2007

Decoupling method for parallel Delaunay two-dimensional mesh generation

Leonidas Linardakis
College of William & Mary - Arts & Sciences

Decoupling Method for Parallel Delaunay 2D Mesh Generation

Leonidas Linardakis

Athens Greece

M.Sc. C.S. College of William & Mary
M.Sc. Mathematics University of Ioannina

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
August 2007

APPROVAL PAGE

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

Doctor of Philosophy

[illegible]

Leonidas Linardakis

Approved by the Committee, June 2007

[illegible]

Committee Chair
Associate Professor Nikos Chrisochoides, Computer Science
The College of William & Mary

[illegible]

Assistant Professor Qun Li, Computer Science
The College of William & Mary

Weizhen Mao

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

[illegible]

Associate Professor Virginia Torczon, Computer Science
The College of William & Mary

[illegible]

Professor Noel J. Walkington, Mathematical Sciences
Carnegie Mellon University

ABSTRACT PAGE

Meshes are central structures for numerical methods, such as the finite element method. These numerical methods require high quality refined meshes in order to achieve good approximations of the analytical model. Unstructured meshes are the most popular; their adaptive nature allows them to give boundary conforming meshes of good quality, with optimal size. The most widely studied 2D mesh generation method is the Delaunay method.

Creating in parallel guaranteed quality large unstructured meshes is a challenging problem. The Delaunay refinement procedure is memory intensive with unpredictable computational behavior. Moreover, geometries may be quite complex, adding difficulty to parallelize the mesh generation. Parallel mesh generation procedures decompose the original mesh generation problem into smaller subproblems that can be solved in parallel. The subproblems can be treated as either completely or partially coupled, or they can be treated as completely decoupled.

Parallel mesh generation procedures that are based on geometric domain decompositions require the permanent separators to be of good quality (in terms of their angles and length), in order to maintain the mesh quality. The *Medial Axis domain decomposition*, an innovative geometric domain decomposition procedure that addresses this problem, is introduced. The Medial Axis domain decomposition is of high quality in terms of the formed angles, and provides separators of small size, and also good work-load balance. It presents for the first time a decomposition method suitable for parallel meshing procedures that are based on geometric domain decompositions.

The *decoupling method* for parallel Delaunay 2D mesh generation is a highly efficient and effective parallel procedure, able to generate billions of elements in a few hundred of seconds, on distributed memory machines. Our mathematical formulation introduces the notion of the *decoupling path*, which guarantees the decoupling property, and also the quality and conformity of the Delaunay submeshes. The subdomains are meshed independently, and as a result, the method eliminates the communication and the synchronization during the parallel meshing. A method for shielding small angles is introduced, so that the decoupled parallel Delaunay algorithm can be applied on domains with small angles. Moreover, we present the construction of a sizing function, that encompasses an existing sizing function and also geometric features and small angles. The decoupling procedure can be used for parallel graded Delaunay mesh generation, controlled by the sizing function.

The decoupling approach allows 100% code re-use of existing, fine-tuned and well tested, sequential mesh generators, minimizing the effort of code parallelization. Our results indicate high scalability of the decoupling approach, and also show superlinear speedups, when compared to the sequential library.

Contents

Acknowledgments **iii**

1 Introduction **1**

1.1 Delaunay Mesh Generation and Ruppert's Algorithm 2
1.2 Parallel Mesh Generation 4
1.3 The Delaunay Decoupling Approach 6

2 The Geometric Domain Decomposion Problem **9**

3 The Medial Axis Domain Decomposition **13**

3.1 The Medial Axis Domain Decomposition 13
3.2 Junction Triangles 15
3.3 The MADD Procedure 18
3.4 The MADD First Algorithm 20
3.5 The MADD Second Algorithm 25
3.6 Proof of Correctness 30
3.7 N-way Decomposition 31

4 MADD Enhancements **33**

4.1 Static and Dynamic MADD 33
4.2 Separator Smoothing Procedure 35
4.3 Parallel MADD 37
4.4 N-way Graded Decomposition 38

5 MADD Implementation and Experimental Results **43**

5.1 Implementation 43
5.2 Experimental Results 43

6 The Uniform Decoupling Method **48**

6.1 The Decoupling Zone 48
6.2 Construction of the Decoupling Path 51
6.3 Proof of Correctness 52
6.4 The Parallel Delaunay Decoupling Procedure 55
6.5 Performance Evaluation 60

7 The Graded Decoupling Method **70**

7.1 Graded Mesh Generation 70
7.2 The Graded Decoupling Approach 71
7.3 Gradation Controlled Domain Decomposition 72

7.4 The Graded Delaunay Decoupling Path 78
7.5 Construction of the Graded Delaunay Decoupling Path 80
7.6 The Graded Delaunay Decoupling Procedure 81
7.7 Performance Evaluation 84

8 The Decoupling Method for Domains with Small Angles 87
8.1 A Shielding Procedure for Small Input Angles 88
8.2 Construction of the Sizing Function 99
8.3 The Decoupling Method for Domains with Small Angles 103

9 Conclusions and Future Work 106

Bibliography 109

VITA 116

Acknowledgments

The members of the committee made usefull comments that helped make this work more complete. I thank them for their help. I also thank Tom Crockett for providing helpful information on the workings of SciClone and the MPI library.

This work was partially funded by NSF ACI-0085969 and NGS-0203974, the Computer Science Department of the College of William and Mary, and the Virginia Institute of Marine Science under the grants ONR N00014-05-1-0831 and NOAA NA04NOS4730254. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained herein are those of the author, and do not necessarily represent official views or endorsements of the funding organizations.

This work was performed using computational facilities at the College of William and Mary, which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

Chapter 1

Introduction

A mesh is a covering (or tessellation) of a bounded n-dimensional domain Ω by a set of "simple" n-dimensional elements. A mesh $\mathcal{M}$ consists of an hierarchy of sets of geometric entities $\mathcal{M} = \{\emptyset, E^0, E^1, \cdots, E^n\}$. The hierarchy represents the dimensions of the entities, E^0 is a set of 0-dimensional entities (points), E^1 is a set of segments, and so on. The entities of a mesh obey the following rules.

1. Any entity $A \in \cup E^i$ belongs to Ω, $A \in \Omega$. Moreover, the union of all the n-dimensional entities is a cover of Ω,
$$\cup_{A \in E^n} A = \Omega.$$
2. The intersection between any two k-dimensional entities is an entity of lower dimension (including the empty set). So, for any $0 < k \leq n$ and any two entities $A, B \in E^k$, we have
$$A \cap B \in E_l, \quad \text{where } l < k.$$
3. The k-dimensional interior of any k-dimensional entity does not contain entities of lower dimension. For $l < k$ and $A \in E^k$, $B \in E^l$, we have
$$\text{int}_k(A) \cap B = \emptyset.$$

In the case of a simplex mesh any entity $A \in E^k$ is a k-dimensional simplex. In the two dimensional (2D) Euclidean space a simplex mesh consists of triangles.

Meshes are central in numerical methods, such as the finite element method (FEM), and finite volume method (FVM). These numerical methods are indispensable for simulating complex physical phenomena, and they require high quality refined meshes, in order to guarantee good approximations of the analytical model. The quality of the mesh is measured by the size of the elements (and the induced gradation), and their shape (the angles they form). Unstructured triangular meshes are popular because they demonstrate adaptivity to the geometry, giving boundary conforming meshes of good quality, and also optimal size. Delaunay meshes are widely used 2D triangular meshes.

1.1 Delaunay Mesh Generation and Ruppert's Algorithm

The Delaunay triangulation is named after the Russian mathematician Boris Nikolaevich Delone [31]. It is a triangulation such that the circumcircle (the circumscribed circle) of every triangle is empty, that is it does not contain any other vertex of the triangulation (see Figure 1.1). This property is referred as the *empty circumcircle property.* The Delaunay triangulation of a set of points in general position is unique, and maximizes the minimum angle over all possible triangulations [82].

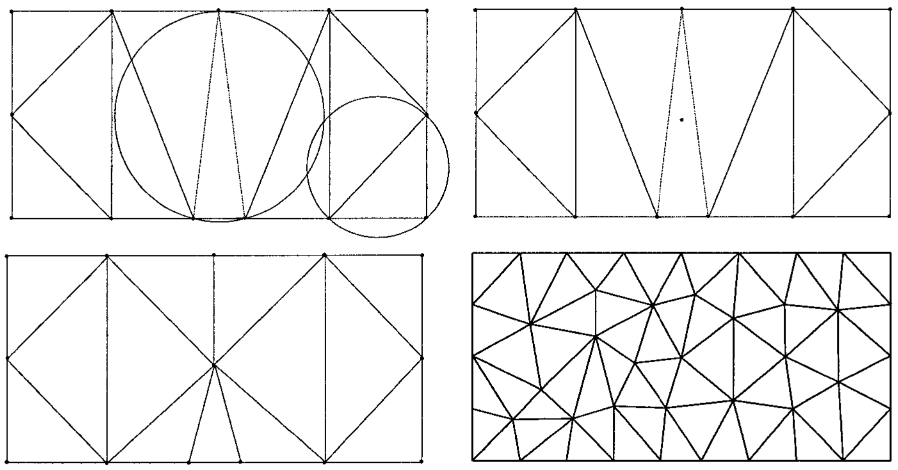


Figure 1.1: Delaunay triangulation and mesh generation. **Top left**, in Delaunay triangulation the circumcircles of the triangles are empty. **Top right**, in the Delaunay mesh generation the circumcenters of the 'bad' triangles are inserted and **bottom left** the mesh is re-triangulated. **Bottom right**, the circumcenter point insertion and triangulation is irregular.

