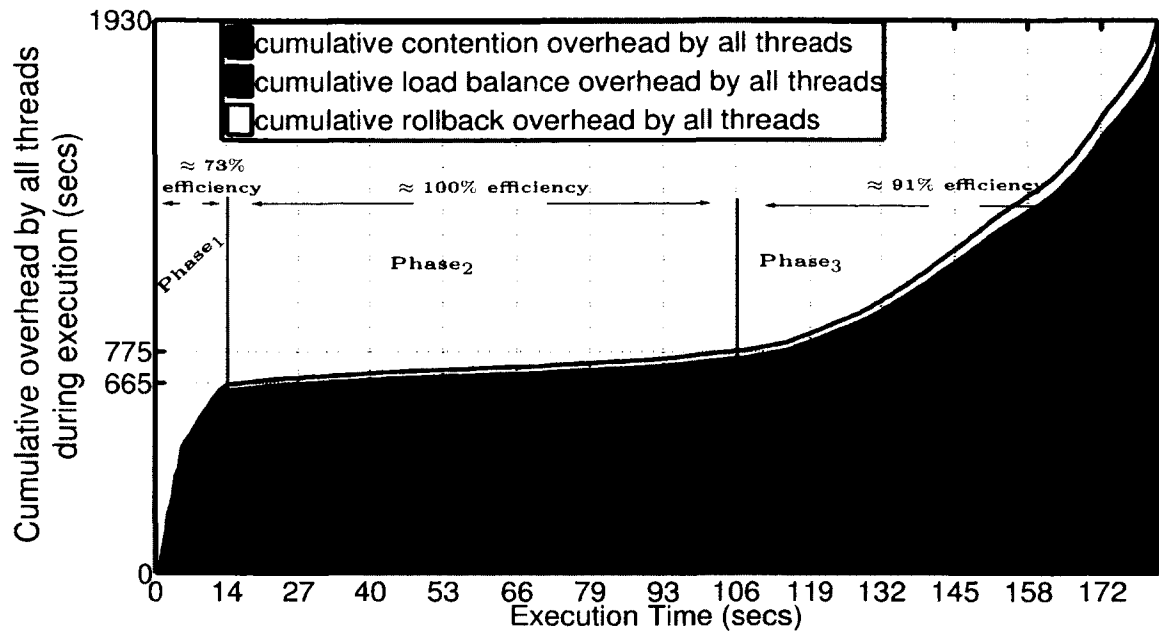
See Table 3.4. Each reported Time is computed as the average among three runs. Although the standard deviation for up to 128 cores is practically zero on both inputs, the same does not apply for higher core counts. Indeed, the standard deviation on the 144-, 160-, and 176-core executions is about 10, 15, and 29 seconds respectively, for both inputs. We attribute this behavior to the fact that in those experiments,

the network switches responsible for the cache coherency were close to the root of the fat-tree topology and therefore, they were shared among many more users, affecting in this way the timings of our application considerably. (Note that the increased bandwidth of the upper level switches does not alleviate this problem, since the bottleneck of our application is latency.) This conjecture agrees with the fact that the the maximum number of hops on the experiments for up to 128 cores was 3, while for 144, 160 and 176 cores, this number became 5.

Nevertheless, observe the excellent speedups for up to 128 threads. On 144 cores, we achieve an unprecedented efficiency of more than 82%, and a rate of more than 14.3 Million Elements per second for both inputs. It is worth mentioning that CGAL [6], the fastest sequential publicly available Isosurface-based mesh generation tool, on the same CT abdominal (<http://www.ircad.fr/software/3Dircadb/3Dircadb2/3Dircadb2.2.zip>) image input, is 81% slower than our single-threaded performance. Indeed, CGAL took 548.21 seconds to generate a similarly-sized mesh ( $1.00 \times 10^7$  tetrahedra) with comparable quality and fidelity to ours (see Section 3.5 for a more thorough comparison case study). Thus, compared to CGAL, the speedup we achieve on 144 cores is 751.25.

Observe, however, that our performance deteriorates beyond this core count. We claim that the main reason of this degradation is not the overhead cycles spent on rollbacks, contention lists, and begging lists (see Section 3.3.5), but the congested network responsible for the communication. Below, we support our claim.

First of all, notice that the total overhead time per thread increases. Since this is a weak scaling case study, the best that can happen is a constant number of overhead seconds per thread. But this is not happening. The reason is that in the beginning of refinement, the mesh is practically empty: only the six tetrahedra needed to fill the virtual box are present (see Figure 3.1). Therefore, during the early stages of refinement, the problem does not behave as a weak scaling case study, but as a strong scaling one: more threads, but in fact the same size, which renders our application a very challenging problem. See Figure 3.6 for an illustration of the 176-core experiment of Table 3.4a. X-axis shows the wall-time clock of the execution. The Y-axis shows the total number of seconds that threads have spent on useless



**Figure 3.6:** Overhead time breakdown with respect to the wall time for the experiment on 176 cores of Table 3.4a. A pair  $(x, y)$  tells us that up to the  $x^{\text{th}}$  second of execution, threads have not been doing useful work so far for  $y$  seconds all together.

computation (i.e., rollback, contention, and load balance overhead, see Section 3.3.5) so far, cumulatively. The more straight the lines are, the more useful work the threads perform. Rapidly growing lines imply lack of parallelism and intense contention. Observe that in the first 14 seconds of refinement (Phase<sub>1</sub>), there is high contention and severe load imbalance. Nevertheless, even in this case,  $\frac{176 \times 14 - 665}{176 \times 14} \approx 73\%$  of the time, all 176 threads were doing useful work, i.e., the threads were working on their full capacity.

However, this overhead time increase cannot explain the performance deterioration. See for example the numbers on 176 threads of Table 3.4a. 176 threads run for 181.10s each, and, on average, they do useless work for 10.55s each. In other words, if there were no rollbacks, no contention list overhead, and no load balancing overhead, the execution time would have to be  $181.10s - 10.55s = 170.55s$ . 170.55s, however, is far from the ideal 90.37s (that the first column with 1 thread shows) by  $170.55s - 90.37 = 80.18s$ . Therefore, while rollbacks, contention management, and load balancing introduce a merely 10.55s overhead, the real bottleneck is the 80.18s overhead spent on

memory (often remote) loads/stores. Indeed, since the problem size increases linearly with respect to the number of threads, either the communication traffic per network switch increases across runs, or it goes through a higher number of hops (each of which adds a 2,000 cycle latency penalty [4]), or both. It seems that after 144 cores, this pressure on the switches slows performance down. A hybrid approach [46] able to scale for larger network hierarchies is left for future work.

**Table 3.5:** Hyper-threaded execution of the case study shown in Table 3.4a. The columns of the Speedup, TLB misses, LLC misses, and Resource stall cycles reported here are relative to the non hyper-threaded execution of Table 3.4a on the same number of cores.

# Cores (2 threads per core)	1	16	32	64	128	144	160	176
# Elements	1.07E · 07	1.72E · 08	3.49E · 08	7.44E · 08	1.32E · 09	1.51E · 09	1.67E · 09	1.85E · 09
Time (secs)	58.03	55.98	61.57	67.28	240.36	342.91	436.72	480.83
Elements per second	1.84E · 05	3.08E · 06	5.67E · 06	1.11E · 07	5.48E · 06	4.41E · 06	3.83E · 06	3.85E · 06
Speedup	1.56	1.43	1.42	<b>1.47</b>	0.39	0.30	0.34	0.38
Overhead secs per thread	1.16	2.55	3.64	4.55	39.60	111.18	91.85	143.37
TLB misses increase per thread	-13.20%	-16.79%	-18.21%	-16.63%	-22.68%	-28.87%	-34.38%	-34.49%
LLC misses increase per thread	81.72%	-39.72%	-34.81%	-46.63%	-67.71%	-58.01%	-72.98%	-63.08%
Resource stall cycles increase per thread	-46.73%	-50.24%	-47.94%	-48.12%	-38.38%	-37.18%	-49.44%	-43.26%

### 3.4.3.1 Hyper-threading

Table 3.5 shows the performance achieved by the hyper-threaded version of our code. For this case study, we used the same input and parameters as the ones used in the experiment shown in Table 3.4a. The only difference is that now there are twice as many threads as there were in Table 3.4a.

Since the hardware threads share the TLB, the cache hierarchy, and the pipeline, we report the impact of hyper-threading on TLB misses, Last Level Cache (LLC) misses, and Resource stall cycles. Specifically, we report the increase of those counters relatively to the non hyper-threaded experiment of Table 3.4a. The reported Speedup is also relative to the non hyper-threaded experiment.

The last three rows of Table 3.5 suggest that the hyper-threaded version utilized the core resources more efficiently. Surprisingly enough, the TLB and LLC misses

actually decrease (notice the negative sign in front of the percentages) when two hardware threads are launched per core. Also, as expected, the pipeline in the hyper-threaded version is busier executing micro-ops, as the decrease of resource stall cycles suggest.

Although hyper-threading achieves a better utilization of the TLB, LLC, and pipeline, there is a considerable slowdown after 64 cores (i.e., 128 hardware threads). Observe that hyper-threading expedited the execution for up to 64 cores. Indeed, the hyper-threaded version is 47% faster on 64 cores compared to the non hyper-threaded version. Beyond this point, however, there is a considerable slowdown. This slowdown cannot be explained by merely the increase in the number of overhead seconds.

See for example the overhead secs per thread on 176 cores in Table 3.5. It is indeed 13 times higher than its non hyper-threaded counterpart; this is, however, expected because the size of the problem is the same but now we use twice as many hardware threads as before. If we subtract the overhead time of the hyper-threaded version on 176 cores, we get that for  $480.83s - 143.37s = 337.46s$ , all hardware threads were doing useful work. But this is still way longer than the  $181.10s - 10.55s = 170.55s$  useful seconds of the non hyper-threaded execution (see Table 3.4a).

We attribute this behavior to the increased communication traffic caused not by the increased problem size (as was mostly the case in the non hyper-threaded version), but by the increased number of “senders” and “receivers”. That is, even though the problem size is the same, the hyper-threaded version utilizes more threads. This means that at a given moment, there will be more packages (originated by the more than before threads) in the switches waiting to be routed than before. This phenomenon increases the communication latency. It seems that the network cannot handle this pressure for more than 64 cores, or equivalently, 128 hardware threads. Note that this agrees with the fact that in the non hyper-threaded version, the slowdown occurred on more than 128 cores, which is again 128 threads (see Table 3.4).

### 3.5 Single-threaded evaluation

Although PI2M introduces extra overhead due to locking, synchronization, contention management bookkeeping (see Section 3.3), and hierarchical load balance (see Section 3.4.1), in this Section we show that the single-threaded performance of our method (PI2M) is better than the performance of CGAL [6] and TetGen [121], the state-of-the-art sequential open source mesh generation tools. Moreover, PI2M has comparable quality with CGAL and much better quality than TetGen. PI2M, CGAL, and TetGen are very robust Delaunay methods, since they all use exact predicates. Specifically, PI2M adopts the exact predicates as implemented in CGAL [6, 54].

It should be mentioned that although CGAL is able to operate directly on segmented multi-tissue images (i.e., it is an Isosurface-based method), TetGen is a PLC-based method (see Section 1.1). That is, TetGen’s inputs are triangulated domains that separate the different tissues. For this reason, we pass to TetGen the triangulated iso-surfaces as recovered by our method, and then let TetGen to fill the underlying volume.

We ran PI2M, CGAL, and TetGen on two different multi-tissue 3D input images obtained from Brigham & Women’s Hospital Surgical Planning Laboratory (<http://www.spl.harvard.edu/>). The first is the MR knee-atlas [112] used in the previous Section and the second is a CT head-neck atlas [80]. Information about these two inputs is displayed in Table 3.3. The resulting output meshes generated by our method PI2M are illustrated in Figure 3.7. We should emphasize that we do not perform any smoothing as a post-processing step, since smoothing tends to deteriorate quality. In fact, in our previous work [61, 66], we show that quality is of great importance in the speed and accuracy of certain applications, such as non-rigid brain registration, and it should not be compromised. Nevertheless, mesh boundary smoothing is desirable for CFD simulations, such as respiratory airway modeling [57, 87, 88]. The extension of our framework to support the computationally expensive step of volume-conserving smoothing [87] and scale invariance [88] in parallel is left for future work.

For fair comparison, we also show the resulting output meshes generated by CGAL and TetGen in Figure 3.8 and Figure 3.9, respectively. A close investigation of the

meshes generated by TetGen (Figure 3.9) reveals that there are fewer labels than the labels recovered by PI2M and CGAL. In other words, the labels of TetGen do not correspond to the same labels of PI2M or CGAL. This is attributable to the way TetGen groups elements together [121] for visualization purposes. As mentioned earlier, the input PLC for TetGen is the set of the triangulated isosurfaces as recovered by PI2M. This PLC divides the domain into the subdomains that constitute the different tissues. In order for the elements of a subdomain  $A$  to be colored by a different label than the elements of a subdomain  $B$ , the user needs to specify two seed points  $p_A$  and  $p_B$ , such that  $p_A$  lies strictly in the interior of  $A$  and  $p_B$  lies strictly in the interior of  $B$ . A straightforward (perhaps not the best) way to compute these seeds is to traverse the input image and to assign a seed point per tissue. The unfortunate discrepancy with such an approach is that seeds might not lie in the intended PLC subdomains, simply because the recovered isosurfaces (that form the PLC) represent the actual tissue geometry within a tolerance (see Theorem 3.1). This problem affects only the visualization of TetGen meshes and it becomes more acute in our case, because there are many tissues that have very little volume, a reality that renders the computation of the appropriate seed points less accurate and robust in general. This fact alters the coloring of the TetGen meshes and this is the reason TetGen coloring does not completely agree with the coloring of the meshes generated by PI2M and CGAL.

**Table 3.6:** Statistics regarding the single-threaded performance and the quality/fidelity achieved by PI2M and CGAL. PI2M includes the extra overhead introduced by synchronization, contention management, and load balancing to support the (potential) presence of other threads.

	knee atlas			head-neck atlas		
	PI2M	CGAL	TetGen	PI2M	CGAL	TetGen
# tetrahedra	67,609	40,069	98,658	96,461	29,077	61,903
seconds	6.5 secs	10.9 secs	4.4 secs	10.3 secs	34.1 secs	16.0 secs
# tetrahedra	139,158	136,749	131,095	993,583	991,509	990,416
max radius-edge ratio	2	4.1	18.6	2	11.2	93.4
smallest boundary planar angle	17.4°	24.6°	18.0°	15.8°	2.4°	15.3°
(min, max) dihedral angles	(4.6°, 170.1°)	(2.5°, 176.3°)	(2.9°, 173.0°)	(4.5°, 170.2°)	(4.1°, 173.9°)	(0.4°, 172.0°)
Hausdorff distance	10.7 mm	10.3 mm	-	15.3 mm	15.2 mm	-

Table 3.6 shows timings and quality statistics for PI2M, CGAL, and TetGen. We used CRIC (see Table 3.2 for its specifications) for this case study. The timings reported account for everything but for disk IO operations. The execution time

reported for PI2M incorporates the 1.9 seconds and 1.2 seconds time interval needed for the computation of the Euclidean distance transform (see Section 3.1) for the knee atlas and the head-neck atlas, respectively.