Delaunay refinement procedures provide theoretical guarantees for the mesh quality (angles), and at the same time are very efficient. In the Delaunay mesh generation points are inserted in the triangulation (commonly called Steiner points) in order to improve the quality of the mesh (see Figure 1.1). A triangle is considered "bad" when it contains a small angle, or equivalently when the *circumradius to shortest edge ratio* is large. In addition to improving the quality of the mesh in terms of the angles, the same refinement procedure is used to reduce the size of the triangles, so that the maximum triangle area is bounded by a desirable size.

Delaunay mesh refinement algorithms became popular by Paul Chew [17, 18] and Jim Ruppert [72]. Ruppert's algorithm provides both quality and gradation guarantees and has been the basis of extensive study and further optimizations in both efficiency and effectiveness. A detailed study is given by Jonathan Shewchuk [78, 81], and also a state-of-art implementation [77, 86]. Miller et al. present an analysis of Ruppert's algorithm in [59]. Several improvements have been proposed to extent the algorithm for 3D and also to cope with small boundary angles (see [15, 14, 58, 65, 80, 79]). The reader will find more information on mesh generation and Delaunay triangulation in [6, 7, 34, 35, 38, 64].

Ruppert's Algorithm. For the sequential mesh procedure we will consider Ruppert's algorithm [72]. This is a Delaunay mesh refinement algorithm for 2D domains, that guarantees the quality of the elements. It creates an initial triangulation and follows an incremental approach to refine the mesh. Triangles which have circumradius to shortest edge ratio greater than $\sqrt{2}$ are split, by inserting points in their circumcenters and constructing a new Delaunay triangulation. Special treatment is required near the boundary of the domain. If a point is inserted too close to the boundary, it will result either poor triangle quality, or an unnecessary large number of triangles. Points that are inside the diametral circle of a boundary segment will not be inserted. Instead, the boundary segment that is encroached will be split in half, and the new Delaunay triangulation includes the two subsegments. The algorithm maintains the Delaunay property after the insertion of each point. In order to guarantee the termination of this procedure the boundary angles should be at least 60°[1].

Let Ω be a 2-dimensional domain formed by by a set of points and line segments intersecting only at their end points. In other words, Ω is defined by a planar straight line

[1]This condition is relaxed in improved versions of the algorithm.

graph (PSLG). An entity of the domain is either a vertex or a segment of the boundary. Two entities are incident when they share a common point.

Definition 1. *The* minimum local feature size *of Ω is defined as the minimum distance between two non incident entities*[2] *[78]; it will be denoted by $lfs_{\min}(\Omega)$.*

The following theorem is known to be true [78]:

Theorem 1. *Let every two incident segments of Ω to form an angle no less than $60°$. Ruppert's algorithm terminates when applied on Ω, giving a mesh of triangles with circumradius to shortest edge ratio at most $\sqrt{2}$ and with no triangle edge shorter than $lfs_{\min}(\Omega)$.*

Ruppert's algorithm is not computational expensive, but is memory intensive and has unpredictable computational behavior, which is input dependent.

1.2 Parallel Mesh Generation

In order to generate a mesh on a multicomputer environment it is necessary to decompose the mesh generation problem. This can be achieved in two ways: (i) by a *mesh data-decomposition* approach, or (ii) by a *geometric domain decomposition* approach. Mesh data-decomposition approaches decompose the mesh data structure, without inserting geometric separators into the geometry. On the other hand, geometric domain decompositions partition the domain by inserting separators into the geometry, and these separators will be a permanent part of the geometry. Methods that follow a mixed approach have also been proposed (see for example Shephard et al. [75], and also de Cougny and Shephard [29]), but they inherently present a higher degree of complexity. Another classification of parallel meshing methods is given by de Cougny and Shephard [27]: (a) mesh interfaces and subdomains concurrently, (b) premesh the interfaces, and (c) postmesh the interfaces. A recent survey of parallel mesh generation methods is given by Chrisochoides [23], where the parallel meshing methods are classified as (a) tightly-coupled, (b) partially-coupled, and (c) decoupled methods.

[2]For a PSLG domain the minimum local feature size will be the minimum distance between two vertices, or a vertex and a segment [78].

Mesh data-decomposition methods are attractive, and have been studied extensively, because they do not have have to face the difficult geometric domain decomposition problem. A data-decomposition approach is used by Löhner and Cebral [55], who employ an octree decomposition of the domain to partition the current front in an advancing front mesh method. For parallel Delaunay mesh generation an octree decomposition is used by Chernikov and Chrisochoides [16] to identify parts of a Delaunay mesh that can be refined independently. Another common data-decomposition approach is to create an initial mesh, and then decompose it using a graph partitioner. The refining procedure can applied on each part of the mesh, with some communication to maintain the conformity; this approach has been followed by Chrisochoides and Nave [24]. Finally, Kadow and Walkington [43] employ a projective method [8] and alternate cuts to create in parallel, and decompose, an initial Delaunay triangulation. The triangulation is further refined in parallel, and the communication is controlled via an encroachment zone along the cuts.

Geometric domain decomposition approaches insert separators into the domain, and these are treated as a constrained part of the geometry (see Fig. 1.2). The separators will be a permanent part the geometry, and they should observe certain quality conditions, like the angles they form. These conditions impose additional difficulty to parallel mesh generation methods that use geometric decompositions. On the other hand these methods have the advantage of low cost of communication during the parallel run. The geometric domain decomposition methods that have been proposed fall into two categories: (a) Those that mesh interfaces and subdomains concurrently, and (b) those that premesh the interfaces. The parallel constrained Delaunay Triangulation, proposed by Chew et al. [19], meshes concurrently the interfaces and the subdomains. The interfaces are treated as external boundary by each process, and a message is sent to the neighboring subdomain when an interface is split. Another concurrent approach, based on templates, is described by Pebay [66].

Methods that mesh a priori the interfaces target mainly the elimination of the communication during the parallel mesh procedure, and also have the benefit of high code re-use (the sequential mesher can be used without, or minimal, modifications). The core Delaunay mesh refinement procedure is fast (although memory intensive), and the increasing

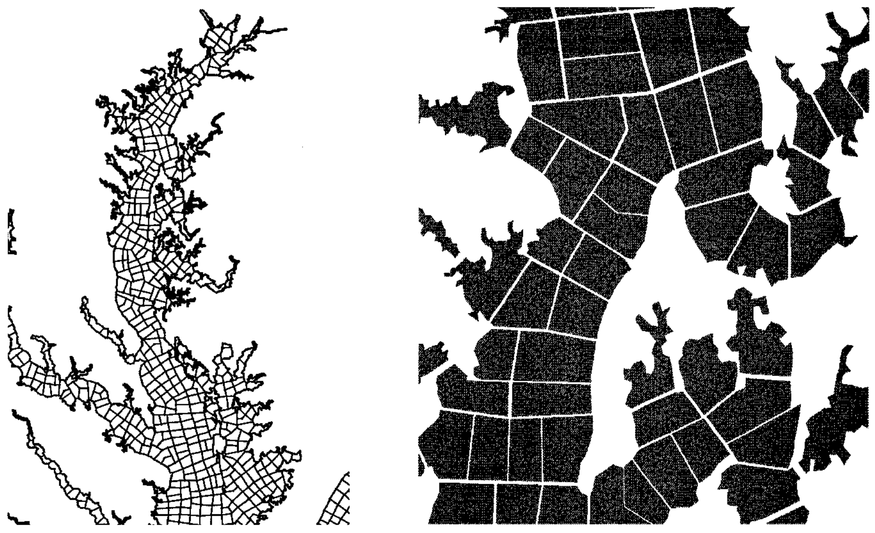


Figure 1.2: **Left:** Part of the Chesapeake Bay geometry decomposed uniformly by the MADD method. **Right:** Detail of the Delaunay mesh of the subdomains. The mesh was created by the decoupling procedure.

processing power of the CPUs reveals the network as the bottleneck for parallel processing. Therefore it is natural to attempt to eliminate the communication. Parallel mesh generation procedures with no communication for distributed memory machines, based on prerefining the interfaces, have been studied in the past. Nigel Weatherill and his collaborators have proposed an a priori scheme for parallel mesh generation on distributed memory computers (see Gaither et al. [36], Said et al. [73], Larwood et al. [48]). The procedure though does not preserve the Delaunay properties globally and does not provide quality guarantees along the separators. A projective method that eliminates the communication for parallel Delaunay triangulation is described by Blelloch et al. [8]. In [37] J. Galtier and P. L. George propose a parallel projective Delaunay mesh generation method which guarantees the quality of the elements and eliminates communication, but may suffer setbacks in the form of regenerating part of the mesh.

1.3 The Delaunay Decoupling Approach

The methods that mesh a priori the interfaces, and eliminate communication, face two problems in order to guarantee the termination of the parallel procedure and also the stability in terms of the quality, conformity and size of the final global mesh. First it is necessary to produce quality geometric domain decompositions that will have minimal negative effect to the mesh generation proccess. The second problem is to calculate the interface refining size, so that we can guarantee the conformity of the final mesh, without compromising its quality. The solution these two problems for 2D domains is the subject of the work in hand.

The Parallel Delaunay decoupling method consists of two major steps (see Fig. 1.2): the Medial Axis domain decomposition step, and the decoupling procedure step.

The domain decomposition procedure should produce decompositions of good quality in terms of the created angles, and also the size of the separators. If small angles are created during the decomposition procedure, these constitute artifacts that will distort the quality and size of the final mesh, and also will affect the stability and performance of the mesh generator. The separators constitute artifacts, and should be kept at minimal size. On the other hand, the Delaunay mesh generation procedure is unpredictable, therefore special care should be taken to achieve good load balance. Finally, the domain decomposition should be able to accommodate graded parallel mesh generation procedures, thus it should have the capability to produce graded decompositions according to given sizing criteria.