We set the sizing parameters of CGAL and TetGen to values that produced meshes of similar size to ours, since generally, meshes with more elements exhibit better quality and fidelity. We assess the achieved quality of these methods in terms of radius-edge ratio and dihedral angles. Those metrics are of great importance to us, because they are shown to improve the speed and robustness of medical application solvers dealing with isotropic materials [37, 61, 66, 71, 119]. Ideally, the radius-edge ratio should be low, the minimum dihedral angle should be large, and the maximum dihedral angle should be low. We also report the smallest boundary planar angles. This measures the quality of the mesh boundary. Large smallest boundary planar angles imply better boundary quality.

PI2M, CGAL, and TetGen allow users to specify the target radius-edge ratio. Apart from TetGen, these methods also allow users to specify the target boundary planar angles. We set the corresponding parameters accordingly, so that the maximum radius-edge ratio is 2 (for PI2M, CGAL, and TetGen), and the smallest boundary planar angle is more than  $30^\circ$  (for PI2M and CGAL only, since TetGen does not give this parameter).

Fidelity measures how well the mesh boundary represents the iso-surfaces. We assess the fidelity achieved by these methods in terms of the symmetric (double-sided) Hausdorff distance. A low Hausdorff distance implies a good representation. Notice that we do not report the Hausdorff distance for TetGen, since the triangular mesh that represents the iso-surfaces is given to it as an input. For the input images we used for Table 3.6, the Hausdorff distances achieved by both PI2M and CGAL are far from ideal. This happens because the values chosen for the sizing parameters at this comparison did not recover isolated clusters of voxels which seem to be artifacts of the segmentation anyway. Nevertheless, Theorem 3.1 guarantees (both in theory and in practice) that if the sample is very dense, then the Hausdorff distance approaches to zero. The goal of this Section is not to generate meshes of high fidelity, but to demonstrate the effectiveness of PI2M by comparing PI2M with the state of the art

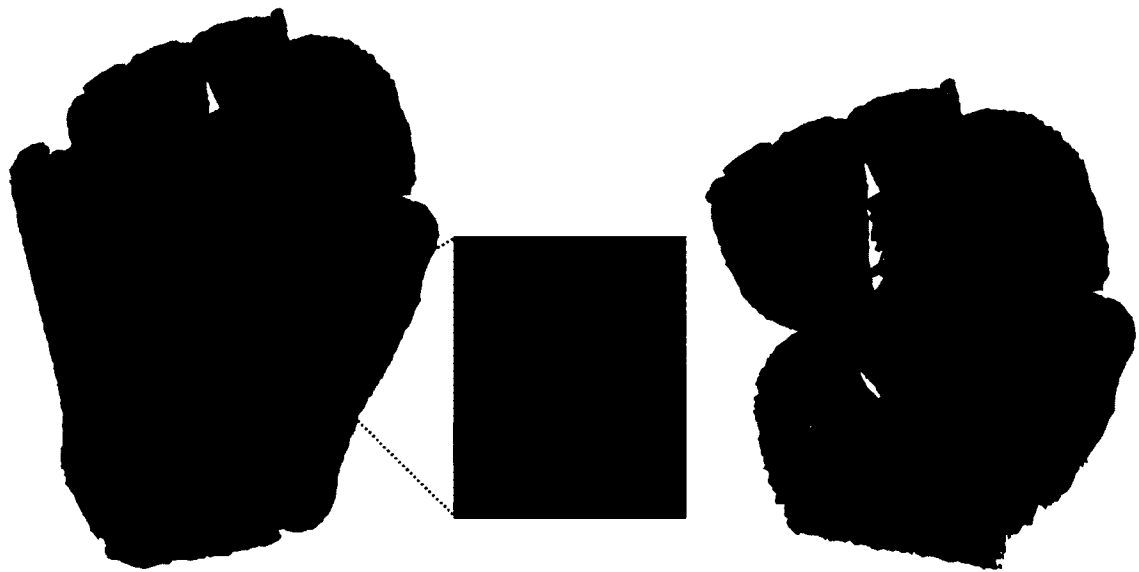


open source meshers.

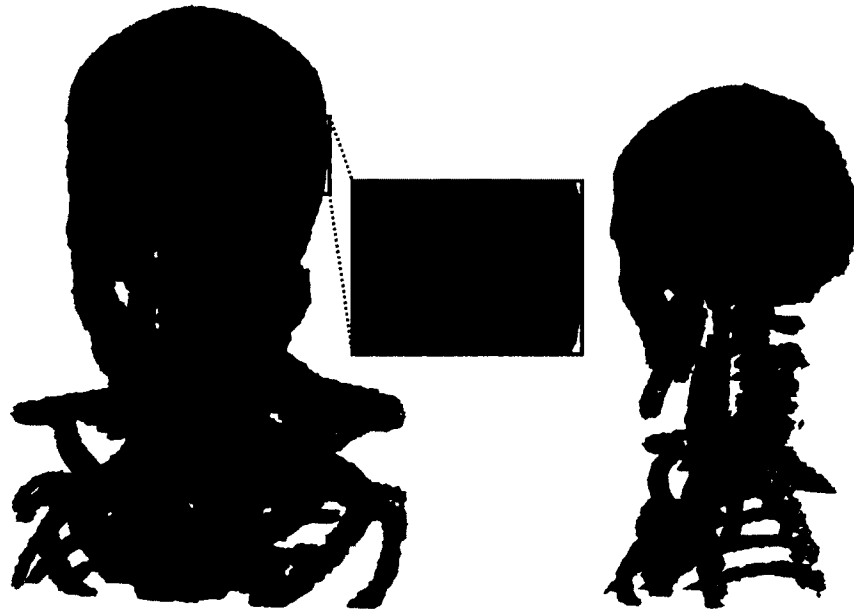
We access the speed of the methods above by comparing the rate of generated tetrahedra per second. Note that since our method not only inserts but also removes points from the mesh (thus reducing the number of mesh elements), a perhaps fairer way to access speed is to compare the rate of performed operations per second. Nevertheless, we do not report this metric for two reasons. First, a high rate of operations does not always imply a high rate of generated tetrahedra. The later, however, is the only thing that matters, since comparing the quality/fidelity achieved by meshes of very different mesh sizes makes no sense. Second, the number of removals performed by PI2M accounts for only 2% over the total number of operations. Thus, the rate of generated tetrahedra is very close the rate of operations per second; indeed, we experimentally found out that those two rates are practically the same.

Observe that the PI2M and CGAL generate meshes of similar dihedral angles, and fidelity, but our method is much faster. Indeed, the rate of the single-threaded PI2M is 68.7% higher than CGAL on the knee atlas and more than 3 times higher on the head-neck atlas. Also note that both PI2M and CGAL prove that the smallest boundary planar angles are more than  $30^\circ$  and that radius-edge ratio is less than 2 [62]. Due to numerical errors, however, these bounds might be smaller in practice than what theory suggests. Nevertheless, observe that PI2M yields much better boundary planar angles and radius-edge ratio than CGAL on the head-neck atlas.

TetGen is faster than PI2M only on the knee atlas by a couple of seconds. For larger meshes (as is the case with the head-neck atlas), TetGen is slower. Indeed, for small meshes, the computation of the Euclidean Distance Transform (EDT) accounts for a considerable percentage over the total execution time, a fact that slows down the overall execution time by a lot. For example, the actual meshing time on the knee atlas was just 4.6 secs, very close to TetGen's time and rate. Another notable observation is that our method generates meshes with much better dihedral angles and radius-edge ratio than TetGen. The achieved boundary planar angles are similar simply because the PLC that is given to TetGen was in fact the triangular boundary mesh of PI2M.

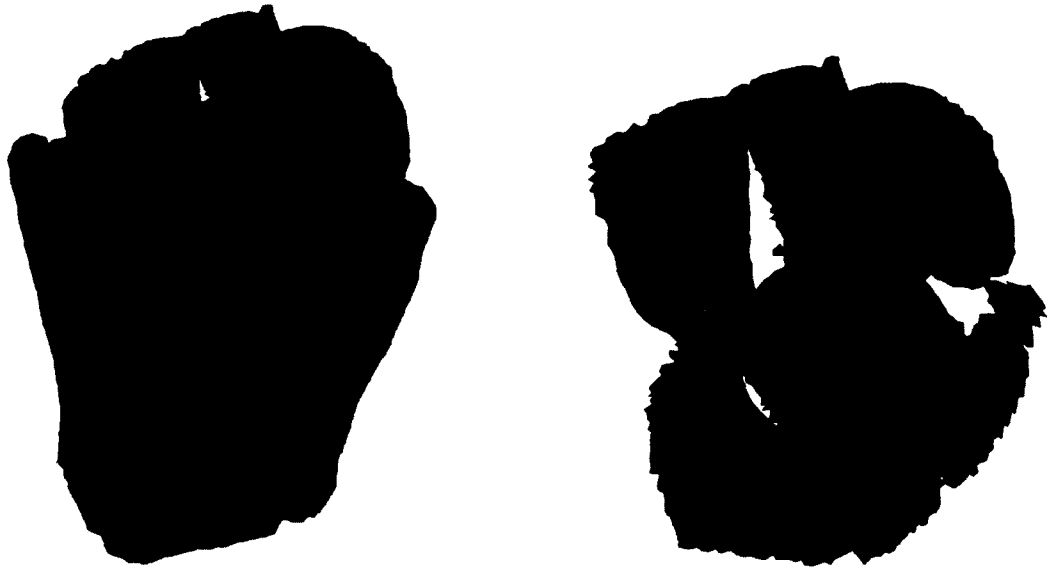


(a) The 439,458 element mesh generated for the MR knee atlas.



(b) The 993,583 element mesh generated for the CT head-neck atlas.

**Figure 3.7:** Output meshes generated by PI2M on the MR knee atlas and on the CT head-neck atlas.



(a) The 136,749 element mesh generated for the MR knee atlas.

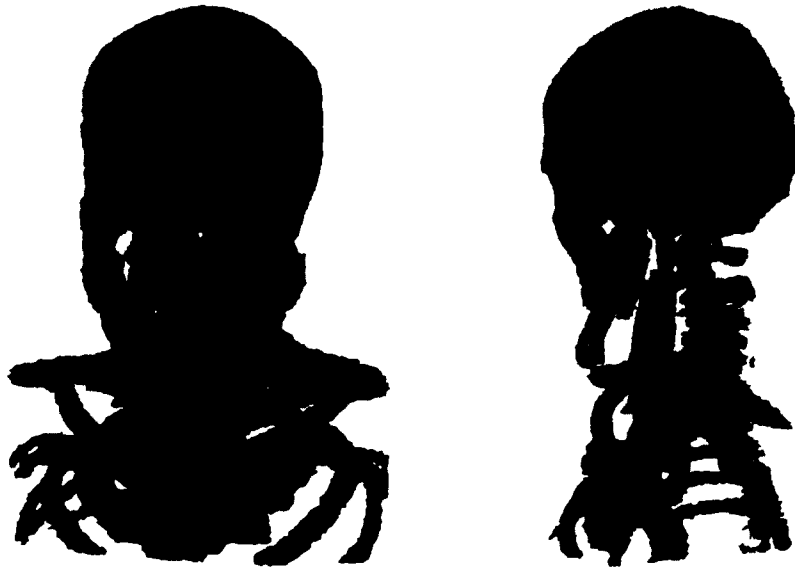


(b) The 991,509 element mesh generated for the CT head-neck atlas.

**Figure 3.8:** Output meshes generated by CGAL on the MR knee atlas and on the CT head-neck atlas.



(a) The 434,095 element mesh generated for the MR knee atlas.



(b) The 990,416 element mesh generated for the CT head-neck atlas.

**Figure 3.9:** Output meshes generated by TetGen on the MR knee atlas and on the CT head-neck atlas.

## Chapter 4

# 4D Space-Time Delaunay Meshing for Medical Images

In this chapter, we present a Delaunay refinement algorithm for 4-dimensional ( $3D+t$ ) segmented images. The output mesh is proved to consist of sliver-free simplices. Assuming that the hyper-surface is a closed smooth manifold, we also guarantee faithful geometric and topological approximation. We implement and demonstrate the effectiveness of our method on publicly available segmented cardiac images.

## 4.1 Preliminaries

The input of our algorithm is a segmented  $n$  dimensional image  $\mathcal{I} \subset \mathbb{R}^n$ . The object  $\Omega \subseteq \mathcal{I}$  is assumed to be represented as a cut function  $f : \mathbb{R}^n \mapsto \mathbb{R}$ , such that its surface  $\partial\Omega$  is defined by the set  $\{f(p) = 0\}$  [89, 109]. Clearly, from a segmented image, the zero-surface  $\{f(p) = 0\}$  can be easily computed by interpolating the voxel values.

We assume that given a point  $p \in \mathbb{R}^d$ , we can ask for  $p$ 's closest point on  $\partial\Omega$ . This can be accomplished by an Euclidean Distance Transform (EDT) [52, 97]. Specifically, the EDT returns the voxel  $p' \in \partial\Omega$  which is closest to  $p$ . Then, we traverse the ray  $pp'$  and we compute the intersection between the ray and  $\partial\Omega$  by interpolating the positions of different signs [96]. Points on  $\partial\Omega$  are referred to as *feature* points.

The *local feature size*  $\text{lfs}_{\partial\Omega}(x)$  of a point  $x \in \partial\Omega$  is defined as the (closest) distance between  $x$  and the medial axis of  $\partial\Omega$ . Since  $\partial\Omega$  is smooth, the local feature size is bounded from below by a positive constant  $\text{lfs}_{\partial\Omega}$ , that is,  $\text{lfs}_{\partial\Omega}(x) > \text{lfs}_{\partial\Omega} > 0$ . Another useful property is that the local feature size is 1-Lipschitz, that is,

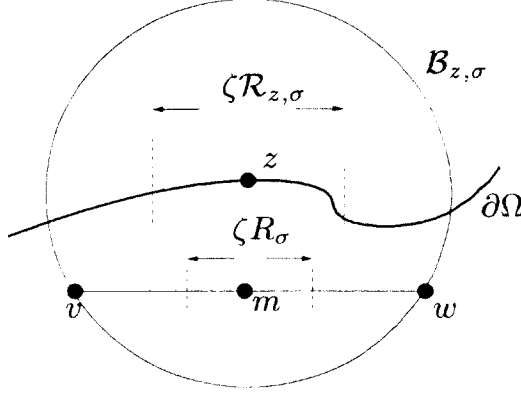
$$\text{lfs}_{\partial\Omega}(p) \leq |p - q| + \text{lfs}_{\partial\Omega}(q). \quad (4.1)$$

A point set  $V \subset \partial\Omega$  is called an  $\varepsilon$ -sample, if for every point  $p \in \partial\Omega$  there is a point  $v \in V$  at a distance at most  $\varepsilon \cdot \text{lfs}_{\partial\Omega}(p)$  from  $p$  [10].