The second step is the *decoupling procedure*. The decoupling property allows the subdomains to be meshed in parallel and independently, and at the same time guarantees the conformity and quality of the global mesh. The *decoupling zone* and the *decoupling path* give a general mathematical formulation for decoupling any Delaunay mesh generation procedure. The separators created by the domain decomposition are refined during a preprocessing step before the parallel mesh generation procedure. The refining procedure results the decoupling property, which allows us to create quality Delaunay meshes in parallel, while eliminating the communication. Our results indicate high scalability of the decoupling approach, and also show superlinear speed-ups, when compared to the sequential library. Moreover, the decoupling approach allows 100% code re-use of existing, state-of-art, sequential mesh generators, minimizing the effort of code parallelization.

The *Medial Axis domain decomposition* (MADD), that we propose in this work, is an innovative domain decomposition procedure, based on an approximation of the Medial Axis of the domain. The MADD method fullfills the requirements described above, and produces decompositions suitable for parallel mesh generation. The geometric domain decomposition problem is described in Chapter 2. In Chapter 3 we present the core MADD algorithm, while in Chapter 4 several extentions and improvements are examined. Finally, in Chapter 5 we present our experimental results.

We describe the notions of the decoupling zone and the decoupling path, in Chapter 6. In the same chapter the uniform decoupling procedure is described. Graded Delaunay meshes, governed by a sizing function or background grid, can also be created in parallel using the decoupling approach. Again, the quality and conformity of the mesh is guaranteed, while the communication is eliminated. This procedure is described in Chapter 7. Finally, a shielding method for pre-processing small input angles is described in Chapter 8. In addition, a method for constructing a sizing function that encompasses an existing sizing function, and also geometric features and small angles, is described in the same chapter. This procedure allows the decoupling method to be applied on domains with small angles, creating in parallel graded Delaunay 2D meshes.

Definition 2. *In the rest of this exposition we define the domain Ω to be the closure of an open connected bounded set in $\mathbb{R}^2$. The boundary $\partial\Omega$ is defined by a planar straight line graph (PSLG), which is formed by a set of line segments, intersecting only at their end points.*

The above definition allows the existence of holes inside the domain, but does not allow internal boundaries. The algorithms we present can be extended to also handle internal boundaries.

Chapter 2

The Geometric Domain Decomposion Problem

Although the Domain Decomposition (DD) problem has been studied for more than twenty years in the context of parallel computing, there are many aspects of this problem which are unsolved. DD methods have been used for solving numerically partial differential equations using parallel computing (cf. [83]). In the context of parallel mesh generation we encounter the Geometric Domain Decomposition problem (GDD). We will study the formulation, solution and implementation of the GDD problem for a continuous 2-dimensional (2D) domain Ω. Our goal is to decompose Ω into non-overlapping subdomains D_i, so that the subdomains D_i create no new artifacts, such as small angles between the separators ∂D_i, and the separators and the external boundary $\partial\Omega$. These decompositions are suitable for stable parallel graded mesh generation procedures, where the termination of these procedures and the quality of the resulting elements depend on the features of the subdomains. Furthermore, the same decompositions can be used for the next step, by the parallel FEM or FD solver. However, the geometric domain decomposition we describe does not depend on how the mesh is used, or what is the PDE solving method.

Geometric domain decomposition techniques partition the domain geometry into subdomains; the subdomains are created by inserting internal boundaries (separators) into the domain. Parallel mesh generation procedures that follow this approach require low communication [19], or no communication at all [37, 73, 52], and thus are very efficient. Geometric domain decomposition methods can be characterized as *topology–based* or *geometry–based.* Typically, topology–based techniques partition a mesh of the domain, or the dual graph of a background mesh, giving a decomposition of the domain. This approach is followed by the Metis library [45]. On the other hand, geometry–based techniques take into account the geometric characteristics of the domain. For example, the Recursive Coordinate Bisection approach [5] recursively bisects the domain along the axes, while the Inertial method [61] uses the inertia axis of the domain to produce a decomposition. Finally, libraries like Chaco [40] provide both topology and geometry-based approaches.

Guaranteed quality mesh generation algorithms [17, 18, 72] produce elements with good aspect ratio and good angles. These algorithms require the initial boundary angles to be within certain good bounds. For example, Ruppert's algorithm [72] requires boundary angles (the angles formed by the boundary edges) no less than $60°$, in order to guarantee the termination[1]. When these algorithms are used in domain decomposition based parallel mesh generation procedures, the separators are treated as external boundary of each subdomain. Consequently, the domain decomposition should create separators that meet the requirements of the mesh generation algorithm. Even in the cases where the meshing algorithm can handle small input angles (as in [14, 65, 80]), these are undesirable when formed by the separators. If small angles are created by the decomposition procedure, these constitute artifacts that will have a negative effect to the quality and size of the final mesh, and also will affect the performance of the mesh generator. Therefore the constructed separators should form angles no less than a given bound Φ_0, which is determined by the sequential mesh generation procedure that will be used to mesh the individual subdomains.

The performance of the parallel mesh generation is affected by the required communication and the work-load balance among the processors. If there is communication, this is usually proportional to the size of the separator, therefore, one of our objectives in the domain decomposition step is to minimize the size of the separators. On the other hand, the load balancing problem is best addressed by over-decomposing the domain [22]. Over-decomposition allows both static and dynamic load balancing methods to distribute equally the work-load among the processors more effectively [2, 52]. These methods though will be less effective, if some of the subdomains represent a much larger work-load than the average[2]. Therefore, we should keep the maximum area of the subdomains close to the

[1]This condition is relaxed in improved versions of the algorithm.

[2]Small work loads do not create load-balancing problems when over-decomposition is used. On the contrary, the resulting granularity can be used to improve the load balance, especially on heterogenous

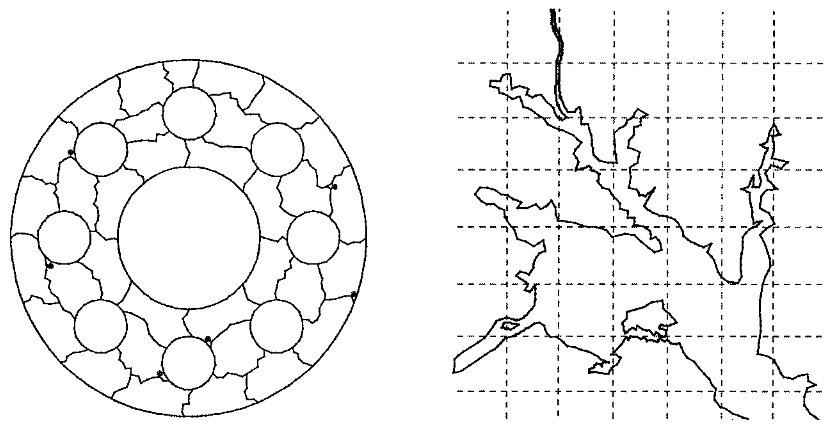


Figure 2.1: **Left:** The Pipe geometry. The angles produced by graph-based partitioner, like Metis, depend on the background mesh, and can be as small as the smaller angle of the mesh. The angles marked with dots are "small" (less than 60°). **Right:** Part of the Chesapeake bay geometry. When the geometry is complicated, methods like the Recursive Coordinate Bisection and the Inertial Method can produce arbitrary small angles between the separators and the domain boundary, and also can place separators arbitrary close to the boundary.

average subdomain area[3].

In conclusion, a geometric domain decomposition is suitable for stable parallel mesh generation, if it satisfies the following criteria.

C1. Create good angles, i.e., angles no smaller than a given tolerance $\Phi_0 < \pi/2$. The value of Φ_0 is determined by the sequential, guaranteed quality, mesh generation algorithm (for Ruppert's algorithm we use the value $\Phi_0 = 60°$).

C2. The length of the separator should be relatively small.

C3. The maximum area of the subdomains should be close to the average subdomain area.

Previous DD approaches are very successful for traditional parallel PDE solvers, but they were not developed for parallel mesh generation procedures, and thus do not address the problem of the formed angles. On the other hand, domain decomposition procedures used for parallel mesh generation aim mostly to solve the load balancing problem and to minimize the communication [32, 44, 87]. For example, graph based partitioning algorithms,

environments.

[3]The area of the subdomains does not always reflect to work-load of the mesh generation procedure. However, for well shaped subdomains, as the ones produced by MADD, and for Delaunay mesh generators, the work-load is analogous to the area of the subdomain (see Section 6.4)

like Metis, give well-balanced decompositions with small separators, but the angles formed by the separators depend on the background mesh, and they can be as small as the smallest angle of the mesh (see Figs. 2.1 left, and 5.5). On the other hand, methods like the Recursive Coordinate Bisection and the Inertial Method can create arbitrary small angles, and also place the separators arbitrary close to the boundary (see Fig. 2.1 right), so they are unsuitable for parallel mesh generation procedures.

In addition to the requirements described above, the domain decomposition should be able to accommodate graded parallel mesh generation procedures, and thus it should have the capability to produce graded decompositions according to given sizing and gradation criteria. The Medial Axis domain decomposition (MADD) addresses all the above conditions, and it is described in the following chapters.

Chapter 3

The Medial Axis Domain Decomposition

3.1 The Medial Axis Domain Decomposition

The Medial Axis Domain Decomposition (MADD) method is based on an approximation of the medial axis (MA) of the domain. The MA was introduced by Blum [9] as a way to depict the shape of an object, and has been studied extensively [13, 12, 20, 50, 76, 89]. It has also found numerous applications in the context of mesh generation (see [1, 35, 39, 69]). A decomposition procedure based on critical medial axis points [85] for sequential quadrilateral mesh generation is described by Tam et al [84]. This procedure was proposed for parallel mesh generation by Chrisochoides [21].