Let  $V$  be a finite set of vertices  $V = \{v_1, \dots, v_N\} \subset \mathbb{R}^n$ . The *Delaunay triangulation* of  $V$  is denoted by  $\mathcal{D}(V)$ . A  $k$ -*simplex*  $\sigma_k = \{v_1, \dots, v_{k+1}\} \in \mathcal{D}(V)$  is a simplex defined by  $k + 1$  vertices. We denote the length of the shortest edge of a simplex  $\sigma$  with  $l_{\min}(\sigma)$ . The *circumball*  $B_\sigma$  of a simplex  $\sigma$  is the smallest closed ball circumscribing  $\sigma$ 's vertices.  $R_\sigma$  is the circumradius length of the simplex and  $c(\sigma)$  is its circumcenter. The radius-edge ratio of a simplex  $\sigma$  is defined as  $\rho(\sigma) = \frac{R_\sigma}{l_{\min}(\sigma)}$ .

The *voronoi cell*  $\text{Vor}(v)$  of a vertex  $v \in V$  is the set  $\text{Vor}(v) = \{p \in \mathbb{R}^n \mid |v - p| \leq |q - p|, \forall q \in V\}$ . The *voronoi dual* of a simplex  $\sigma \in \mathcal{D}(V)$  is defined as the set  $\text{Vor}(\sigma) = \{\text{Vor}(v_i) \cap \text{Vor}(v_j) \mid \forall v_i, v_j \in \sigma\}$ .

The restriction of  $\mathcal{D}(V)$  to a topological space  $\mathcal{X}$  is denoted by  $\mathcal{D}_{|\partial\Omega}(\mathcal{X})V$ .  $\mathcal{D}_{|\partial\Omega}(\mathcal{X})V$  is a *simplicial complex* (as is  $\mathcal{D}(V)$ ) that contains simplices of  $\mathcal{D}(V)$



**Figure 4.1:** A 2D illustration. The simplex  $\sigma = \{v, w\}$  and its surface ball  $\mathcal{B}_{z,\sigma}$ .  $m$  is the midpoint of  $\sigma$ . Observe that since the radius  $\mathcal{R}_{z,\sigma}$  of  $\mathcal{B}_{z,\sigma}$  is larger than the radius  $R_\sigma = |m - v|$  of  $B_\sigma$ , the picking region of  $\sigma$  as defined here is larger than the picking region of [90].

whose voronoi dual intersects  $\mathcal{X}$  in a non-empty set. Consider a  $k$  simplex  $\sigma$  and let  $\text{Vor}(\sigma)$  intersect  $\mathcal{X}$  at a point  $z$ . Any ball centered at  $z$  circumscribing  $\sigma$  is called a *surface ball* [28]. The corresponding surface ball is denoted by  $\mathcal{B}_{z,\sigma}$  and its radius by  $\mathcal{R}_{z,\sigma}$ , in the sequel. By the definition of Voronoi diagrams,  $\mathcal{B}_{z,\sigma}$  does not contain any vertex of  $V$  in its interior.

Following the definition of [90], the metric we use to characterize the quality of a simplex  $\sigma_k$  is  $\tau_{\sigma_k} = \frac{\text{Vol}_{\sigma_k}}{l_{\min}(\sigma_k)^k}$ . Low values of  $\tau$  imply a poor-quality element.

**Definition 4.1 (Sliver [90])** *Simplex  $\sigma$  is a sliver if it contains a  $k$ -simplex  $\sigma_k$  ( $k \leq 4$ ) such that  $\rho(\sigma_k) < \bar{\rho}$ ,  $\tau_{\sigma_k} < \bar{\tau}$ , and for any  $m$ -simplex  $\sigma_m$  of  $\sigma$  ( $m < k$ ),  $\rho(\sigma_m) < \bar{\rho}$ ,  $\tau_{\sigma_m} \geq \bar{\tau}$ .*

The *picking region*  $\mathcal{PR}(\sigma_4)$  of a 4-simplex  $\sigma_4$  is defined as the 4-dimensional solid ball centered at  $c(\sigma_4)$  with radius  $\zeta R_{\sigma_4}$ ,  $\zeta < 1$ . Consider a restricted  $k$ -simplex  $\sigma_k$  and its surface ball  $\mathcal{B}_{z,\sigma}$   $k < 4$ . Its picking region  $\mathcal{PR}(\sigma_k)$  is the intersection between  $\partial\Omega$  and the 4-dimensional solid ball centered at  $z$  with radius  $\zeta \mathcal{R}_{z,\sigma}$ ,  $\zeta < 1$ . Note that  $\mathcal{PR}(\sigma_4)$  and  $\mathcal{PR}(\sigma_k)$  are contained in  $B_\sigma$  and  $\mathcal{B}_{z,\sigma}$ , respectively. Observe that the picking region of  $\sigma_k$  ( $k < 4$ ) is a topological  $k$ -ball and does not belong (necessarily) in the affine  $k$  dimensional space defined by  $\sigma_k$ . This is different than the definition in [90], where the picking regions are defined inside the intersection of  $B_\sigma$  with the affine space of  $\sigma$ . The reason for this change is the fact that the input of our algorithm

is not a *Piecewise Linear Complex* (PLC) but a cut function.

A good point  $p \in \mathcal{PR}(\sigma)$  is a point that does not introduce smaller slivers. A sliver is small when its radius is less than  $bR_\sigma$ . In [90], it is proved that (a) the number of small slivers  $S(\sigma)$  possibly created after the insertion of  $p$  is constant, and (b) the volume  $|F_\sigma|$  (the *forbidden region*) inside which  $p$  forms a small sliver is bounded from above. The same findings hold in our case too, where the picking region of a restricted facet  $\sigma_3$  is not inside the intersection of  $B_{\sigma_3}$  and  $\sigma_3$ 's affine space, but inside the intersection of  $\mathcal{B}_{z,\sigma_3}$  and  $\partial\Omega$ .

**Lemma 4.1** *Given an almost-good mesh, a point  $p$  inside the picking region of a  $\sigma_k$  can be found in a constant number of random rounds, such that any new sliver created after the insertion of  $p$  has circumradius no smaller than  $bR_{\sigma_k}$  if  $k = 4$ , or no smaller than  $b\mathcal{R}_{z,\sigma_k}$  if  $k = 3$ .*

**Remark 4.1** *The proof is similar to [90], since  $|F_\sigma|$  and  $S(r)$  do not change and the volume of the intersection of  $B_{\sigma_3}$  and  $\sigma_3$ 's affine space is smaller than the intersection of  $\mathcal{B}_{z,\sigma_3}$  and  $\partial\Omega$ . See Figure 4.1 for an illustration.*

## 4.2 Algorithm

The user specifies a parameter  $\delta$ . It will be clear in Section 4.4 that the lower  $\delta$  is, the better the mesh boundary will approximate  $\partial\Omega$ . For brevity, the quantity  $\delta \cdot \text{lfs}_{\partial\Omega}(z)$  is denoted by  $\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point.

Our algorithm initially inserts the 16 corners of a hyper-box that contains the 4 dimensional object  $\Omega$ , such that the distance between a box corner  $x$  and its closest feature point  $z = \text{cfp}_{\partial\Omega}(x)$  is at least  $2\Delta_{\partial\Omega}(z)$ . After the computation of this initial triangulation, the refinement starts dictating which extra points are inserted. At any time, the Delaunay triangulation  $\mathcal{D}(V)$  of the current vertices  $V$  is maintained. Note that by construction,  $\mathcal{D}(V)$  always covers the entire hyper-volume and that any point on the box is separated from  $\partial\Omega$  by a distance at least  $2\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point.



During the refinement, some vertices are inserted exactly on the box; these vertices are called *box vertices*. The box vertices might lie on 1, 2, or 3-dimensional box faces. We shall refer to the vertices that are neither box vertices nor feature vertices as *free vertices*.

The algorithm inserts new vertices for three reasons: to guarantee that (a)  $\partial\Omega$  is correctly recovered, (b) all the elements have small radius-edge ratio, and (c) there are no slivers. Specifically, for a 4-simplex  $\sigma_4$  in the mesh, the following rules are checked in this order:

- **R1:** Let  $B_{\sigma_4}$  intersect  $\partial\Omega$  and  $z$  be equal to  $\text{cfp}_{\partial\Omega}(c(\sigma_4))$ . If  $z$  is at a distance no closer than  $\Delta_{\partial\Omega}(z)$  to any other feature vertex, then  $z$  is inserted.
- **R2:** Let  $B_{\sigma_4}$  intersect  $\partial\Omega$  and  $z$  be equal to  $\text{cfp}_{\partial\Omega}(c(\sigma_4))$ . If  $R_\sigma \geq 2\Delta_{\partial\Omega}(z)$ ,  $c(\sigma_4)$  is inserted.
- **R3:** Let  $c(\sigma_4)$  lie inside  $\Omega$ . If  $\rho(\sigma_4) \geq \bar{\rho}$ ,  $c(\sigma_4)$  is inserted.
- **R4:** Let  $c(\sigma_4)$  lie inside  $\Omega$ . If  $\sigma_4$  contains a sliver, a good point inside  $\mathcal{PR}(\sigma_4)$  is inserted.
- **R5:** Let  $\sigma_3$  ( $\sigma_3 \subset \sigma_4$ ) be a restricted facet. If the vertices of  $\sigma_3$  are not feature vertices, then a good point  $z$  inside  $\mathcal{PR}(\sigma_3)$  is inserted. All the free vertices closer than  $\Delta_{\partial\Omega}(z)$  to  $z$  are deleted.

For  $i < j$ , priority is given to  $R_i$  over  $R_j$ . That is, right before the insertion of a point because of  $R_j$ , there is no element that violates a rule  $R_i$ . Also, in R4, priority is given to the lower dimensional slivers that  $\sigma_4$  might contain.

Whenever there is no simplex for which R1, R2, R3, or R4 apply, the refinement process terminates. *The final mesh reported is the set of pentatopes whose circumcenters lie inside  $\Omega$ .*

In a nutshell, R1 and R2 is responsible for generating a sufficiently dense sample on  $\partial\Omega$ . R5 makes sure that the vertices of the simplices restricted to  $\partial\Omega$  lie on  $\partial\Omega$  similarly to [109]. Lastly, R3 and R4 deal with the quality guarantees. In Section 4.3, we will show that there are values for  $b$ ,  $\zeta$ , and  $\bar{\rho}$  that do not compromise termination.

To prove termination, no vertices should be inserted outside the bounding box. Notice, however, that vertices inserted due to R2 may lie outside the bounding box. To deal with such cases,  $c(\sigma_4)$  is rejected for insertion. Instead, its *projection*  $c'(\sigma_4)$  on the box is inserted in the triangulation. That is,  $c'(\sigma_4)$  is the closest to  $c(\sigma_4)$  box point. In Section 4.3 and Section 4.4, we prove that the insertion of projected points do not compromise either quality or fidelity. Note that these projections are different than the traditional *encroachment rules* described in [117, 118].

Recall that pentatopes with circumcenters on  $\partial\Omega$  or outside  $\Omega$  are not part of the final mesh, and that is why rule R3 and R4 do not check them.

### 4.3 Termination and Quality

In this section, we will specify the appropriate values for  $\zeta$ ,  $\bar{\rho}$ , and  $b$ , so that the algorithm terminates. Specifically, we will show during refinement the shortest edge introduced into the mesh cannot be arbitrarily small.

Suppose that  $\sigma$  violates a rule  $R_i$ .  $\sigma$  is called an  $R_i$  element.  $R_i$  dictates the insertion of a point  $p$  (and possibly the removal of free points). Point  $p$  is called an  $R_i$  point. According to [117, 118], the *insertion radius*  $R_p$  of  $p$  is defined as the length of the shortest edge incident to  $p$  created right after the end of  $R_i$  and the *parent*  $Par(p)$  of  $p$  as the most recently inserted vertex incident to the shortest edge of  $\sigma$ .

**Lemma 4.2** *Let  $p$  and  $q$  define the shortest edge of a simplex  $\sigma$  and  $q$  being inserted after  $p$ . Then  $R_q \leq l_{\min}(\sigma)$ .*

**Proof:** Assume that right after the insertion of  $q$ ,  $p$  is the closest point to  $q$ . In this case,  $R_q = |p - q| = l_{\min}(\sigma)$ . Otherwise, there has to be another closest vertex to  $q$ , which implies that  $R_q < |p - q| = l_{\min}(\sigma)$ . ■

The following Lemma bounds from below the shortest edge introduced into the mesh after the insertion of a box vertex:

**Lemma 4.3** *Let  $v$  be a box vertex inserted into the mesh. Then,  $R_v \geq 2\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point.*

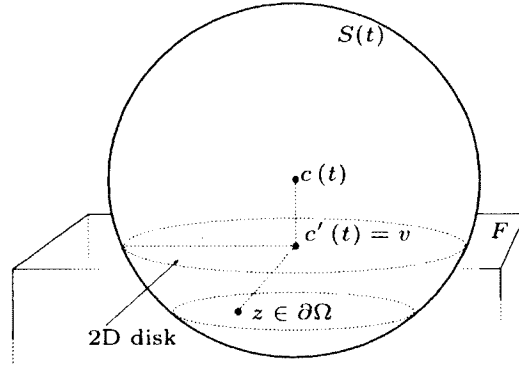


Figure 4.2: Proof of Lemma 4.3, a 3D illustration.

**Proof:** A box point  $v$  is inserted only because of R2. The circumcenter  $c(\sigma)$  of a pentatope  $\sigma$  lies on or outside the box and its projection  $c'(\sigma) = v$  falls on the box. Without loss of generality, assume that the projection lies on the interior of a 3-face (i.e. a box tetrahedron)  $F$ . See Figure 4.2 for a 3D illustration. (If  $c(\sigma)$  lies precisely on the box,  $c'(\sigma)$  is equal to  $v$ .) Consider the (2D) disk (drawn) of  $B_\sigma$  which is coplanar with  $F$ . That disk contains  $v$  and separates  $B_\sigma$  in two sides: the side that contains  $c(\sigma)$  and the side that contains a part of the box.