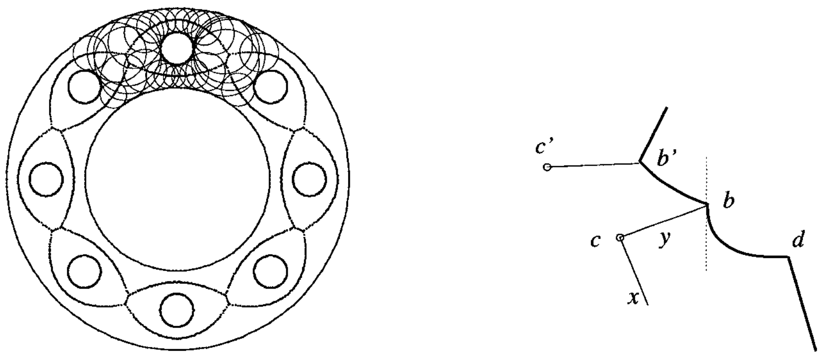


Figure 3.1: **Left:** The medial axis of a domain is the locus of the centers of the maximal inscribed circles. **Right:** The angles formed by a point c' of the medial axis and its contact points (b') are at least $90°$. The angle at b can not be less than 90^o, unless c is a not a point of the medial axis, or b its not its contact point

A circle $C \subseteq \Omega$ is said to be maximal in Ω, if there is no other circle $C' \subseteq \Omega$ such that $C \subsetneq C'$. The closure of the locus of the circumcenters of all maximal circles in Ω is called the *medial axis* Ω (see Fig 3.1 left), and will be denoted by MA(Ω). The intersection of a

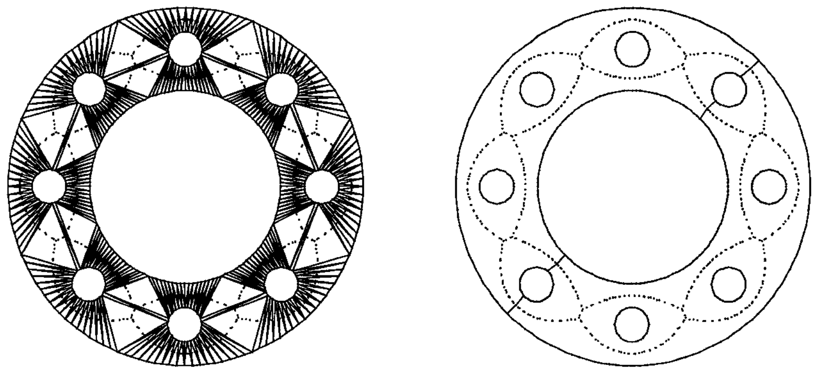


Figure 3.2: **Left:** The Delaunay triangulation of the pipe intersection. The circumcenters of the triangles approximate the medial axis. **Right:** The circumcenters are the Voronoi points. The separator is formed by selecting a subset of the Voronoi points and connecting them with the boundary.

boundary of Ω and a maximal circle C is not empty. The points $C \cap \partial\Omega$, where a maximal circle C intersect the boundary, are called *contact points* of c, where c is the center of C. Every point $c \in \mathrm{MA}(\Omega) \setminus \partial\Omega$ has at least two contact points. The domain decomposition method I propose is based on the following simple geometric property:

Lemma 2. *Let b a contact point of $c \in MA(\Omega)$. The angles formed by the segment cb and the tangent of the boundary $\partial\Omega$ at b are at least $\pi/2$.*

Proof. We will prove the lemma in the general case when Ω has a piecewise C^1 boundary. Suppose that the proposition is not true. Then there is a point $c \in MA(\Omega)$ of the medial axis and a contact point $b \in \partial\Omega$ of c, such that cb forms an angle $\phi < \pi/2$ with the boundary at b (see Fig. 3.1, right). Take c to be the origin of the axes and cb to define the y axis. Without loss of generality we assume that ϕ is formed by the tangent from the right. Let $(x(s), y(s))$ be locally the normal parametric representation of the curve, with $b = (x(0), y(0)) = (0, y(0))$ and $x(s) \geq 0$. We have $y(0) > 0$. Since $\phi < \pi/2$, we have $y'(0) < 0$. Let $R(s) = x^2(s) + y^2(s)$ be the square of the distance between c and the points of the curve. Because b is a contact point of c, it must be $R(s) \geq R(0) = |cb|^2$. We have $R'(0) = 2y(0)y'(0) < 0$. This means that locally $R(s) < R(0)$, which is a contradiction. □

The medial axis of Ω can be approximated by Voronoi points of a discretization of the domain [13, 12]. We make use of the property of Lemma 2 to construct separators that consist of linear segments which connect the Voronoi points to the boundary. The approximation of the $MA(\Omega)$ is achieved in two steps: (1) discretization of the boundary, and (2) computation of a boundary conforming Delaunay triangulation using the points from step (1). The circumcenters of the Delaunay triangles are the Voronoi points of the boundary vertices. The separators will be formed by connecting these circumcenters to the vertices of the Delaunay triangles. Figure 3.2 depicts the boundary conforming mesh of the cross section of a rocket (left), and the media axis approximation and a 2-way separator for the same geometry (right).

The level of the discretization of the boundary determines the quality of the approximation of the medial axis. However, our goal is not to approximate accurately the medial axis, but to obtain good angles from the separator. Therefore our criteria for the discretization of the domain will be determined from the quality of the angles formed between the separators and the boundary $\partial\Omega$. We achieve our goal by defining a new set of triangles.

Definition 3. *A 2-way decomposition of a domain Ω can be defined as follows. A* complete separator *$\mathcal{H} \subseteq \Omega$ is a finite set of simple paths (a continuous 1-1 map $h : [0,1] \rightarrow \Omega$), which we call* partial separators*, that do not intersect and define a decomposition D_1, D_2 of Ω, such that: D_1 and D_2 are connected sets, with $D_1 \cup D_2 = \Omega$, and for every path $P \subset \Omega$, which connects a point of D_1 to a point of D_2, we have $P \cap \mathcal{H} \neq \emptyset$.*

In Figure 3.2 right a two-way decomposition is depicted. The complete separator is formed by four partial separators.

3.2 Junction Triangles

Definition 4. *Let $\mathcal{T}$ be a Delaunay triangulation of a discretization Z_Ω of the boundary $\partial\Omega$. We call a triangle $t \in \mathcal{T}$ a* junction triangle*(see Fig. 3.3) if:*

1. *it includes its circumcenter c,*
2. *at least two of its edges are not in Z_Ω,*
3. *at least two of the segments defined by the circumcenter and the vertices of t form angles $\geq \Phi_o$, both with the boundary and each other.*

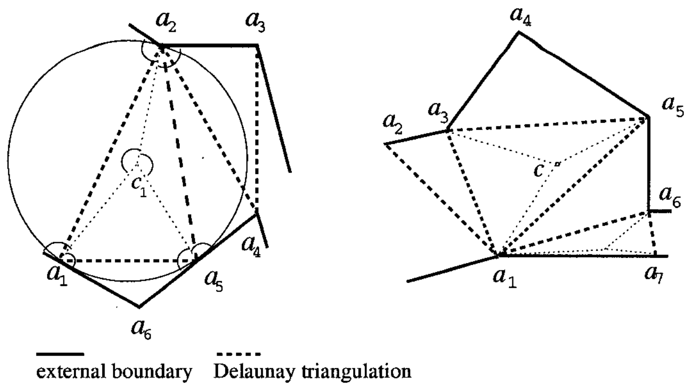


Figure 3.3: Examples of junction and non junction triangles. **Left:** Triangle $a_1a_2a_5$ is a junction triangle, while $a_1a_5a_6$ has two edges on the boundary and $a_2a_4a_5$ does not include its cicrcumcenter, so they are not junction triangles. **Right:** The only junction triangles is $a_1a_3a_5$.

In Fig. 3.3 right, triangle $a_1a_3a_5$ satisfies all the above criteria and is a junction triangle. The other triangles are not junction triangles. $a_1a_2a_3$ and $a_1a_5a_6$ do not include their circumcenter and violate property (1); $a_3a_4a_5$ has two edges on the boundary, violating property (2); $a_1a_6a_7$ does not include a partial separator that has acceptable angles (both angles at a_1 and a_7 are less than the tolerance Φ_0 (for $\Phi_0 = 60°$), so it violates property (3).

The first criterion is set only for the simplicity of the MADD algorithm, in order to avoid negative weights and guarantee that at least two angles between the segments are good. The second criterion prevents a decomposition that will create very small subdomains. The third criterion guarantees the quality of the angles. Let $a_1a_2a_3$ be the vertices of t. Then the third criterion demands the existence of at least one pair of segments a_ica_j, where c is the circumcenter of $a_1a_2a_3$, so that all the angles formed with these segments are greater or equal to Φ_o. Such pairs a_ica_j are called *partial separators* and they will be candidates to form a *complete separator.* A complete separator decomposes a domain into two connected subdomains.

Let $g(\Omega)$ be the number of holes of Ω. The desirable level of refinement Z_Ω satisfies two conditions:

(i) In the Delaunay triangulation $\mathcal{T}$ of Z_Ω there are at least $g(\Omega) + 1$ junction triangles.

(ii) Every segment of the discretization Z_Ω has an empty diametral circle.

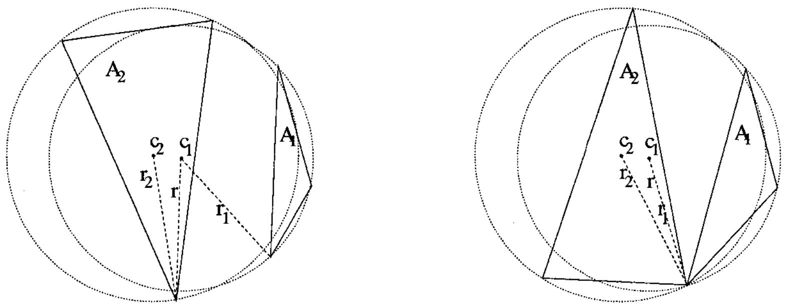


Figure 3.4: On the left, the two Delaunay triangles, A_1, A_2, do not have common vertices. On the right, the triangles share one common vertex; the case of two common vertices is reduced to this.

The first condition requires the existence of at least $g(\Omega) + 1$ junction triangles. We will see in Section 3.6 that this condition is sufficient, although not necessary, for the existence of at least one complete separator. The existence of enough candidate separators depends on the discretization Z_Ω of the domain Ω. A discussion on the level of refinement of Z_Ω is presented in Section 4.1.

The second condition guarantees that all the segments of Z_Ω will appear as edges in $\mathcal{T}$. It also guarantees that all the circumcenters of the triangles of $\mathcal{T}$ are contained in Ω [78]. This in turn guarantees the existence of at least one triangle that includes its circumcenter (Lemma 4).

Lemma 3. *Let A_1, A_2 be two triangles of a Delaunay triangulation, such that the circumcenter c_1 of A_1 is in the triangle A_2 and they don't have the same circumcircle. Let c_2 be the circumcenter of A_2 and r_1, r_2 be the radii of the circumcircles of A_1 and A_2 respectively. Then we have $r_1 < r_2$.*

Proof. Let r be the smaller distance of c_1 from the vertices of A_2, see Figure 3.4. Then $r \geq r_1$. So we have $r_2 > r$, and consequently $r_2 > r_1$. □

Lemma 4. *If all segments in Z_Ω have empty diametral circles, then there is at least one triangle in the Delaunay triangulation $\mathcal{T}$ of Z_Ω that includes its circumcenter.*

Proof. We know that, when the boundary segments have empty diametral circles, all the circumcenters of the triangles of $\mathcal{T}$ are in $\mathcal{T}$ [78]. We assume that the points are in general position, i.e. there are no co-circular points. We will prove the lemma by contradiction.

Suppose that the lemma is not true. Then for every triangle A_i there is another triangle $A_{i+1} \neq A_i$, such that the circumcenter c_i of A_i is included in A_{i+1}. Let r_i be the radius of the circumcircle of A_i. Since we assumed that no triangle includes its circumcenter, the sequence $\langle A_i \rangle$ is infinite. On the other hand the set $\{t_i\}$ of all triangles in Z_Ω is finite, so the sequence $\langle A_i \rangle$ includes an element t_k twice. Then $A_l = A_m = t_k$, for some $l < k$. From the previous lemma we have $r_l < r_{l+1} < ... < r_m$, which contradicts to the fact that r_l and r_m are the radii of the same circle, and thus equal. So the lemma must hold. □

3.3 The MADD Procedure

The MADD algorithm uses as a starting point the approximation of the medial axis by the Delaunay triangulation $\mathcal{T}$, as described in the previous section. The complete separator is formed by partial separators (see Definition 3). The partial separators connect two points of the boundary, since $\mathcal{T}$ is a boundary conforming triangulation. The properties of junction triangles permit the construction of good angles between the partial separators and the external boundary of the geometry. We have developed two MADD algorithms. The first MADD algorithm is described in Section 3.4, and selects a set of partial separators from the junction triangles. The MADD second algorth will allow additionally partial separators to be edges of the Delaunay triangulation and is described in Section 3.5. The selection of the partial separators in both algorithms is based on minimizing the size of the separators, and are guaranteed to form a complete separator.

The MADD algorithm uses as a starting point the approximation of the medial axis by the Delaunay triangulation $\mathcal{T}$, as described in the previous section. Any algorithm that gives a Delaunay boundary conforming triangulation can be used to create it. For our implementation we have used Triangle [77], which is considered to be a state of art Delaunay mesher for planar geometries. The MADD algorithm uses the Delaunay triangulation to identify a set of candidate partial separators. Then it will form a complete separator by a set of partial separators, that will guarantee the decomposition of the domain into two subdomains. The selection of partial separators is based on minimizing the size of the separators, while maintaining the balance of the areas.

The MADD algorithm maps the Delaunay triangulation $\mathcal{T}$ into a graph $G_{\mathcal{T}}$. The information encapsulated in this graph includes: (a) the topology of $\mathcal{T}$, (b) the length of the partial separators, and (c) the area of the subdomains that will be created. This information will be used to : (1) guarantee that the inserted partial separators form a complete separator, (2) minimize the length of separators, and (3) keep the subdomain areas balanced.

After $G_{\mathcal{T}}$ is constructed, the graph is contracted into a graph $G'_{\mathcal{T}}$, so that only the acceptable partial separators are represented in $G'_{\mathcal{T}}$. Then the contracted graph is partitioned, in a way that maintains the balance between the subgraph weights, and minimizes the cut cost. Any of the well known graph partitoning algorithms [46, 4, 41, 42, 44, 88], that decompose a connected graph into two connected subgraphs and satisfy the above criteria can be used. In the cases where the partitioner gives non-connected subgraphs, a connectivity check step must be preformed (see Section 5.1). Finally, the graph partition is translated back into insertions of partial separators, which results a 2-way decomposition. The major steps of the algorithm are:

1. Create a modified graph $G_{\mathcal{T}}$ from the Delaunay triangulation $\mathcal{T}$.

2. Contract $G_{\mathcal{T}}$ into the graph $G'_{\mathcal{T}}$, so that only the candidate partial separators are represented.

3. Partition the graph $G'_{\mathcal{T}}$, optimizing the subgraph weight balanced and the cut-cost.

4. Translate the cuts of the previous partition into the corresponding partial separators and insert them into the geometry.

In Sections 3.4 and 3.5 are described two MADD algorithms. In the first algorithm the graph nodes represent edges of the triangulation, and the contraction procedure is applied on the non-junction triangles. In the second algorithm the graph nodes represent triangles; additionally, edges of the Delaunay triangles that form good angles are allowed as partial separators.

3.4 The MADD First Algorithm

Construction of the Graph $G_{\mathcal{T}}$

In the first MADD algorithm the Delaunay triangulation $\mathcal{T}$ is represented as a weighted graph, which is the dual graph of the edges of the triangles. Two nodes of the graph are adjacent if their corresponding edges belong in the same triangle. The length of the radius of the circumcircle of this triangle will be the weight of the graph edge. The weights of the nodes are set to zero in this step, and they will be computed in the graph contraction step (see Section 3.4).

Figure 3.5 left depicts the step for constructing the graph $G_{\mathcal{T}}$. One graph node is created for each edge of the triangulation, and two nodes are connected if they belong to the same triangle. Let d_{ij} be the node corresponding to the edge $a_i a_j$. The weight of the edge connecting d_{ij}, d_{jk} is the length $|c_l a_j|$, where c_l is the circumcenter of the triangle. For example, the edge that connects d_{12} and d_{25} has weight the length $|c_1 a_2|$. The above procedure is described by the Algorithm 3.1.

```
Algorithm 3.1.
1.  for all the edges $a_i a_j$ in $\mathcal{T}$ do
2.      Add node $d_{ij}$ to the graph $G_{\mathcal{T}}$, with zero weight
3.  endfor
4.  for all triangles $t \in \mathcal{T}$ do
5.      for the three pairs $(a_i a_j, a_j a_k)$ of edges of $t$ do
6.          Create a graph edge between the corresponding nodes $d_{ij}, d_{jk}$,
7.          with weight the length of the circumradius of $t$
8.      endfor
9.  endfor
```

Graph Contraction

In this step the graph $G_{\mathcal{T}}$ produced from the previous step is contracted into a new graph $G'_{\mathcal{T}}$, so that only the edges of junction triangles are represented as nodes in $G'_{\mathcal{T}}$. The nodes of $G_{\mathcal{T}}$ that correspond to edges of non junction triangles of $\mathcal{T}$ are contracted in $G'_{\mathcal{T}}$.

In order to contract the graph $G_{\mathcal{T}}$, first we iterate through all the triangles that are not junction triangles. The nodes of $G_{\mathcal{T}}$ that correspond to the three edges of a non-junction

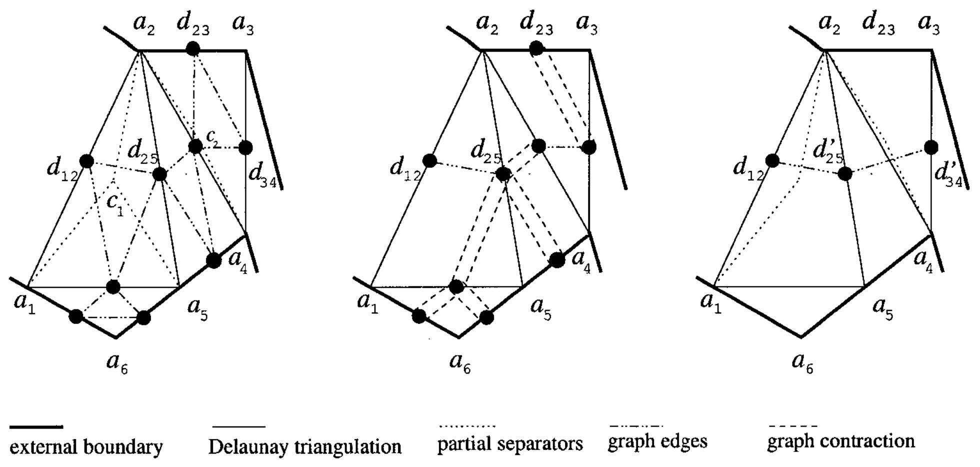


Figure 3.5: An example of creating the MADD graph. **Left** is a part of the Delaunay triangulation and the creation of the corresponding initial graph $G_{\mathcal{T}}$. **Middle**, the procedure of contracting the graph by combining the nodes of $G_{\mathcal{T}}$. The nodes connected by dashed lines are combined. **Right** is the final graph $G'_{\mathcal{D}}$ that corresponds to this part.

triangle are combined into a single node and the new node replaces the initial nodes in the external graph edges, while the edges between the three initial nodes are deleted. The weight of the new node is the sum of the weights of the initial ones, plus the area of the triangle.