We claim that the closest vertex —say  $w$ — to  $v$  lies on the intersection of  $B_\sigma$ 's boundary and the ray  $\overrightarrow{c(\sigma)v}$ . To see why, note that  $B_\sigma$  is empty of vertices, and therefore, the closest to  $v$  that an arbitrary vertex  $w'$  already in the triangulation can be is when it lies on the boundary of  $B_\sigma$  and on the side of  $B_\sigma$  that contains a part of the box, as shown. Consider the triangle  $\Delta w'vc(\sigma)$ . From the law of cosines, we have that:

$$\begin{aligned}
 |v - w'|^2 &= |c(\sigma) - w'|^2 + |c(\sigma) - v|^2 - 2|c(\sigma) - w'| |c(\sigma) - v| \cos \omega \\
 &\geq |c(\sigma) - w'|^2 + |c(\sigma) - v|^2 - 2|c(\sigma) - w'| |c(\sigma) - v|, \text{ since } \cos \omega \leq 1 \\
 &= (|c(\sigma) - w'| - |c(\sigma) - v|)^2 \\
 &= (R_\sigma - |c(\sigma) - v|)^2, \text{ since } w' \text{ lies on the sphere} \\
 &= |v - w|^2,
 \end{aligned}$$

and our claim is proved.

Therefore, any possible new edge connected to  $v$  has length at least  $|v - w|$ . Since, however,  $\sigma$  triggers R2,  $B_\sigma$  has to intersect  $\partial\Omega$ . Therefore, there has to be a feature point  $q \in \partial\Omega$  (illustrated) inside  $B_\sigma$  and on the same side of  $F$  as  $w$ . Let us denote

with  $q'$ , the projection of  $q$  to the box face  $F$ . By construction,  $|q - q'|$  is at least  $2\Delta_{\partial\Omega}(z)$ , where  $z$  is a feature point. Observe, however, that  $|v - w|$  is always larger than  $|q - q'|$ , because  $vw \parallel qq'$ , and the statement holds. Similar reasoning applies in the case where  $c'(\sigma)$  lies on a box triangle or a box edge. ■

The following Lemma proves a lower bound on the lengths created into the mesh because of R1 and R2:

**Lemma 4.4** *Let  $p$  be a vertex inserted into the mesh because of R1 or R2. Then,  $R_p \geq \Delta_{\partial\Omega}(z)$ , where  $z$  is the closest feature point to  $p$ .*

**Proof:** If R1 is triggered, then  $p$  is equal to  $z$  and since there is no other feature point already inserted in the mesh closer than  $\Delta_{\partial\Omega}(p)$  to  $p$ , the statement holds. Otherwise, R2 applies for a simplex  $\sigma_4$  and  $p$  is equal to  $c(\sigma_4)$ . Due to the empty ball property,  $R_p$  is at least  $R_{\sigma_4} \geq 2\Delta_{\partial\Omega}(\text{cfp}_{\partial\Omega}(p))$ , and the statement holds. ■

**Lemma 4.5** *Let  $p$  be a vertex inserted into the mesh because R3 applies for an element  $\sigma$ . Then,  $R_p \geq \bar{\rho}R_{Par(p)}$ .*

**Proof:** Since  $p$  is equal to  $c(\sigma)$ ,  $R_p \geq R_\sigma = \rho(\sigma)l_{\min}(\sigma) \geq \bar{\rho}l_{\min}(\sigma)$ . Lemma 4.2 suggests that  $l_{\min}(\sigma) \geq R_{Par(p)}$ , and the results follows. ■

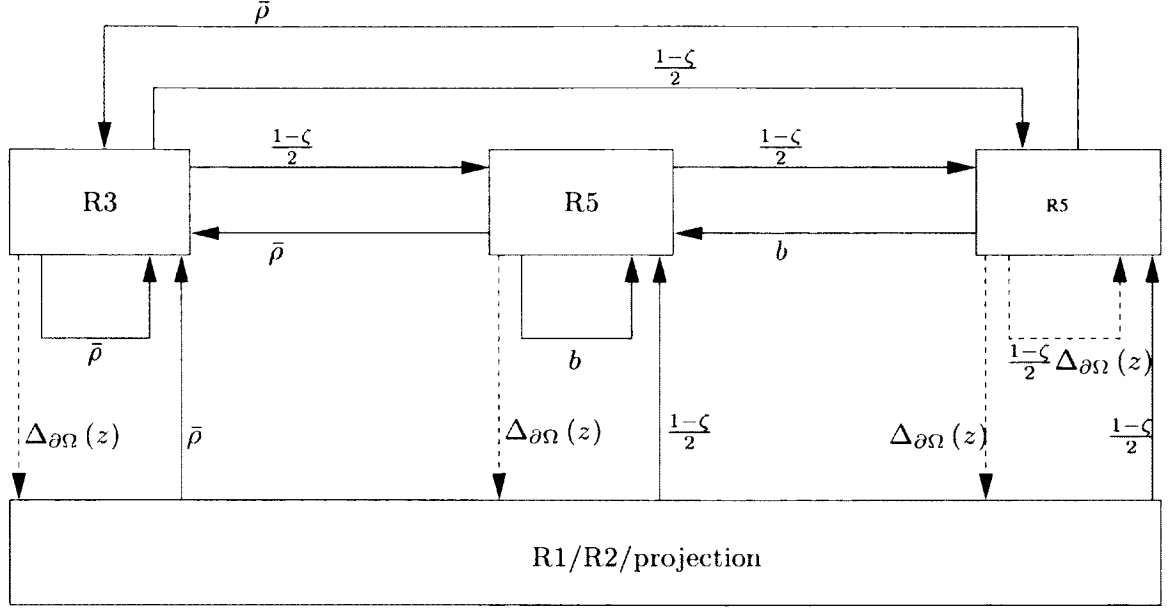
**Lemma 4.6** *Let  $p$  be inserted into the mesh because of R4. Then,*

- $R_p \geq \frac{1-\zeta}{2}R_{Par(p)}$ , if  $Par(p)$  is neither R4 nor R5,
- $R_p \geq bR_{Par(p)}$ , otherwise.

**Proof:** Let  $\sigma$  be the simplex that violates R4.

Suppose that  $Par(p)$  is neither R4 nor R5. Since  $p$  belongs to the picking region of  $\sigma$ ,  $R_p \geq (1 - \zeta)R_\sigma \geq \frac{1-\zeta}{2}l_{\min}(\sigma)$ . From Lemma 4.2, we have that  $R_p \geq \frac{1-\zeta}{2}R_{Par(p)}$ .

Otherwise, consider the case  $Par(p)$  is an R4 point. From Lemma 4.1, we know that the circumradius of  $\sigma$  is more than  $b$  times the circumradius of the R4 simplex  $\sigma'$  that inserted  $Par(p)$ . Therefore,  $R_p \geq (1 - \zeta)R_\sigma \geq (1 - \zeta)bR_{\sigma'}$ . However, the quantity  $(1 - \zeta)R_{\sigma'}$  is equal to  $R_{Par(p)}$ , and the statement holds.



**Figure 4.3:** Flow diagram depicting the relationship among the rules. No solid cycle should have a product less than 1. The dashed arrows break the cycle.

The exact same logic holds when  $Par(p)$  is an R5 point, by just substituting  $\mathcal{R}_{z,\sigma'}$  for  $R_{\sigma'}$ , where  $\sigma'$  is an R5 simplex.  $\blacksquare$

**Lemma 4.7** *Let  $p$  be inserted into the mesh because of R5. Then,*

- $\frac{1-\zeta}{2} R_{Par(p)}$ , if  $Par(p)$  is not an R5 point,
- $\Delta_{\partial\Omega}(z)$ , otherwise.

**Proof:** Let  $\sigma_3$  be the simplex that violates R4.

Suppose that  $Par(p)$  is not an R5 point. Because of Lemma 4.2, the shortest edge of  $\sigma_3$  is at least  $R_{Par(p)}$ . Therefore, any surface ball of  $\sigma_3$  has radius at least  $\frac{1}{2} R_{Par(p)}$ . Since the surface ball does not contain any vertex in its interior,  $R_p \geq \frac{1-\zeta}{2} R_{Par(p)}$ .

Suppose that  $Par(p)$  is an R5 point. Note that when  $Par(p)$  is inserted, all the free vertices closer than  $\Delta_{\partial\Omega}(Par(p))$  to  $Par(p)$  are deleted. Due to R5,  $\sigma_3$  contains at least one free vertex. Since  $Par(p)$  is the most recently inserted vertex incident to the closest edge of  $\sigma_3$ , the edge that contains  $Par(p)$  and the free vertex has to be at least

$\Delta_{\partial\Omega}(Par(p))$ . Therefore, any surface ball of  $\sigma_3$  has radius at least  $\frac{1}{2}\Delta_{\partial\Omega}(Par(p))$ . Hence,  $R_p \geq \frac{1-\zeta}{2}\Delta_{\partial\Omega}(Par(p))$ . ■

Putting all the Lemmas together, the solid arrows of Figure 4.3 show the insertion radius of the inserted point as a fraction of the insertion radius of its parent. An arrow from  $R_i$  to  $R_j$  with label  $x$  implies that the insertion radius of an  $R_j$  point  $p$  is at least  $x$  times larger than the insertion radius of its  $R_i$  parent  $Par(p)$ . The label  $x$  of the dashed arrows is the absolute value of  $R_p$ . Note that the labels of the dashed arrows depend on the local feature size of  $\partial\Omega$ , and as such are always positive constants.

Recall that during refinement, free vertices might be deleted (because of R5). Nevertheless, such deletions of vertices are always preceded by insertion of feature points. Considering the fact that feature vertices are never deleted from the mesh, termination is guaranteed if we prove that the insertion radii of the inserted vertices cannot decrease indefinitely. Clearly, [90, 117, 118], if there is no solid cycle of product less than 1, termination is guaranteed.

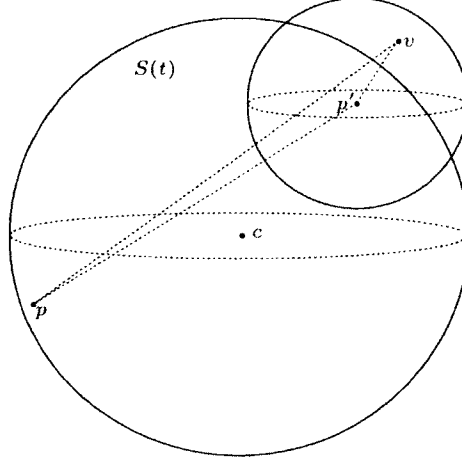
**Theorem 4.1** *The algorithm terminates producing simplices of bounded aspect ratio, if*

- $\frac{(1-\zeta)^2}{4}\bar{\rho} \geq 1$ , and
- $\frac{1-\zeta}{2}b \geq 1$ .

**Proof:** See Figure 4.3. The smallest product is produced by the solid cycles  $R_3 \rightarrow R_4 \rightarrow R_5 \rightarrow R_3$  and  $R_4 \rightarrow R_5 \rightarrow R_4$ . By requiring the label product of these loops to be more than 1, the desired result follows. ■

## 4.4 Accuracy

In this section, we prove that the mesh boundary is equal to the restriction of a  $\partial\Omega$  sample  $\Delta$  to  $\partial\Omega$ . In the literature, it is proved that these tetrahedra approximate the surface correctly, in geometric and topological sense [9, 27, 35].



**Figure 4.4:** Proof of Lemma 4.8.

First, we show that  $\delta$  directly controls the density of the feature vertices. Let  $V$  be the set of vertices in the final mesh and  $\Delta$  be equal to  $V \cap \partial\Omega$ .

**Lemma 4.8** *Let  $\delta < \frac{1}{4}$ . Then  $\Delta$  is a  $\frac{5\delta}{1-4\delta}$ -sample of  $\partial\Omega$ .*

**Proof:** Recall that upon termination, there is no tetrahedron for which R1, R2, R3, R4, or R5 apply.

See Figure 4.4. Let  $p$  be an arbitrary point on  $\partial\Omega$ . Since  $\mathcal{D}(V)$  covers all the domain, point  $p$  has to lie on or inside the circumsphere of a pentatope  $\sigma$  (not shown). Hence,  $B_\sigma$  intersects  $\partial\Omega$ . Let point  $p'$  be the feature point closest to  $c(\sigma)$ . Note that  $|c(\sigma) - p| \geq |c(\sigma) - p'|$  and therefore  $p'$  lies on or inside  $\sigma$ 's circumsphere. We also know that  $\sigma$ 's circumradius has to be less than  $2\Delta_{\partial\Omega}(p')$ , since otherwise R2 would apply for  $t$ . Therefore, we have the following:

$$\begin{aligned}
 |p - p'| &< 2R_\sigma && \text{(because both } p \text{ and } p' \text{ lie on or inside } B_\sigma) \\
 &< 4\Delta_{\partial\Omega}(p') && \text{(because of R2)} \\
 &\leq 4\delta(|p - p'| + \text{lfs}_{\partial\Omega}(p)) && \text{(from Inequality (4.1)),}
 \end{aligned}$$

and by reordering the terms, we obtain that:

$$|p - p'| < \frac{4\delta}{1 - 4\delta} \text{lfs}_{\partial\Omega}(p), \text{ with } \delta < \frac{1}{4}. \quad (4.2)$$

Moreover, there must exist a feature vertex  $v$  in the triangulation closer than  $\Delta_{\partial\Omega}(p') = \delta \cdot \text{lfs}_{\partial\Omega}(p')$  to  $p'$ , since otherwise R1 would apply for  $\sigma$ . Hence,  $|v - p'| <$

$\delta \cdot \text{lfs}_{\partial\Omega}(p')$ , and using Inequality (4.1), we have that:

$$|v - p'| < \delta (|p - p'| + \text{lfs}_{\partial\Omega}(p)) \quad (4.3)$$

Applying the triangle inequality for  $\Delta pvp'$  yields the following:

$$\begin{aligned} |p - v| &\leq | -pp'| + |v - p'| \\ &< |p - p'| + \delta (|p - p'| + \text{lfs}_{\partial\Omega}(p)) \quad (\text{from Inequality (4.3)}) \\ &= |p - p'| (1 + \delta) + \delta \cdot \text{lfs}_{\partial\Omega}(p) \\ &< \frac{4\delta}{1-4\delta} \text{lfs}_{\partial\Omega}(p) (1 + \delta) + \delta \cdot \text{lfs}_{\partial\Omega}(p) \quad (\text{from Inequality (4.2)}) \\ &= \left( \frac{4\delta(1+\delta)}{1-4\delta} + \delta \right) \text{lfs}_{\partial\Omega}(p) \\ &= \frac{5\delta}{1-4\delta} \text{lfs}_{\partial\Omega}(p), \end{aligned}$$

and the proof is complete. ■

Let us denote with  $\omega_i$  one of the  $n$  connected components that  $\Omega$  consists of:  $\Omega = \bigcup_{i=1}^n \omega_i$ . The next two Lemmas prove a few useful properties for the mesh  $\mathcal{M}$  and its boundary  $\partial\mathcal{M}$ . Our goal is to show that  $\partial\mathcal{M}$  is always non-empty and does not have boundary (Lemma 4.10), a fact that will be used for proving the fidelity guarantees (Theorem 4.2).

**Lemma 4.9** *Let  $\delta \leq \frac{1}{4}$ . Then, for every  $\omega_i$  there is a pentatope  $\sigma \in \mathcal{D}(V)$ , such that  $c(\sigma)$  lies inside  $\omega_i$ .*

**Proof:** Let us consider a single connected component  $\omega_i$ . The same reasoning applies for any connected component of  $\Omega$ .

For the sake of contradiction, assume that there is no pentatope whose circumcenter lies inside  $\omega_i$ . Since the triangulation  $\mathcal{D}(V)$  covers all the domain, the circumballs of the pentatopes in  $\mathcal{D}(V)$  also cover the domain  $\omega_i$ . Therefore, there has to be a circumball  $B_\sigma$  ( $\sigma \in \mathcal{D}(V)$ ) which intersects a point  $m$  on the medial axis of  $\partial\omega_i$ , such that  $m$  lies inside  $\omega_i$ . By our assumption, the circumcenter  $c(\sigma)$  cannot lie inside  $\omega_i$ . Therefore,  $B_\sigma$  intersects  $\partial\omega_i$ . Also, recall that R2 cannot apply to any pentatope. Hence, we have the following:

$$\begin{aligned} 2 \cdot \delta \cdot \text{lfs}_{\partial\Omega}(\text{cfp}_{\partial\Omega}(c(\sigma))) &> R_\sigma && (\text{from R2}) \\ &\geq \frac{|\text{cfp}_{\partial\Omega}(c(\sigma)) - m|}{2} && (\text{since } m \text{ and } \text{cfp}_{\partial\Omega}(c(\sigma)) \text{ do not lie outside } B_\sigma) \\ &\geq \frac{\text{lfs}_{\partial\Omega}(\text{cfp}_{\partial\Omega}(c(\sigma)))}{2} && (\text{since } m \text{ is on the medial axis}) \\ \delta &> \frac{1}{4}. && \Rightarrow \end{aligned}$$



which raises a contradiction. ■

**Lemma 4.10** *Let  $\delta \leq \frac{1}{4}$ . Then,  $\partial\mathcal{M}$  is a non-empty set and does not have boundary.*

**Proof:** The fact that  $\partial\mathcal{M}$  is a non-empty set follows directly from Lemma 4.9: since  $\mathcal{M}$  cannot be empty, its boundary  $\partial\mathcal{M}$  cannot be empty too. For the other part, since  $\partial\mathcal{M}$  is the boundary of a set of tetrahedra, it cannot have a boundary. ■

The following Theorem proves the fidelity guarantees:

**Theorem 4.2** *The mesh boundary  $\partial\mathcal{M}$  is the restriction to  $\partial\Omega$  of  $\Delta = V \cap \partial\Omega$ .*

**Proof:** Let  $f$  be a tetrahedron  $\sigma_3$  in  $\partial\mathcal{M}$ . As such,  $\text{Vor}(\sigma_3)$  intersects  $\partial\Omega$ . Due to R5, the vertices of  $\sigma_3$  lie on  $\partial\Omega$ . Recall that the surface ball  $\mathcal{B}_{z,\sigma_3}$  does not contain vertices in its interior. Therefore,  $\mathcal{B}_{z,\sigma_3}$  is empty of vertices in  $V \cap \partial\Omega$  also. Without loss of generality, assume that the vertices in  $V$  are in general position. Since there is a ball that circumscribes  $\sigma_3$  and does not contain vertices of  $V \cap \partial\Omega$  in its interior,  $\sigma_3$  has to appear as a simplex in  $\mathcal{D}(V \cap \partial\Omega)$ . Since the center  $z$  of the surface ball lies on  $\partial\Omega$ , then the voronoi dual of  $\sigma_3$  intersects  $\partial\Omega$  in  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$ , as well. Hence,  $\partial\mathcal{M} \subseteq \mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$ .

For the other direction, we will prove that  $\partial\mathcal{M}$  cannot be a proper subset of  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$ , and therefore, equality between these 2 sets is forced. Toward this direction, we will prove that any proper non-empty subset of  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$  has boundary; this is enough, because we have proved in Lemma 4.10 that  $\partial\mathcal{M}$  is non-empty and does not have boundary.

$\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$  is the restriction of a sample of a closed manifold  $\partial\Omega$  and therefore it is a 3-manifold without boundary [9]. That means that any 2-simplex in  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$  is incident to exactly two 3-simplices of  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$ . Since any proper non-empty subset  $\mathcal{A}$  of  $\mathcal{D}_{|\partial\Omega}(\partial\Omega) \cap V$  has fewer 3-simplices,  $\mathcal{A}$  contains at least a 2-simplex  $\sigma_2$  incident to only one 3-simplex. But this implies that  $\sigma_2$  belongs to the boundary of  $\mathcal{A}$ , and the proof is complete. ■

## 4.5 Experimental Evaluation

The algorithm is implemented in C++. We employed the Bowyer-Watson kernel [30, 128] for point insertions. The removal of a point  $p$  is implemented by computing the *small* Delaunay triangulation of the vertices incident to  $p$  [55], such that the vertices inserted earlier in the triangulation are inserted into the small triangulation first. It can be shown [63] that these new created pentatopes can always be connected back to the original triangulation without introducing invalid elements. For the Euclidean Distance Transform, we made use of the related filter implemented in ITK [7] and described in [97]. Lastly, we borrowed CGAL’s [6] exact predicates for the accurate computation of the 4D in-sphere tests.

**Table 4.1:** Information about the images of the five patients used in this section. The spacing for all the images is  $(1.77, 1.77, 6, 1)mm^4$ .

Case	Pat1	Pat2	Pat3	Pat4	Pat5
#Voxels	$(100 \times 100 \times 44 \times 15)$	$(100 \times 100 \times 34 \times 15)$	$(100 \times 100 \times 26 \times 15)$	$(100 \times 100 \times 31 \times 15)$	$(100 \times 100 \times 29 \times 15)$

We ran our code on five (segmented) images obtained from the 4D Heart Database [103]. The first three represent the moving left ventricle of the patients, while the last two the ventricle together with the myocardium for 15 cardiac cycles. Information about these data is given in Table 4.1.

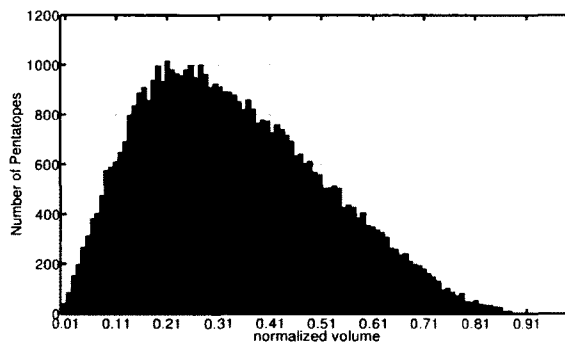
**Table 4.2:** Statistics of the output meshes generated for each patient.

	Pat1	Pat2	Pat3	Pat4	Pat5
# Pentatopes	49.479	43.673	8.883	63.016	56.528
# Boundary Tetrahedra	30.758	29.089	8.271	36.281	33.308
# Vertices	4.709	4.314	1.362	5.567	5.132
Shortest edge (mm)	3.45	3.87	3.90	3.5	4.63
Radius-edge ratio (maximum, average, deviation)	(1.93, 1.02, 0.17)	(1.78, 0.98, 0.15)	(1.54, 0.92, 0.10)	(2.20, 1.06, 0.18)	(1.87, 1.05, 0.18)
Normalized volume (minimum, average, deviation)	(0.01, 0.34, 0.18)	(0.01, 0.38, 0.18)	(0.02, 0.43, 0.17)	(0.01, 0.32, 0.17)	(0.01, 0.33, 0.17)

Recall that our algorithm needs the distance of any point on  $\partial\Omega$  from the medial axis. The robust computation of the medial axis is a very difficult problem (see [56, 70] for computing the exact medial axis, [49] for a review of image-based medial axis methods, and [11] for computing the medial axis given a set of surface points) and out of the scope of this thesis. Therefore, in the implementation, we assume that  $\text{lfS}_{\partial\Omega}(p)$  is uniform and equal to the unit, which implies that  $\Delta_{\partial\Omega}(p)$  becomes equal

to  $\delta$ . That is, in practice,  $\delta$  determines a uniform and (if small enough) dense sample of the surface. We experimentally verified that a  $\delta$  value equal to 5 (the length of five consecutive voxels along the fourth dimension) yielded manifold mesh boundaries with vertices lying precisely on the iso-surface in accordance with Theorem 4.2.

The quantity  $\tau_\sigma$  determines the aspect ratio of pentatope  $\sigma$  [90], but it is not normalized, and therefore, it is hard to draw comparative conclusions. For this reason, for a pentatope  $\sigma$  of the final mesh, we report its *normalized volume*  $\hat{\tau}_\sigma$  defined as the ratio of its volume over the volume of a regular pentatope with circumradius equal to the circumradius of  $\sigma$  (or alternatively  $\hat{\tau}_\sigma = \frac{384 \text{Vol}_\sigma}{24R_\sigma^4 \sqrt{5}}$ ). Clearly,  $\hat{\tau}_\sigma \in [0, 1]$ , where a value of 0 implies a degenerate pentatope, while 1 implies a perfect quality.



**Figure 4.5:** Normalized volume histogram of the output mesh obtained for the input image Pat1.

Table 4.2 shows quantitative data for the mesh generated on each image. We set the radius of the picking regions equal to  $\zeta = \frac{1}{2}$ . Theorem 4.1 suggests that  $\bar{\rho}$  be at least 16 and  $b$  at least 4. We experimentally observed that by selecting 4 to 10 random points within the picking regions (both the 4- and the 3-topological balls), no small element  $\sigma$  was created with  $\hat{\tau}_\sigma$  less than 0.01. Despite the fact a value of 0.01 is rather small, it is three orders of magnitude larger than the minimum normalized volume reported in the case where no picking regions are employed at all. Also, notice that the average normalized volume is much higher than the minimum value. This fact together with the observed small standard deviation implies that most pentatopes have normalized volume away from the minimum value and very close to the average. Figure 4.5 shows the histogram of the normalized volumes for the first experiment of Table 4.2, that is, when the input image Pat1 was used. Similar histograms are

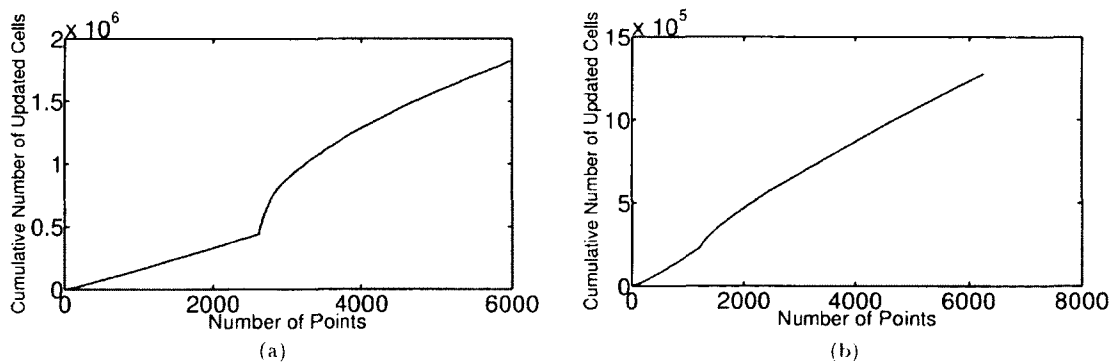
observed for all the other inputs as well.

## 4.6 Real-Time 4D Meshing

During the development of the 4D Delaunay refinement code, we realized that its performance behaves very differently than the performance of the 3D code we developed and described in Chapter 2 and Chapter 3. This is due to mainly two reasons: (a) the storage requirements and computations involved in a point insertion or removal are higher because of the increased dimensionality, and (b) the 4D CGAL predicates we employed to enforce robustness are not as well optimized as their 3D counterparts. Indeed, the achieved rate of meshing a 4D hyper-sphere with 40,000 elements is 145 pentatopes per second, while the achieved rate of meshing the hypersphere’s equator with the same number of elements is 107,037 tetrahedra per second.

In this Section, we improve the speed of our 4D code by optimizing its complexity and by parallelizing the whole process. Since point removals account for approximately less than 1% of the total number of operations in all the cases we investigated, we focus on 4D Delaunay point insertions.

### 4.6.1 Complexity



**Figure 4.6:** The complexity of the 4D code (a) before, and (b) after the Rule reordering.

Ignoring the time involved for locating the first element in a point’s cavity, the optimal complexity of a Delaunay insertion is constant. Therefore, inserting  $n$  points costs  $\Omega(n)$  time. Although the 3D code reaches the optimal complexity in all the case studies we experimented on, its 4D counterpart behaves very differently. Indeed, Figure 4.6a shows the number of deleted and created elements involved so far with respect to the number of inserted points on the 4D hyper-sphere. If the complexity was optimal, then the curve should look like a straight line. We observe, however, that the complexity is far from ideal after the insertion of approximately 2,500 points. We obtained similar results when we ran our code on other inputs, such as hyper-torus and the five 4D hearts of Section 4.5.

Nevertheless, it can be proved [99] that it is possible to reach the optimal complexity if, at any given moment of refinement, the radius-edge ratio is bounded from above. In fact, this technique has already been applied successfully in the literature [77]. Therefore, we reordered the Rules of our algorithm (see Section 4.2), such that rule R3 has the highest priority among all the rest of the Rules. In this way, the mesh is always of bounded radius-edge ratio and as such, the expected complexity should be close to the optimal. Indeed, Figure 4.6b shows that the complexity curve behaves linearly. This improvement boosted the performance of the 4D code by 27% on the hyper-sphere mentioned above, bringing the rate of 145 pentatopes per second up to 184 for the 40,000 element mesh generated.

#### 4.6.2 Parallelization

In this Subsection, we parallelize the 4D algorithm to take advantage of the multi-core and many-core platforms already available in the market. To our knowledge, this is the first attempt to parallelize the mesh generation and refinement of 4D space-time domains.

We employed a tightly-coupled approach similar to the concept of the 3D PI2M. That is, before a thread applies any change, it has to lock all the associated vertices. An attempt to lock an already acquired vertex results in a rollback. The avoidance of livelocks is achieved via the Local-CM presented in detail in Section 3.3. since it

was shown to be the most effective way to eliminate livelocks, even in the presence of very little parallelism.