The remaining nodes correspond to the edges of junction triangles. Junction triangles contain candidate partial separators, whose number may vary from one to three. From the three possible partial separators we keep the one that forms the greater minimum angle. Since in junction triangles there is at least one partial separator that forms angles no less than Φ_o, the selected partial separator forms angles $\geq \Phi_o$. We establish this partial separator by combining the two of the three nodes that correspond to edges of the triangle. Let $a_1a_2a_3$ be a junction triangle and c its circumcenter. Let d_{ij} be the corresponding node to the edge a_ia_j, then the weight of the node d_{ij} is updated by adding the weight of the area included by the triangle ca_ia_j. Let a_jca_k be the partial separator that forms the greater minimum angle. Then the nodes d_{ji} and d_{ki} are contracted into a single node, where a_i is the remaining vertex of the triangle $a_1a_2a_3$. The procedure is illustrated with the following example.

Example. Figure 3.5 (center) illustrates the procedure of contracting the graph. The bold lines indicate the external boundary. The triangles are part of the boundary conforming Delaunay triangulation of the domain. As above, we denote by d_{ij} the graph node that corresponds to the segment $a_i a_j$. We demonstrate four different cases.

Case I: The triangle $a_1 a_5 a_6$ has two edges on the boundary, so it is not a junction triangle, and the three corresponding nodes are combined to one. The edges connecting the new node d'_{15} are the external ones i.e., the edges that connect d_{15} to d_{12} and d_{15} to d_{25}. The weight of d'_{15} is equal to the area of the triangle $a_1 a_5 a_6$.

Case II: The triangle $a_2 a_4 a_5$ does not include its circumcenter and so it is not a junction triangle. We follow the same procedure as in Case I. The nodes d_{25}, d_{24}, d_{45} are contracted into a new node d'_{25}. The new node has weight the area of the triangle $a_2 a_4 a_5$ and is connected to the nodes $d_{12}, d'_{15}, d_{23}, d_{34}$.

Case III: The triangle $a_1 a_2 a_5$ is a junction triangle. The areas of the triangles formed by its circumcenter c_1 and its corners are added to the weight of the corresponding nodes. For example, the area $|a_2 c_1 a_1|$ is added to the node d_{12}, similarly the areas $|a_2 a_5 c_1|$, and $|a_1 c_1 a_5|$ are added to the nodes d'_{25} d'_{15}, respectively. Suppose that the partial separator $a_1 c_1 a_2$ is the one that that forms the greater minimum angle. Then the nodes d'_{15} and d'_{25} are contracted into a new node d'_{25} with its weight to be equal to the sum weights of the two previous nodes. The graph edge connecting the nodes d'_{15} and d'_{25} is deleted, while the two other graph edges are contracted into one edge connecting d'_{25} to d_{12}; the new edged weight is equal to the sum of the two previous edge weights, which is equal to the length of the partial separator $a_1 c_1 a_2$.

Case IV: The triangle $a_2 a_3 a_4$ is also a junction triangle. As for the previous triangle, first we add the areas of the triangles formed by the circumcenter c_2 and the vertices. The areas $|a_2 c_2 a_4|$, $|a_2 c_2 a_3|$, and $|a_3 c_2 a_4|$ are added to the weight of the nodes d'_{25}, d'_{23}, and d'_{34}, respectively. However, suppose in this case the angle θ, formed by the segment $c_2 a_3$ and the external boundary segment $a_3 b$, is less than Φ_o. Then the two partial separators that include this segment are rejected and we keep the separator $a_2 c_2 a_4$, which is the one that forms the greater minimum angle. The nodes d_{23} and d_{34} are combined to the node d'_{34}. The new node is connected to d'_{25} by an edge with weight equal to the sum of the two previous edge weights, which is the length of the partial separator $a_2c_2a_4$. Figure 3.5 (right) shows the final graph.

Algorithm 3.2.

```
1.   for all non junction triangles $t \in \mathcal{T}$ do
2.       Combine the three nodes that correspond to the edges of $t$,
3.       generating a new node $d'$
4.       Add the area of $t$ to the weight of $d'$
5.   endfor
6.   for all junction triangles $t \in \mathcal{T}$ do
7.       Let $c$ be circumcenter of $t$
8.       for all edges $a_ia_j$ of $t$ do
9.           Add the area of the triangle $a_ica_j$ to the weight
10.          the corresponding node $d_{ij}$
11.      endfor
12.      Find the partial separator $a_ica_j$ in $t$ forming a max min angle
13.      Combine the nodes $d_{ik}$ and $d_{jk}$, where $a_k$ is the remaining vertex
14.  endfor
```

The above procedure is described in Algorithm 3.2.

The Construction of the Separator

After contracting the graph, the constructed graph $G'_{\mathcal{T}}$ is partitioned. The number of the edges of the graph is less or equal to the number of junction triangles, thus the size of the graph partitioning problem is significantly smaller than the element-wise dual graph of the boundary conforming Delaunay triangulation $\mathcal{T}$. Graph partitioning can be very expensive, and reducing the size of the results smaller partitioning time cost.

After partitioning $G'_{\mathcal{T}}$, the final step of the MADD is to construct the separator of the geometry. From the previous step we have a partition of the graph $G'_{\mathcal{T}}$ in two connected subgraphs. This partition will give a corresponding separator for the geometry. Each edge of the graph corresponds to a partial separator of the form a_ica_j, where c is a circumcenter of a junction triangle and a_i, a_j are two of its vertices. For every graph edge that is cut by the partition we will insert the related partial separator in the geometry. In our example above (see Figure 3.6) the partial separator $a_2c_2a_4$ is created in the case that the graph partitioner chooses to cut the edge e_2.

The algorithm traverses the list of all triangles and identifies those triangles whose edges

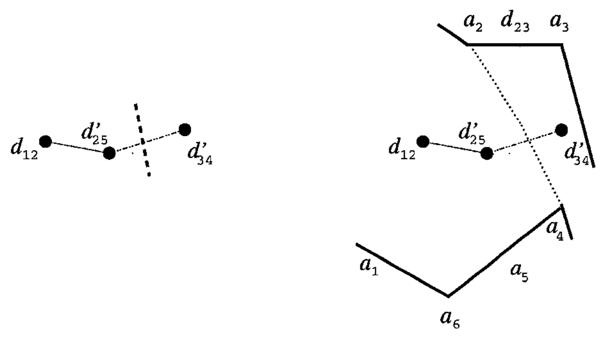


Figure 3.6: A partition of the graph and the corresponding separator, on the right, depicted with dotted lines.

correspond to disconnected nodes after the graph partition. In these triangles the partial separators are inserted, separating the edges that don't belong to the same subgraph. In Figure 3.6 the partial separator $a_2c_2a_4$ separates the edge a_2a_4 from the edges a_2a_3 and a_3a_4. The set of all these inserted partial separators establishes a (complete) separator for the domain, as we will see in Section 3.6. The construction of the separator is described in Algorithm 3.3.

Algorithm 3.3.

```
1.  for all triangles $t \in \mathcal{T}$ do
2.      if one of the edges $a_ia_j$ of $t$ belong to a different
3.      subgraph from the other two edges then
4.          Insert the partial separator $a_ica_j$,
5.          where $c$ is the circumcenter of $t$
6.      endif
7.  endfor
```

The ratio of the cost of the cut to the weight of the subgraphs is translated to the ratio of the total length of the separator to the area of the subdomains. Provided that the graph partitioner gives a good cut cost to subgraph weight ratio, the ratio of length of the separator to the area of the subdomains is also good. This way we obtain separators of relatively small size, and the areas of the subdomains are balanced. Moreover, since all the partial separators, by the construction of $G'_{\mathcal{T}}$, form good angles, the constructed separator forms good angles. In summary, the constructed separator meets the decomposition criteria C1 - C3 in Section 2.

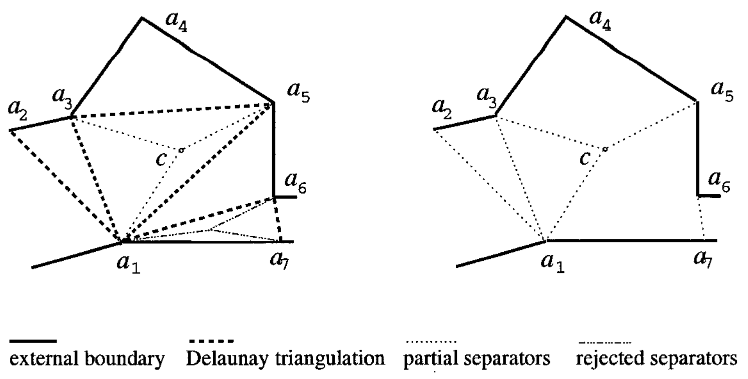


Figure 3.7: **Left** is a part of the Delaunay triangulation and **right** are the partial separators. Triangle $a_1a_3a_5$ is a junction triangle, while the other triangles are not.

3.5 The MADD Second Algorithm

In the second MADD algorithm we have two types of partial separators (see Fig. 3.7): (a) non-boundary edges of the Delaunay triangulation that form angles $\geq \Phi_0$ with the boundary, and (b) segments that connect a circumcenter of a junction triangle with its vertices. For the first type of partial separator we only have to scan the non-boundary edges of the Delaunay triangulation and select the ones that create angles at least equal to our tolerance bound Φ_0. The second type of partial separators are included in junction triangles, as in the first algorithm.

In Fig. 3.7, triangle $a_1a_3a_5$ is a junction triangle, while the other triangles are not junction. The partial separators are either internal Delaunay edges, like a_1a_2, a_1a_3 and a_6a_7, or are formed by connecting the circumcenter of a junction triangle to its vertices. In our example a_1ca_3, a_1ca_5 and a_3ca_5 are the three possible partial separators inside the junction triangle $a_1a_2a_3$. The partial separators always connect two points of the boundary, since $\mathcal{T}$ is a boundary conforming triangulation. The complete separator is formed by choosing a subset of partial separators that will decompose the domain into two connected subdomains.

Construction of the Graph $G_{\mathcal{T}}$

In this step the junction triangles of the Delaunay triangulation $\mathcal{T}$ are divided into three triangles, and the final triangulation is represented as a weighted dual graph. Each of the

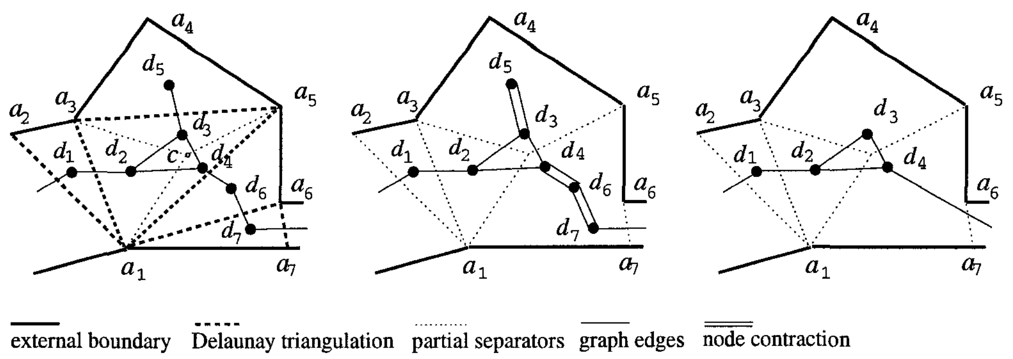


Figure 3.8: An example of creating the MADD graph. **Left** is a part of the Delaunay triangulation and the creation of the corresponding initial graph $G_{\mathcal{T}}$. **Center**, the procedure of contracting the graph by combining the nodes of $G_{\mathcal{T}}$. The nodes connected by doubled lines are combined. **Right** is the final graph $G'_{\mathcal{D}}$ that corresponds to this part.

three triangles included into a junction triangle are represented by three graph nodes. Non-junction triangles are represented by a single graph node. Nodes that represent adjacent triangles are connected by a graph edge. The weight of each node is set equal to the area of the corresponding triangle, while the weight of a graph edge connecting two nodes is set equal to the length of the common triangle edge that is shared by the two corresponding triangles. Algorithm 3.4 describes the graph construction procedure.

Algorithm 3.4.

```
1.  for all the triangles $a_i a_j a_k$ in $\mathcal{T}$ do
2.      if $a_i a_j a_k$ is a junction triangle then
3.          let $c$ be the circumcenter of $a_i a_j a_k$;
4.          create three nodes corresponding to triangles
                $a_i c a_j$, $a_i c a_k$, $a_j c a_k$ with weight equal to their areas;
5.      else
6.          create one node with weight equal to $|a_i a_j a_k|$;
7.      endif
8.  endfor
9.  for all nodes $d \in G_{\mathcal{T}}$ do
10.     find the adjacent triangles and connect the corresponding
            nodes by a graph edge with weight equal to the length of
            their common triangle edge;
11. endfor
```

Fig. 3.8 (left) depicts the step for constructing the graph $G_{\mathcal{T}}$. Triangles $a_1a_2a_3$, $a_1a_5a_6$, $a_3a_4a_5$ and $a_1a_6a_7$ are not junctions, and each is represented by one node, d_1, d_6, d_5, and d_7

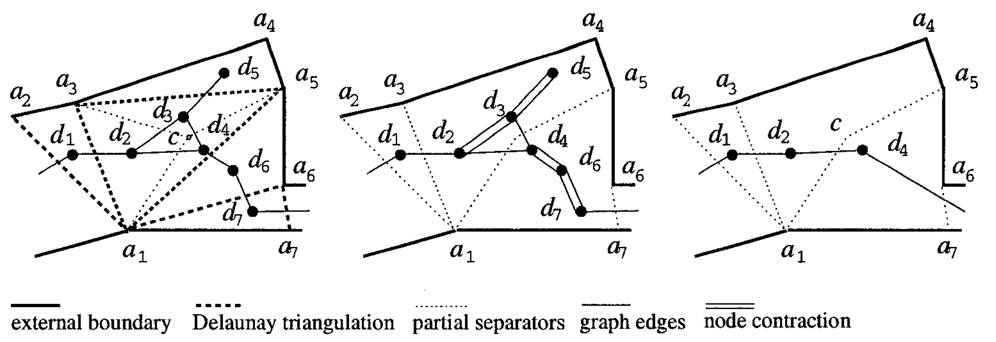


Figure 3.9: An example of contraction of the nodes inside of a junction triangle. **Left** is a part of the Delaunay triangulation and the creation of the corresponding initial graph $G_{\mathcal{T}}$. **Center**, the procedure of contracting the graph, in this case the two nodes of the junction triangle $a_1a_3a_5$ are combined. **Right** is the final graph $G'_{\mathcal{D}}$ and the corresponding candidate partial separators.

respectively. Triangle $a_1a_3a_5$ is a junction triangle and is divided in three triangles: a_1ca_3, a_1ca_5 and a_3ca_5, where c is the circumcenter of $a_1a_3a_5$. These triangles are represented by the nodes d_2, d_4, and d_3 respectively. The weight of each node is equal to the area of the corresponding triangle. For example, the node d_2 has weight equal to the area $|a_1ca_3|$. Nodes that represent adjacent triangles are connected by a graph edge, with weight equal to the length of their common triangle edge. For example, the nodes d_1 and d_2 are connected by a graph edge with weight equal to the length $|a_1a_3|$, while the nodes for d_2 and d_3 are connected by a graph edge with weight equal to $|ca_3|$. The above procedure is described by Algorithm 3.4.

Graph Contraction

In this step the graph $G_{\mathcal{T}}$ produced from the previous step is contracted into a new graph $G'_{\mathcal{T}}$, so that only the acceptable partial separators are represented as edges in $G'_{\mathcal{T}}$. In order to contract the graph $G_{\mathcal{T}}$ we iterate through all the graph edges and eliminate those that correspond to not acceptable triangle edges. A triangle edge is not acceptable if at least one of the angles that it creates is less than Φ_0. The graph edge that corresponds to non-acceptable triangle edges is deleted, and the two graph nodes that were connected by the eliminated edge are combined into one node; the new node represents the total area of the triangles represented by the contracted nodes.

Fig. 3.8 (center) illustrates the procedure of contracting the graph. The triangle edge a_3a_5 forms small angles with the boundary and is not acceptable. The corresponding graph edge d_3d_5 is eliminated, while the nodes d_3 and d_5 are combined into a new node. The new node represents the polygon $a_3ca_5a_4$ and its weight is equal to the polygon area, which is the sum of the two previous areas. The new node also inherits all the external graph edges of the two previous nodes, which in this case are the two edges d_3d_4 and d_2d_3. The same procedure is followed for eliminating the edges d_4d_6 and d_6d_7. In Fig. 3.8 right the final graph $G'_{\mathcal{T}}$ is depicted with the corresponding areas and partial separators.

In Fig. 3.9 we have a slightly different geometry, which depicts the elimination of an internal edge of a junction triangle. The triangle edge ca_3 forms a small angle with the boundary, so it is not acceptable and it is eliminated. The two nodes d_2 and d_3 in the junction triangle $a_1a_3a_5$, which are separated by this edge, are combined into a new node. The new node inherits two graph edges connecting it to the same node d_4. These two edges have a total weight equal to the length of the partial separator a_1ca_5. The above procedure is described by Algorithm 3.5.

Algorithm 3.5.

```
1.  for all edges $d_id_j \in G_{\mathcal{T}}$ do
2.      if the corresponding triangle edge
            forms an angle $< \Phi_0$ then
3.          delete the edge $d_id_j$;
4.          create a new node $d$ with weight equal to the
                sum of the weights of the nodes $d_i$, $d_j$;
5.          transfer all the external graph edges of
                $d_i$ and $d_j$ to the new node $d$;
6.      endif
7.  endfor
```

The Construction of the Separator

The result of the previous step is a graph $G'_{\mathcal{T}}$, whose edges represent the partial separators that can be used to decompose the domain. The next step is to partition the graph in two connected subgraphs and translate this partition into a geometric domain decomposition. The weights of the nodes of $G'_{\mathcal{T}}$ represent the size of the corresponding areas, while the

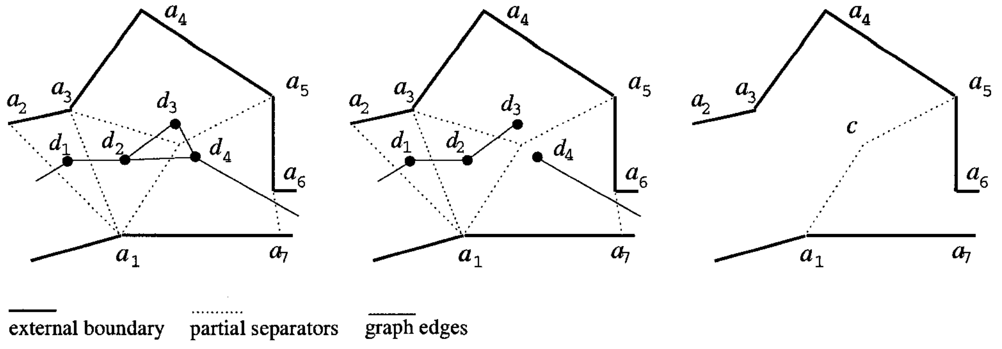


Figure 3.10: **Left** is depicted the graph G'_T with the corresponding areas. **Center** The graph is partitioned by deleting two graph edges. **Right** The corresponding partial separator is inserted to the geometry.

weights of the edges represent the length of the corresponding partial separators. The objective of the graph partitioner is to minimize the ratio of the cut-cost to the subgraph weight. The graph contraction step reduces significantly the size of the graph, resulting a smaller partitioning time cost.