We deactivated the picking region technique described in detail in the previous Sections of this Chapter, because we wanted to perform a 1-to-1 comparison with the 3D code and investigate which parallelization techniques that were applied successfully in 3D benefit the parallelization of the 4D problem as well. Keeping the picking regions would imply more than one rounds per insertion causing a considerable increase in the number of rollbacks, a fact not associated to the nature of the 4D problem, but to the technique of eliminating slivers.

**Table 4.3:** The performance of the parallel 4D method (a) without, and (b) with fine grained parallelism.

(a)			
Threads	1	6	12
#Elements	39,696	40,598	39,870
Time (secs)	207.0	131.5	134.0
Elements per second	191.7	308.8	297.6
Speedup	1.00	1.61	1.55
Contention seconds per thread	0.0	92.0	111.2
Balance seconds per thread	0.0	1.3	2.7
Rollback seconds per thread	0.0	1.3	9.4
Total overhead seconds per thread	0.0	94.7	123.2

(b)			
Threads	1	6	12
#Elements	39,696	39,632	39,612
Time (secs)	204.9	74.1	72.8
Elements per second	193.8	534.5	544.0
Speedup	1.00	2.76	2.81
Contention seconds per thread	0.0	34.8	47.3
Balance seconds per thread	0.0	1.5	6.6
Rollback seconds per thread	0.0	1.5	0.1
Total overhead seconds per thread	0.0	37.8	54.0

Table 4.3a illustrates the strong scaling performance of the 4D parallel implementation on the Pat5 input 4D heart. It also shows the average total overhead seconds per thread (last column) and the exact source of the overhead, i.e., contention, balance, and rollback overhead, as defined in Section 3.3.5.

Although the same parallelization techniques scaled the 3D counterpart for a core count higher than 128, we observe that intensive overhead hampers scalability even on 12 cores in 4D domains. For example, 91% of the total execution time on 12 cores was spent waiting on contention lists, balance lists, and rollbacks. Interestingly, the overhead of the 3D counterpart on a slice of the same 4D input was only 69% on 12 cores, when it generated a mesh of approximately the same size. This different behavior could be attributed to the fact that now the size of the cavity is much larger in 4D than it is in 3D. Indeed, we computed that the average size of the 4D cavity (4D Pat5 heart) is about 72.9 pentatopes, while the average size of the 3D cavity (slice of Pat5) is 18.0.

Nevertheless, the fact that most of the time is spent idling on contention and balance lists gives us the opportunity to perform cavity expansions in parallel. When a thread is working on inserting a point, it invites idling threads to perform the operation in parallel. This parallelization scheme is called *fine grained* parallelization and was successfully employed in the past by our group [13].

Table 4.3b shows the fine grained performance of our implementation. Observe that the overhead seconds were greatly reduced. For example, on 12 cores, the overhead seconds were reduced by 2.2X, simply because threads help active threads to do useful work and therefore, they wait on the contention/balance lists much less. As an immediate result, the fine grained implementation is 1.7X and 1.8X faster on 6 and 12 cores respectively.

## Chapter 5

# Conclusions and Future Work

In this work, we presented a 3D Delaunay refinement image-to-mesh conversion algorithm that operates directly on segmented images. It is able to create an appropriate sample set on the object's surface, and to mesh the volume and the surface at the same time. This flexibility (besides the fact that it solves three distinct problems, that is, sampling, surface recovery, and volume meshing) results in a generally lower number of vertices than in the case where the surface is meshed without considering the rules of quality. For instance, the insertion of points that improve the quality might help the density requirement at the same time.

Apart from the fidelity guarantees we give (namely, that the mesh boundary is a good geometric and topological approximation of the object's surface), our algorithm provably achieves very low radius-edge ratio without sacrificing fidelity. The planar angles of the boundary facets are also guaranteed to be larger than  $30^\circ$ . Moreover, by slightly relaxing the quality guarantees, our algorithm provably exhibits good grading.

Experimental evaluation on various images shows that the final meshes are free of slivers and exhibit both volume and (in most cases) surface grading, a fact that greatly reduces the size of the mesh making the subsequent FEM analysis [18, 19, 47] faster. Lastly, demonstration of the use of custom size functions shows that our algorithm allows for additional flexibility to meet user-defined mesh density.

We also presented PI2M: the first parallel Image-to-Mesh (PI2M) Conversion



Isosurface-based algorithm and its implementation. Starting directly from a multi-label segmented 3D image, it is able to recover and mesh both the isosurface  $\partial\Omega$  with geometric and topological guarantees (see Theorem 3.1) and the underlying volume  $\Omega$  with quality elements.

This work is different from parallel Triangulators [20, 23, 24, 63], since parallel mesh generation and refinement focuses on the quality of elements (tetrahedra and facets) and the conformal representation of the tissues' boundaries/isosurfaces by computing on demand the appropriate points for insertion or deletion. Parallel Triangulators tessellate only the convex hull of a set of points.

Our tightly-coupled method greatly reduces the number of rollbacks and scales up to a much higher core count, compared to the tightly-coupled method our group developed in the past [105]. The data decomposition method [43] does not support Delaunay removals, a technique that it is shown to be effective in the sequential mesh generation literature [62, 64]. The extension of partially-coupled [39] and decoupled [92] methods to 3D is a very difficult task, since Delaunay-admissible 3D medial decomposition is an unsolved problem. On the contrary, our method does not rely on any domain decomposition, and could be extended to arbitrary dimensions as well. Indeed, we plan to extend PI2M to 4 dimensions and generate space-time elements (needed for spatio-temporal simulations [21, 111]) in parallel, thus, exploiting parallelism in the fourth dimension. As future work, we also leave the mesh boundary smoothing required for CFD simulations, such as respiratory airway modeling [57, 87, 88].

Our code is highly optimized through carefully designed contention managers, and load balancers which take advantage of NUMA architectures. Our Global Contention Manager (Global-CM) and Local Contention Manager (Local-CM) provably eliminate deadlocks and livelocks. They achieve a speedup even on 256 cores, when other traditional contention managers, found in the mesh generation literature, fail to terminate. Local-CM also reduced the number of overhead cycles by a factor of 2 compared to the Global-CM on 256 cores, improving energy-efficiency by avoiding energy waste because of rollbacks. Lastly, our Hierarchical Work Stealing load balancer (HWS) sped up the execution by a factor of 1.45 on 176 cores, as a result of a 22.8% remote

accesses reduction.

All in all, PI2M achieves a strong scaling efficiency of more than 82% on 64 cores. It also achieves a weak scaling efficiency of more than 82% on up to 144 cores. We are not aware of any 3D parallel Delaunay mesh refinement algorithm achieving such a performance.

It is worth noting that PI2M exhibits excellent single-threaded performance. Despite the extra overhead associated with synchronization, contention management, and load balancing, PI2M generates meshes 40% faster than CGAL and with similar quality. Moreover, PI2M achieves better quality than TetGen, and it is also faster than TetGen for large mesh sizes.

Recall that in our method, threads spend time idling on the contention and load balancing lists. And this is necessary in our algorithm for correctness and performance efficiency. This fact offers great opportunities to control the power consumption, or alternatively, to maximize the  $\frac{\text{Elements}}{\text{second} \times \text{Watt}}$  ratio. Since idling is not the time critical component in our algorithm, the CPU frequency could be decreased during such an idling. Nevertheless, the appropriate frequency drop, the amount of idling, and performance is a trade-off, and its investigation is left as future work.

As already explained, for core counts higher than 144, weak scaling performance deteriorates because communication traffic (per switch) is more intense and passes through a larger number of hops. In the future, we plan to increase scalability by employing a hierarchically layered (distributed and shared memory) implementation design [46] and combine this tightly-coupled method with the decoupled and partially coupled methods we developed in the past, exploring in this way different levels of concurrency.

Lastly, in this dissertation, we presented a space-time meshing method for  $(3D \cdot t)$  image data. The method is able to provably clean up slivers and recover the hyper-surfaces faithfully. Experiments on five 4D cardiac images show that the resulting meshes consist of elements of bounded aspect ratio.

Efficient Discontinuous Galerkin formulations require that not only the hyper-surface should be recovered but also the evolving 3D object at certain time steps [48]. This is a more challenging task considering the non-manifold nature of the underlying

space-time domain and it is left as future work.

Because of the increased memory space needed for high dimensional meshing, our 4D algorithm is rather slow: it is approximately 700 times slower than our three dimensional Delaunay mesher, as described in Section 4.6. Nevertheless, the fine grained parallelization for the 4D code did yield a 2.81 speedup on 12 cores. We argue that the main bottleneck for its scalability is the excessive amount of contention, a fact that we did not observe in the 3D counterpart. We attribute this difference in behavior between the 3D and 4D implementation to the fact that the cavity size increases in higher dimensions and therefore, tightly-coupled techniques need to lock many more vertices. In the future, we plan to investigate other parallelization techniques, such as data decomposition [43] and domain decomposition [39, 92], since they are expected to alleviate the increased synchronization overhead observed in high dimensional meshing. In the future, we also plan to theoretically characterize the complexity of our parallel methods described in Chapter 3 and Chapter 4, determining in this way their scalability on machines of different architecture [51].

## Appendix A

# Installing and Using the Software

In this Appendix, we provide brief instructions regarding the usage of the pieces of software presented in Chapter 3 and Chapter 4, respectively. In particular, we describe how to use (a) the 3D parallel Image to Mesh Conversion code (PI2M) and (b) its 4D counterpart (PI2M4). The source code is located at `code/ParallelMeshGeneration3D/` and `code/ParallelMeshGeneration4D/`, respectively.

The developed software depends on various other libraries which need to be installed prior to compiling it. Specifically, our software depends on CMake [2], ITK [7], VTK [3], CGAL [6], PAPI [5], and BoostC++ [1]. Both executables are invoked as follows:

```
./Parallel --threads value --input name --delta value [--output name] [--plc name]
```

where:

- threads: the number of threads.
- input: an ITK compatible 3D/4D segmented image.
- delta: the  $\delta$  parameter that controls the density.
- output: the optional output mesh name where the mesh will be stored in the disk. Also, mesh statistics about the quality of the mesh are reported. In 4D, the sequence of slices are

generated (name0, name1...) as extracted from the mesh. If not given, then no output mesh will be created.

- plc: the optional output PLC name. If not given, then no output PLC will be created. 3D only.

Next, we list several important macros that affect the performance and the functionality of both 3D and 4D code, depending on whether or not they are on. These macros can be found in either Config.h or Parallel\_Mesh\_Generator.h. Table A.1a and Table A.1b elaborate on these macros and their effect.

Lastly, under code/ParallelMeshGeneration4D/, we have implemented two PI2M4 versions: v12 and v13. The difference is that v13 re-arranges the priority of the Rules for complexity improvement, as described in Section 4.6.1. These versions have been kept separate for maintainability and readability reasons.

**Table A.1:** The list and descriptions of influential macros.

(a) Macros in Config.h that can be activated or deactivated by the user.

macro	description	comment
ASSERT	sanity checks for the sequential mode	one thread only
PARALLEL_ASSERT	sanity checks for the parallel mode	
MEMORY_MANAGER	if on, a pool of memory is maintained by each thread	
COMPUTE_MESH_ELEMENTS_ON_THE_FLY	if on, the desired mesh can be extracted from the triangulation in constant time. Otherwise, all the elements need to be traversed and classified during a post-processing step	3D only
REMOVE_ON_THE_FLY	if on, invalid elements are removed from their corresponding PEL right away. Otherwise, they are just marked as invalid and they are removed from their PEL when the the responsible thread examines it for splitting in a lazy manner	

(b) Macros in Parallel\_Mesh\_Generation.h that can be activated or deactivated by the user.

macro	description	comment
NO_GOOD_ANGLES	if on, no dihedral angle improvement is performed	3D only
NO_REJECTION_STRATEGIES	if on, the isosurface is not protected	3D only
SLIVER_REMOVAL	picking regions are activated	4D only with 1 thread
FINE_GRAINED	fine grained parallelism is activated	4D only
CONTENTION_LOCAL_ON	Local-CM is on	
CONTENTION_SIGNAL_ON	Global-CM is on	3D only
CONTENTION_SLEEP_ON	Random-CM is on	3D only
INFINITE_THREAD	if on, there is a dedicated thread for inserting/removing points on the box. This decreases the number of rollbacks on the convex hull	
HLB	the Hierarchical work stealing Load Balancer	
REPORT_COUNTERS	several statistics are obtained from each thread during the execution and reported at the end. This introduces zero synchronization overhead, but it does introduce some computation overhead	
PAPI_ON	PAPI counters are activated: TLB and LLC misses are counted per thread, as well as the number of resource stall cycles and remote accesses	
MARK_BOUNDARY_TRIANGLES	if on, then the resulted VTK mesh file marks the faces of each tetrahedron as boundary triangles or not. Useful when applying boundary conditions. It requires that the optional mesh name is given	3D only
NUMA	if on, the hop-wise distance of any node pair is computed	

# Bibliography

- [1] Boost C++ libraries. <http://www.boost.org/>.
- [2] CMAKE. <http://www.cmake.org>.
- [3] VTK, Visualization Toolkit. <http://www.vtk.org>.
- [4] SGI UV 100/1000 system specifications. <http://www.sgi.com/products/servers/uv/specs.html>, 2012. available online.
- [5] PAPI, Performance Application Programming Interface. <http://icl.cs.utk.edu/papi/>, 2012.
- [6] CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org>, v4.0.
- [7] ITK, Insight Segmentation and Registration Toolkit. <http://www.itk.org>, v4.1.0.
- [8] PIERRE ALLIEZ, DAVID COHEN-STEINER, MARIETTE YVINEC, AND MATHIEU DESBRUN. Variational tetrahedral meshing. *ACM Trans. Graph.*, 24:617–625, July 2005.
- [9] NINA AMENTA AND MARSHALL BERN. Surface reconstruction by Voronoi filtering. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 39–48. New York, NY, USA, 1998. ACM.
- [10] NINA AMENTA, SUNGHEE CHOI, TAMAL K. DEY, AND N. LEEKHA. A Simple Algorithm for Homeomorphic Surface Reconstruction. *International Journal of Computational Geometry and Applications*, 12(1-2):125–141, 2002.
- [11] NINA AMENTA, SUNGHEE CHOI, AND RAVI KRISHNA KOLLURI. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266. New York, NY, USA, 2001. ACM.
- [12] KATRIN AMUNTS, CLAUDE LEPAGE, LOUIS BORGEAT, HARTMUT MOHLBERG, TIMO DICKSCHEID, MARC-ÉTIENNE ROUSSEAU, SEBASTIAN BLUDAU, PIERRE-LOUIS BAZIN, LINDSAY B. LEWIS, ANA-MARIA OROS-PEUSQUENS, NADIM J. SHAH, THOMAS LIPPERT, KARL ZILLES, AND ALAN C. EVANS. BigBrain: An Ultrahigh-Resolution 3D Human Brain Model. *Science*. 340(6139):1472–1475, June 2013.
- [13] CHRISTOS ANTONOPOULOS, FILIP BLAGOJEVIC, ANDREY CHERNIKOV, NIKOS CHRISOCHOIDES, AND DIMITRIS NIKOLOPOULOS. A multigrain Delaunay mesh generation method for multicore smt-based architectures. *Journal on Parallel and Distributed Computing*. 69:589–600. 2009.
- [14] CHRISTOS ANTONOPOULOS, XIAONING DING, ANDREY CHERNIKOV, FILIP BLAGOJEVIC, DIMITRIS NIKOLOPOULOS, AND NIKOS CHRISOCHOIDES. Multigrain parallel Delaunay mesh generation: Challenges and opportunities for multithreaded architectures. In *ACM International Conference on Supercomputing*, number 19, pages 367–376. ACM. 2005.

- [15] NECULAI ARCHIP, OLIVIER CLATZ, ANDRIY FEDOROV, ANDRIY KOT, STEPHEN WHALEN, DAN KACHER, NIKOS CHRISOCHOIDES, FERENC JOLESZ, ALEXANDRA GOLBY, PETER BLACK, AND SIMON K. WARFIELD. Non-rigid alignment of preoperative MRI, fMRI, DT-MRI, with intra-operative MRI for enhanced visualization and navigation in image-guided neurosurgery. *Neuroimage*, 35(2):609–624, 2007.
- [16] KRSTE ASANOVIC, RASTISLAV BODIK, JAMES DEMMEL, TONY KEAVENY, KURT KEUTZER, JOHN KUBIATOWICZ, NELSON MORGAN, DAVID PATTERSON, KOUSHIK SEN, JOHN WAWRZYNEK, DAVID WESSEL, AND KATHERINE YELICK. A view of the parallel computing landscape. *Commun. ACM*, 52:56–67, October 2009.
- [17] DOMINIQUE ATTALI, HERBERT EDELSBRUNNER, AND YURIY MILEYKO. Weak witnesses for delaunay triangulations of submanifolds. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, SPM '07, pages 143–150, New York, NY, USA, 2007. ACM.
- [18] M. AUDETTE, M. MIGA, J. NEMES, K. CHINZEI, AND T. PETERS. A Review of Biomechanical Modeling of the Brain for Intracranial Displacement Estimation and Medical Simulation. *Biomechanical Systems, General Anatomy*, pages 83–112, 2007.
- [19] M. A. AUDETTE, H. DELINGETTE, A. FUCHS, O. BURGERT, AND K. CHINZEI. A topologically faithful, tissue-guided, spatially varying meshing strategy for computing patient-specific head models for endoscopic pituitary surgery simulation. *Computer Aided Surgery*, 12(1):43–52, 2007.
- [20] VICENTE H.F. BATISTA, DAVID L. MILLMAN, SYLVAIN PION, AND JOHANNES SINGLER. Parallel geometric algorithms for multi-core computers. *Computational Geometry*, 43(8):663–677, 2010.
- [21] MAREK BEHR. Simplex space-time meshes in finite element simulations. *International Journal for Numerical Methods in Fluids*, 57:1421–1434, 2008.
- [22] M BEN-ARI. *Principles of concurrent programming. Chapter 3, pages 30-43*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [23] DANIEL K. BLANDFORD, GUY E. BLELLOCH, AND CLEMENS KADOW. Engineering a compact parallel Delaunay algorithm in 3D. In *Proceedings of the 22<sup>nd</sup> Symposium on Computational Geometry*, SCG '06, pages 292–300, New York, NY, USA, 2006. ACM.
- [24] GUY E. BLELLOCH, GARY L. MILLER, JONATHAN C. HARDWICK, AND DAFNA TALMOR. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 24(3):243–269, 1999.
- [25] HARRY BLUM. A Transformation for Extracting New Descriptors of Shape. In *Models for the Perception of Speech and Visual Form*, Weiant Wathen-Dunn, editor, pages 362–380. MIT Press, Cambridge, 1967.
- [26] ROBERT D. BLUMOFF, CHRISTOPHER F. JOERG, BRADLEY C. KUSZMAUL, CHARLES E. LEISERSON, KEITH H. RANDALL, AND YULI ZHOU. Cilk: an efficient multithreaded runtime system. In *Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '95, pages 207–216, New York, NY, USA, 1995. ACM.
- [27] JEAN-DANIEL BOISSONNAT, LEONIDAS J. GUIBAS, AND STEVE Y. OUDOT. Manifold reconstruction in arbitrary dimensions using witness complexes. *Discrete Comput. Geom.*, 42:37–70, May 2009.



- [28] JEAN-DANIEL BOISSONNAT AND STEVE OUDOT. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005.
- [29] DOBRINA BOLTCHEVA, MARIETTE YVINEC, AND JEAN-DANIEL BOISSONNAT. Mesh Generation from 3D Multi-material Images. In *Medical Image Computing and Computer-Assisted Intervention*, pages 283–290. Springer, September 2009.
- [30] ADRIAN BOWYER. Computing Dirichlet tessellations. *Computer Journal*, 24:162–166, 1981.
- [31] CARSTEN BURSTEDDE, OMAR GHATTAS, MICHAEL GURNIS, TOBIN ISAAC, GEORG STADLER, TIM WARBURTON, AND LUCAS WILCOX. Extreme-scale amr. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10*, pages 1–12. IEEE Computer Society, 2010.
- [32] FRÉDÉRIC CAZALS AND JOACHIM GIESEN. Delaunay triangulation based surface reconstruction: Ideas and algorithms. In *Effective Computational Geometry for Curves and surfaces*, pages 231–273. Springer, 2006.
- [33] F. CHAZAL AND A. LIEUTIER. Stability and homotopy of a subset of the medial axis. In *Proceedings of the ninth ACM symposium on Solid modeling and applications, SM '04*, pages 243–248, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.
- [34] SIU-WING CHENG, TAMAL K. DEY, HERBERT EDELSBRUNNER, MICHAEL A. FACELLO, AND SHANG-HUA TENG. Sliver exudation. *Journal of the ACM*, 47(5):883–904, 2000.
- [35] SIU-WING CHENG, TAMAL K. DEY, AND EDGAR A. RAMOS. Manifold reconstruction from point samples. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05*, pages 1018–1027, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
- [36] SIU-WING CHENG, TAMAL K. DEY, AND EDGAR A. RAMOS. Delaunay refinement for piecewise smooth complexes. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1096–1105. ACM Press, 2007.
- [37] NUTTAPONG CHENTANEZ, RON ALTEROVITZ, DANIEL RITCHIE, LITA CHO, KRIS K. HAUSER, KEN GOLDBERG, JONATHAN R. SHEWCHUK, AND JAMES F. O'BRIEN. Interactive simulation of surgical needle insertion and steering. In *Proceedings of ACM SIGGRAPH 2009*, pages 88:1–10, Aug 2009.
- [38] ANDREY CHERNIKOV AND NIKOS CHRISOCHOIDES. Three-Dimensional Semi-Generalized Point Placement Method for Delaunay Mesh Refinement. In *Proceedings of the 16<sup>th</sup> International Meshing Roundtable*, pages 25–44, Seattle, WA, October 2007. Elsevier.
- [39] ANDREY CHERNIKOV AND NIKOS CHRISOCHOIDES. Algorithm 872: Parallel 2D constrained Delaunay mesh generation. *ACM Transactions on Mathematical Software*. 34:6 25, January 2008.
- [40] ANDREY CHERNIKOV AND NIKOS CHRISOCHOIDES. Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity. *SIAM Journal on Scientific Computing*. 33:3491–3508, 2011.
- [41] ANDREY CHERNIKOV AND NIKOS CHRISOCHOIDES. Tetrahedral image-to-mesh conversion for biomedical applications. In *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*. pages 125–134. Chicago, IL, August 2011.

- [42] ANDREY CHERNIKOV AND NIKOS CHRISOCHOIDES. Generalized insertion region guides for Delaunay mesh refinement. *SIAM Journal on Scientific Computing*, 34(3):A1333–A1350, 2012.
- [43] ANDREY N. CHERNIKOV AND NIKOS P. CHRISOCHOIDES. Three-dimensional Delaunay refinement for multi-core processors. In *Proceedings of the 22nd annual international Conference on Supercomputing*, ICS '08, pages 214–224, New York, NY, USA, 2008. ACM.
- [44] L. PAUL CHEW. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the 9<sup>th</sup> ACM Symposium on Computational Geometry*, pages 274–280, San Diego, CA, 1993.
- [45] L. PAUL CHEW. Guaranteed-quality Delaunay meshing in 3D. In *Proceedings of the 13<sup>th</sup> ACM Symposium on Computational Geometry*, pages 391–393, Nice, France, 1997.
- [46] NIKOS CHRISOCHOIDES, ANDREY CHERNIKOV, ANDRIY FEDOROV, ANDRIY KOT, LEONIDAS LINARDAKIS, AND PANAGIOTIS FOTEINOS. Towards exascale parallel Delaunay mesh generation. In *International Meshing Roundtable*, number 18, pages 319–336, Salt Lake City, Utah, October 2009. Springer Berlin Heidelberg.
- [47] OLIVIER CLATZ, HERVÉ DELINGETTE, ION-FLORIN TALOS, ALEXANDRA J. GOLBY, RON KIKINIS, FERENC JOLESZ, NICHOLAS AYACHE, AND SIMON WARFIELD. Robust non-rigid registration to capture brain shift from intra-operative MRI. *IEEE Transactions on Medical Imaging*, 24(11):1417–1427, November 2005.
- [48] BERNARDO COCKBURN, GEORGE E. KARNIADAKIS, AND CHI-WANG SHU. Discontinuous galerkin methods: theory, computation and applications. *Lecture notes in Computational Science and Engineering*, 11, 2000.
- [49] DAVID COEURJOLLY AND ANNICK MONTANVERT. Optimal separable algorithms to compute the reverse euclidean distance transformation and discrete medial axis in arbitrary dimension. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29:437–448, March 2007.
- [50] H. L. DE COUGNY AND M. S. SHEPHARD. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46(7):1101–1125, 1999.
- [51] DAVID CULLER, RICHARD KARP, DAVID PATTERSON, ABHIJIT SAHAY, KLAUS ERIK SCHAUER, EUNICE SANTOS, RAMESH SUBRAMONIAN, AND THORSTEN VON EICKEN. Logp: Towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.
- [52] PER ERIC DANIELSSON. Euclidean Distance Mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.
- [53] WN DAWES, SA HARVEY, S. FELLOWS, N. ECCLES, D. JAEGGI, AND WP KELLAR. *A practical demonstration of scalable. parallel mesh generation*. American Institute of Aeronautics and Astronautics, 1801 Alexander Bell Dr., Suite 500 Reston VA 20191-4344 USA,, 2009.
- [54] OLIVIER DEVILLERS AND SYLVAIN PION. Efficient exact geometric predicates for delaunay triangulations. In *Proc. 5th Workshop Algorithm Eng. Exper.*, pages 37–44. SIAM, 2003.
- [55] OLIVIER DEVILLERS AND MONIQUE TEILLAUD. Perturbations and vertex removal in a 3D Delaunay triangulation. In *Proceedings of the 14<sup>th</sup> ACM-SIAM Symposium on Discrete algorithms*, SODA '03, pages 313–319. SIAM, 2003.
- [56] TAMAL K. DEY AND WULUE ZHAO. Approximate medial axis as a voronoi subcomplex. *Computer-Aided Design*, 36(2):195–202, 2004.

- [57] VOLODYMYR DYEDOV, DANIEL R. EINSTEIN, XIANGMIN JIAO, ANDREW P. KUPRAT, JAMES P. CARSON, AND FACUNDO DEL PIN. Variational generation of prismatic boundary-layer meshes for biomedical computing. *International Journal for Numerical Methods in Engineering*, 79(8):907–945, 2009.
- [58] HERBERT EDELSBRUNNER AND DAMRONG GUOY. An experimental study of sliver exudation. *Engineering with Computers*, 18:229–240, 2002.
- [59] HERBERT EDELSBRUNNER AND NIMISH R. SHAH. Triangulating topological spaces. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 285–292, New York, NY, USA, 1994. ACM.
- [60] JEFF ERICKSON, DAMRONG GUOY, JOHN M. SULLIVAN, AND ALPER ÜNGÖR. Building spacetime meshes over arbitrary spatial domains. *Eng. with Comput.*, 20(4):342–353, August 2005.
- [61] ANDRIY FEDOROV AND NIKOS CHRISOCHOIDES. Tetrahedral Mesh Generation for Non-rigid Registration of Brain MRI: Analysis of the Requirements and Evaluation of Solutions. In *International Meshing Roundtable*, pages 55–72. Springer Verlag, October 2008.
- [62] PANAGIOTIS FOTEINOS, ANDREY CHERNIKOV, AND NIKOS CHRISOCHOIDES. Guaranteed Quality Tetrahedral Delaunay Meshing for Medical Images. In *Proceedings of the 7<sup>th</sup> International Symposium on Voronoi Diagrams in Science and Engineering*, pages 215–223. IEEE Computer Society, June 2010.
- [63] PANAGIOTIS FOTEINOS AND NIKOS CHRISOCHOIDES. Dynamic parallel 3D Delaunay triangulation. In *International Meshing Roundtable*, pages 3–20, Paris, France, October 2012. Springer Berlin Heidelberg.
- [64] PANAGIOTIS FOTEINOS AND NIKOS CHRISOCHOIDES. High-quality multi-tissue mesh generation for finite element analysis. In *Image-Based Geometric Modeling and Mesh Generation*, Yongjie (Jessica) Zhang, editor, volume 3 of *Lecture Notes in Computational Vision and Biomechanics*, pages 159–169. Springer Netherlands, 2013.
- [65] PANAGIOTIS FOTEINOS, DAMING FENG, ANDREY CHERNIKOV, AND NIKOS CHRISOCHOIDES. Multi-layered unstructured mesh generation. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ICS '13, pages 471–472, New York, NY, USA, 2013. ACM.
- [66] PANAGIOTIS FOTEINOS, YIXUN LIU, ANDREY CHERNIKOV, AND NIKOS CHRISOCHOIDES. An Evaluation of Tetrahedral Mesh Generation for Non-Rigid Registration of Brain MRI. In *Computational Biomechanics for Medicine V, 13<sup>th</sup> International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI) Workshop*, pages 126–137. Springer, September 2010.
- [67] LORI A. FREITAG AND CARL OLLIVIER-GOOCH. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.
- [68] JÉRÔME GALTIER AND PAUL-LOUIS GEORGE. Prepartitioning as a way to mesh subdomains in parallel. In *Special Symposium on Trends in Unstructured Mesh Generation*, pages 107–122. ASME/ASCE/SES, 1997.
- [69] PAUL-LOUIS GEORGE AND HOUMAN BOROUCHEKI. *Delaunay triangulation and meshing. Application to finite elements*. HERMES, 1998.

- [70] PETER GIBLIN AND BENJAMIN B. KIMIA. A formal classification of 3D medial axis points and their local geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:238–251, January 2004.
- [71] ORCUN GOKSEL AND SEPTIMIU E. SALCUDEAN. High-quality model generation for finite element simulation of tissue deformation. In *12<sup>th</sup> International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, MICCAI '09, pages 248–256, Berlin, Heidelberg, 2009. Springer-Verlag.
- [72] JOHN L. GUSTAFSON. Reevaluating Amdahl's law. *Communications of the ACM*, 31:532–533, 1988.
- [73] ULRICH HARTMANN AND FRITHJOF KRUGGEL. A Fast Algorithm for Generating Large Tetrahedral 3D Finite Element Meshes from Magnetic Resonance Tomograms. In *Proceedings of the IEEE Workshop on Biomedical Image Analysis, WBIA*, pages 184–192, Washington, DC, USA, 1998. IEEE Computer Society.
- [74] MAURICE HERLIHY, VICTOR LUCHANGCO, AND MARK MOIR. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems, ICDCS '03*, pages 522–. IEEE Computer Society, 2003.
- [75] MAURICE HERLIHY AND J. ELIOT B. MOSS. Transactional memory: architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993.
- [76] PING HU, HUI CHEN, WEN WU, AND PHENG-ANN HENG. Multi-tissue tetrahedral mesh generation from medical images. In *International Conference on Bioinformatics and Biomedical Engineering (iCBBE)*, pages 1–4. IEEE, June 2010.
- [77] BENOÎT HUDSON, GARY MILLER, AND TODD PHILLIPS. Sparse voronoi refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356. Springer Berlin Heidelberg, 2006.
- [78] YASUSHI ITO, ALAN SHIH, ANIL ERUKALA, BHARAT SONI, ANDREY CHERNIKOV, NIKOS CHRISOCHOIDES, AND KAZUHIRO NAKAHASHI. Parallel mesh generation using an advancing front method. *Mathematics and Computers in Simulation*, 75:200–209, September 2007.
- [79] E. IVANOV, H. ANDRĂ, AND A. KUDRYAVTSEV. Domain decomposition approach for automatic parallel generation of tetrahedral grids. Technical Report 87. Fraunhofer (ITWM), 2006.
- [80] M. JAKAB AND R. KIKINIS. Head and neck atlas. 11 2012. Available at: <http://www.spl.harvard.edu/publications/item/view/2271>.
- [81] JANE TOURNOIS, RAHUL SRINIVASAN, AND PIERRE ALLIEZ. Perturbing Slivers in 3D Delaunay Meshes. In *Proceedings of the 18<sup>th</sup> International Meshing Roundtable*, pages 157–173. Salt Lake City, Utah, USA, October 2009. Sandia Labs.
- [82] XIANGMIN JIAO, ANDREW COLOMBI, XINLAI NI, AND JOHN HART. Anisotropic mesh adaptation for evolving triangulated surfaces. *Eng. with Comput.*, 26(4):363–376. 2010.
- [83] CLEMENS MARTIN JOACHIM KADOW. *Parallel Delaunay Refinement Mesh Generation*. 2004. PhD Thesis. Carnegie Mellon University.
- [84] RICKY A. KENDALL, MASHA SOSONKINA, WILLIAM D. GROPP, ROBERT W. NUMRICH, AND THOMAS STERLING. Parallel programming models applicable to cluster computing and beyond. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, A.M. Bruaset and A. Tveito, editors, pages 3–55. Springer, 2005.

- [85] BRYAN MATTHEW KLINGNER AND JONATHAN RICHARD SHEWCHUK. Aggressive tetrahedral mesh improvement. In *Proceedings of the International Meshing Roundtable*, pages 3–23. Springer, 2007.
- [86] MILIND KULKARNI, PATRICK CARRIBAULT, KESHAV PINGALI, GANESH RAMANARAYANAN, BRUCE WALTER, KAVITA BALA, AND L. PAUL CHEW. Scheduling strategies for optimistic parallel execution of irregular programs. In *Proc. Symp. on Parallelism in algorithms and architectures (SPAA)*, pages 217–228. New York, NY, USA, 2008. ACM.
- [87] A. KUPRAT, A. KHAMAYSEH, D. GEORGE, AND L. LARKEY. Volume conserving smoothing for piecewise linear curves, surfaces, and triple lines. *Journal of Computational Physics*, 172(1):99–118, 2001.
- [88] ANDREW P. KUPRAT AND DANIEL R. EINSTEIN. An anisotropic scale-invariant unstructured mesh generator suitable for volumetric imaging data. *J. Comput. Phys.*, 228(3):619–640, February 2009.
- [89] FRANÇOIS LABELLE AND JONATHAN RICHARD SHEWCHUK. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, 26(3):57.1–57.10, 2007.
- [90] XIANG-YANG LI. Generating Well-Shaped D-dimensional Delaunay Meshes. In *Computing and Combinatorics*, Jie Wang, editor, volume 2108 of *Lecture Notes in Computer Science*, pages 91–100. Springer Berlin / Heidelberg, 2001.
- [91] XIANG-YANG LI AND SHANG-HUA TENG. Generating Well-Shaped Delaunay meshed in 3D. In *Proceedings of the 12<sup>th</sup> annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–37, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
- [92] LEONIDAS LINARDAKIS AND NIKOS CHRISOCHOIDES. Graded Delaunay decoupling method for parallel guaranteed quality planar mesh generation. *SIAM Journal on Scientific Computing*, 30(4):1875–1891, March 2008.
- [93] ANWEI LIU AND BARRY JOE. On the shape of tetrahedra from bisection. *Math. Comput.*, 63:141–154, July 1994.
- [94] YIXUN LIU, CHENGJUN YAO, LIANGFU ZHOU, AND NIKOS CHRISOCHOIDES. A point based non-rigid registration for tumor resection using iMRI. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1217–1220. IEEE Press, April 2010.
- [95] RAINALD LÖHNER. A 2nd generation parallel advancing front grid generator. In *Proceedings of the 21st International Meshing Roundtable*, Xiangmin Jiao and Jean-Christophe Weill, editors, pages 457–474. Springer Berlin Heidelberg, 2013.
- [96] WILLIAM E. LORENSEN AND HARVEY E. CLINE. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.
- [97] CALVIN R. MAURER, QI RENSHENG, AND VIJAY RAGHAVAN. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, feb 2003.
- [98] GARY L. MILLER, DAFNA TALMOR, AND SHANG-HUA TENG. Data generation for geometric algorithms on non-uniform distributions. *Int. J. Comput. Geometry Appl.*, 9(6):577–598, 1999.
- [99] GARY L. MILLER, DAFNA TALMOR, SHANG-HUA TENG, AND NOEL WALKINGTON. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proceedings of the 27th Annu. ACM Sympos. Theory Comput.*, pages 683–692. ACM, 1995.

- [100] SCOTT A. MITCHELL. Cardinality bounds for triangulations with bounded minimum angle. In *CCCG*, pages 326–331, 1994.
- [101] SCOTT A. MITCHELL AND STEPHEN A. VAVASIS. Quality mesh generation in higher dimensions. *SIAM J. Comput.*, 29(4):1334–1370, February 2000.
- [102] NEIL MOLINO, ROBERT BRIDSON, JOSEPH TERAN, AND RONALD FEDKIW. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proceedings of the 12<sup>th</sup> International Meshing Roundtable*, pages 103–114. Sandia National Laboratories, September 2003.
- [103] L. NAJMAN, J. COUSTY, M. COUPRIE, H. TALBOT, S. CLĂMMENT-GUINAUDEAU, T. GOISEN, AND J. GAROT. An open, clinically-validated database of 3D+t cine-mr images of the left ventricle with associated manual and automated segmentation. <http://www.laurentnajman.org/heart/index.html>.
- [104] DEMIAN NAVE, PAUL CHEW, AND NIKOS CHRISOCHOIDES. Guaranteed-quality parallel Delaunay refinement for restricted polyhedral domains. In *ACM Symposium on Computational Geometry (SoCG)*, pages 135–144. ACM, July 2002.
- [105] DEMIAN NAVE, NIKOS CHRISOCHOIDES, AND PAUL CHEW. Parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28:191–215, 2004.
- [106] MARTIN NEUMÜLLER AND OLAF STEINBACH. Refinement of flexible space–time finite element meshes and discontinuous Galerkin methods. *Computing and Visualization in Science*, 14:189–205, 2011.
- [107] T. OKUSANYA AND J. PERAIRE. 3D parallel unstructured mesh generation, 1997. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7898>.
- [108] LEONID OLIKER AND RUPAK BISWAS. Parallelization of a dynamic unstructured algorithm using three leading programming paradigms. *IEEE Trans. Parallel Distrib. Syst.*, 11(9):931–940, September 2000.
- [109] STEVE OUDOT, LAURENT RINEAU, AND MARIETTE YVINEC. Meshing volumes bounded by smooth surfaces. In *Proceedings of the International Meshing Roundtable*, pages 203–219. Springer-Verlag, September 2005.
- [110] JEAN-PHILIPPE PONS, FLORENT SÉGONNE, JEAN-DANIEL BOISSONNAT, LAURENT RINEAU, MARIETTE YVINEC, AND RENAUD KERIVEN. High-Quality Consistent Meshing of Multi-label Datasets. In *Information Processing in Medical Imaging*, pages 198–210. Springer Berlin Heidelberg, 2007.
- [111] THOMAS C. S. RENDALL, CHRISTIAN B. ALLEN, AND EDWARD D. C. POWER. Conservative unsteady aerodynamic simulation of arbitrary boundary motion using structured and unstructured meshes in time. *International Journal for Numerical Methods in Fluids*, 70(12):1518–1542, 2012.
- [112] J.A. RICHOLT, M. JAKAB, AND R. KIKINIS. SPL Knee Atlas. January 2011. Available at: [http://www.spl.harvard.edu/publications\\_item/view/1953](http://www.spl.harvard.edu/publications_item/view/1953).
- [113] LAURENT RINEAU AND MARIETTE YVINEC. Meshing 3D domains bounded by piecewise smooth surfaces. In *Proceedings of the International Meshing Roundtable*, pages 443–460, 2007.

- [114] R. SAID, N.P. WEATHERILL, K. MORGAN, AND N.A. VERHOEVEN. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 177(1-2):109-125, 1999.
- [115] WILLIAM N. SCHERER, III AND MICHAEL L. SCOTT. Advanced contention management for dynamic software transactional memory. In *Proceedings of the 24<sup>th</sup> annual ACM symposium on Principles of distributed computing*, PODC '05, pages 240-248. ACM, 2005.
- [116] P. SEWELL, T.M. BENSON, C. CHRISTOPOULOS, D. W P THOMAS, A. VUKOVIC, AND J.G. WYKES. Transmission-line modeling (TLM) based upon unstructured tetrahedral meshes. *Microwave Theory and Techniques, IEEE Transactions on*, 53(6):1919-1928, 2005.
- [117] JONATHAN RICHARD SHEWCHUK. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14<sup>th</sup> ACM Symposium on Computational Geometry*, pages 86-95, Minneapolis, MN, 1998. ACM.
- [118] JONATHAN RICHARD SHEWCHUK. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21-74, May 2002.
- [119] JONATHAN RICHARD SHEWCHUK. What is a Good Linear Element? - Interpolation, Conditioning, and Quality Measures. In *Proceedings of the 11<sup>th</sup> International Meshing Roundtable*, pages 115-126. Sandia National Laboratories, September 2002.
- [120] HANG SI. Constrained Delaunay tetrahedral mesh generation and refinement. *Finite Elements in Analysis and Design*, 46:33-46, 2010.
- [121] HANG SI. TetGen, A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator. <http://tetgen.berlios.de/>, v1.4.3.
- [122] AMY HENDERSON SQUILLACOTE. *ParaView Guide, A Parallel Visualization Application*. Kitware Inc., 2008.
- [123] ROBERT STAUBS, ANDRIY FEDOROV, LEONIDAS LINARDAKIS, BENJAMIN DUNTON, AND NIKOS CHRISOCHOIDES. Parallel n-dimensional exact signed euclidean distance transform. *The Insight Journal*, September 2006.
- [124] SHRIPAD THITE. Efficient spacetime meshing with nonlocal cone constraints. In *13th International Meshing Roundtable*, pages 47-58, 2004.
- [125] YANGHAI TSIN, KLAUS KIRCHBERG, GUENTER LAURITSCH, AND CHENYANG XU. A deformation tracking approach to 4d coronary artery tree reconstruction. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2009*, Guang-Zhong Yang, David Hawkes, Daniel Rueckert, Alison Noble, and Chris Taylor, editors, volume 5762 of *Lecture Notes in Computer Science*. pages 68-75. Springer Berlin / Heidelberg, 2009.
- [126] TIANKAI TU, DAVID R. O' HALLARON, AND OMAR GHATTAS. Scalable parallel octree meshing for terascale applications. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 4-. IEEE Computer Society, 2005.
- [127] M VON SIEBENTHAL, G SZÉKELY, U GAMPER, P BOESIGER, A LOMAX, AND PH CATTIN. 4D MR imaging of respiratory organ motion and its variability. *Physics in Medicine and Biology*. 52(6):1547-1564, 2007.
- [128] DAVID F. WATSON. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*. 24:167-172, 1981.

- [129] ERNST WEIGANG, FABIAN A. KARI, FRIEDHELM BEYERSDORF, MAXIMILIAN LUEHR, CHRISTIAN D. ETZ, ALEX FRYDRYCHOWICZ, ANDREAS HARLOFF, AND MICHAEL MARKL. Flow-sensitive four-dimensional magnetic resonance imaging: flow patterns in ascending aortic aneurysms. *European Journal of Cardio-Thoracic Surgery*, 34(1):11–16, 2008.
- [130] GEORGE ZAGARIS, SHAHYAR PIRZADEH, AND NIKOS CHRISOCHOIDES. A framework for parallel unstructured grid generation for practical aerodynamic simulations. In *47th AIAA Aerospace Sciences Meeting*, January 2009.
- [131] MIN ZHOU, ONKAR SAHNI, TING XIE, MARK S. SHEPHARD, AND KENNETH E. JANSEN. Unstructured mesh partition improvement for implicit finite element at extreme scale. *J. Supercomput.*, 59(3):1218–1228, March 2012.



## VITA

## Panagiotis A. Foteinos

Panagiotis Foteinos joined the Computer Science Department of the College of William and Mary in Fall 2007 towards a PhD Degree. He is currently a PhD candidate and research assistant. In Fall 2010, he concurrently worked at Old Dominion University as a visitor researcher. He entered the Electrical and Computer Engineering Department of University of Thessaly in Fall 2002, where he received his Bachelor's on Computer Science.