After partitioning the graph G'_T into two connected subgraphs, the final step is to construct the separator of the geometry, by translating the graph edge cuts to insertions of partial separators. The partial separators, that correspond to edges cut by the graph partitioner, are inserted into the geometry. In Fig. 3.10 (left) the graph G'_T is depicted, the graph partition cuts of the two edges d_2d_3 and d_2d_4 (middle), and the corresponding partial separator a_1ca_5 is inserted to the geometry (right). The construction of the separator is described in Algorithm 3.6.

```
Algorithm 3.6.
1.  for all the edges $d_id_j \in G'_T$ do
2.      if $d_i$ and $d_j$ belong to different subgraphs then
3.          insert the partial separator, corresponding to $d_id_j$,
                into the geometry;
4.      endif
5.  endfor
```

If the graph G'_T has at least two nodes, then a 2-way partition exists and it will give a decomposition of the domain into two subdomains Provided that the graph partitioner gives a small cut cost and balanced subgraph weights, the length of the separator will be relatively small and the areas of the subdomains will be approximately equal. Moreover, since all the partial separators, by the construction of G'_T, form good angles, the constructed separator will also form good angles. Thus, the constructed separator meets the decomposition criteria C1 - C3 described in Section 2.

3.6 Proof of Correctness

In this subsection we prove that the MADD algorithm decomposes the domain Ω in two connected subdomains. We remind that the domain Ω is the closure of an open connected bounded set and the boundary $\partial\Omega$ is a PSLG formed by a set of linear segments which do not intersect. A separator $\mathcal{H} \subseteq \Omega$ is a finite set of simple paths (a continuous 1-1 map $h : [0,1] \to \Omega$) that do not intersect and define a decomposition A_1, A_2 of Ω in the following way: A_1 and A_2 are connected sets, with $A_1 \cup A_2 = \Omega$, and $U \cap \mathcal{H} \neq \emptyset$ for every path $U \subset \Omega$ which connects a point of A_1 to a point of A_2.

Lemma 5. *Let $g(\Omega)$ be the genus (number of holes) of Ω and n the number of junction triangles. If $n > m$, then there is a separator for Ω formed by partial separators.*

Proof. We will prove the lemma by induction on $g(\Omega)$. If $g(\Omega) = 0$, then $n \geq 1$, and there is at least one partial separator. In this case, every partial separator is a separator for Ω, since every simple path $f : [a,b] \to \Omega$, with $f(a), f(b) \in \partial\Omega$ and $f(a,b) \subset \Omega^o$, is a separator for Ω.

Suppose the lemma is true for $g(\Omega) = q$, we will prove it is true for $g(\Omega) = q + 1$. We have that $n > q + 1$. If for a partial separator acb, where $a, b \in \partial\Omega$, we have that both a, b don't belong to the boundary of a hole, then acb forms a separator, as in the case $g(\Omega) = 0$. In the case that one of the points a, b belong to the boundary of a hole O, then by inserting the partial separator acb we eliminate O. The new domain has q holes and $n - 1 > q$ junction triangles. Thus, by the inductive hypothesis, it can be decomposed by partial separators. Therefore there is a separator formed by partial separators, when the conditions of the lemma are satisfied. □

Theorem 6. *Let $g(\Omega)$ be the genus Ω and n the number of junction triangles. If $n > g(\Omega)$, then the MADD algorithm decomposes Ω in two subdomains.*

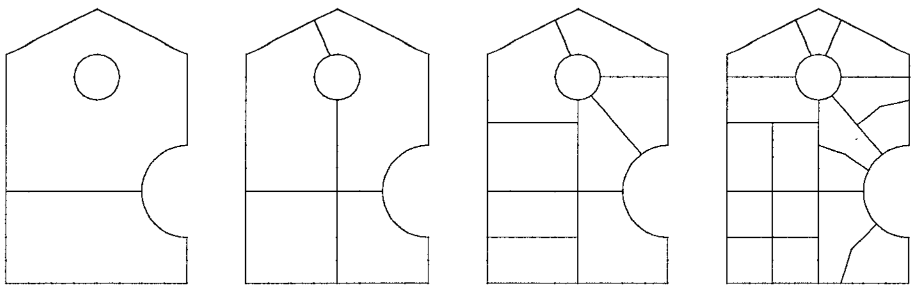


Figure 3.11: N-way partitions, where $N = 2, 4, 8, 16$, by the MADD divide and conquer method.

Proof. Let $e_i, i = 1, ..., n$ be the edges of the contracted graph $G'_\mathcal{T}$ created by MADD. Each of these edges corresponds to a partial separator $h_i, i = 1, ..., n$. We will show that every decomposition of the graph $G'_\mathcal{T}$ corresponds to a decomposition of Ω formed by partial separators, and vice versa.

Let $E = \{e_i, i \in I\}$ be the set of edges that the graph partitioner cuts, creating two subgraphs G_1, G_2. Let $\mathcal{H} = \{h_i, i \in I\}$ be the set of partial separators that are correspond to these edges. Finally, let $A_1, A_2 \subset \Omega$ be the two corresponding areas to the subgraphs G_1, G_2. Obviously $A_1 \cup A_2 = \Omega$. From the construction of the graph we have that the connected subgraphs correspond to path connected areas of Ω. Assuming that the graph partitioner decomposes $G'_\mathcal{T}$ in two connected subgraphs, then G_1, G_2 are connected, and so A_1, A_2 are also connected. Every path $U \subset \Omega$ from a point of A_1 to a point of A_2 corresponds to a path U' in $G'_\mathcal{T}$ form a node of G_1 to a node of G_2. Since the edges E decompose G_1 from G_2, we have $U' \cap E \neq \emptyset$. Let $e_j \in U' \cap E$. Then we have $U \cap h_j \neq \emptyset$, and the path U intersects $\mathcal{H}$. Thus $\mathcal{H}$ is a separator for Ω. Working backwards we see that a separator for Ω corresponds to a partition of the graph. The existence of such a separator is proved in Lemma 5, and this completes the proof. □

3.7 N-way Decomposition

So far we have described the MADD procedure for a 2-way decomposition. In order to create more than two subdomains we apply the MADD procedure following the divide and conquer paradigm (see Figure 3.11). The created subdomain are further decomposed by applying the MADD independently. The resulting decomposition shows good adaptivity to the geometry and also the divide and conquer approach lends itself to a parallel domain decomposition procedure, as the one described in Section 4.3

The N-way decomposition can be controlled by user defined criteria as to which subdomains should be decomposed. For example, a maximum area criterion will result subdomains with area less than a given bound. In section 4.4 we examine decomposing criteria that produce graded decompositions, and in Section 7.3 we describe a gradation cotrolled N-way domain decomposition.

Chapter 4

MADD Enhancements

4.1 Static and Dynamic MADD

The existence and the quality of a complete separator depends on the number and quality of the partial separators, which in turn depends on the level of the discretization of the boundary segments, and also on the geometry of the domain. There are three parameters that effect the level of required discretization: (1) the number of subdomains we want to create, (2) the characteristics of the initial geometry, and (3) the lower angle bound Φ_0. It is hard to define what would be a difficult geometry to decompose, since geometries that look complicated may form areas where "natural" cuts can be made, while geometries that look simple may lack these natural cuts.

Estimating the level of the refinement, that would give an optimal decomposition, is a difficult problem. Increasing the refinement will result a better approximation of the medial axis, and more – and better in terms of the C1-C3 criteria – partial separators. However, over-refinement creates a number of problems. First, it increases the time for decomposing the geometry, since the time for creating the Delaunay triangulation, and also for the MADD procedure, depends on the number of input points (see the experimental results in Section 5.2). Second, it could result into arithmetic rounding errors when calculating geometric entities, like circumcenters and angles. We implemented two approaches for the refining problem. The first is a static approach, where the refining is predetermined, and the second is a dynamic approach, where the refining is an adaptive procedure.

Static MADD. During the static MADD refining procedure the refining size of boundary and separator edges is precomputed. The level of refinement is based on a user-defined uniform refinement factor r, the average boundary segment size L_a, and the $\sqrt{N}$, where N is the number of subdomains. The square root of the number of subdomains was chosen in a heuristic way, based on the fact that the square of the lengths of the separators is analogous to the areas of the subdomains. The average area of subdomain is A/N, where A is the total area and N the number of subdomains. So, the separator lengths will be proportional to $\sqrt{1/N}$, and consequently the level of refinement should be analogous to $\sqrt{N}$. The average boundary segment size L_a is used to produce a close to uniform discretization, by breaking the initial segments into subsegments of length L_a/r, where r is a user-defined factor. This factor is chosen to reflect the lower angle bound in relation to the geometry we want to decompose.

Dynamic MADD. In the dynamic MADD approach each subdomain is refined individually and adaptively. The refinement is not permanent and is performed locally on the subdomain we want to decompose. First an initial refining is applied, as described in the static approach. If the decomposition procedure fails to find separators, the refining is recalculated, in a geometric increasing level. This means that the refining factor r takes gradually the values r, $2r$, $4r$, and so on, until the decomposition with the given conditions is achieved. After successfully decomposing a subdomain, the refining is discarded, allowing a next adaptive refining procedure to take place.

This adaptive approach allows large decompositions to be created (we have created decompositions of the order of 50,000 subdomains), and at the same time is efficient. The refining procedure is fast, and most of the subdomains will be decomposed in the first step, using minimum refinement, and thus are decomposed fast. The subdomains that are harder to decompose will go into the next levels of refinement. As the decomposition progresses, the created subdomains tend to have simple shapes, and thus require small refinement. The subdomains that require more than two refining iterations are a small precentage of the total number of subdomains (usually less than 1%), and thus they result a small adaptivity performance cost.

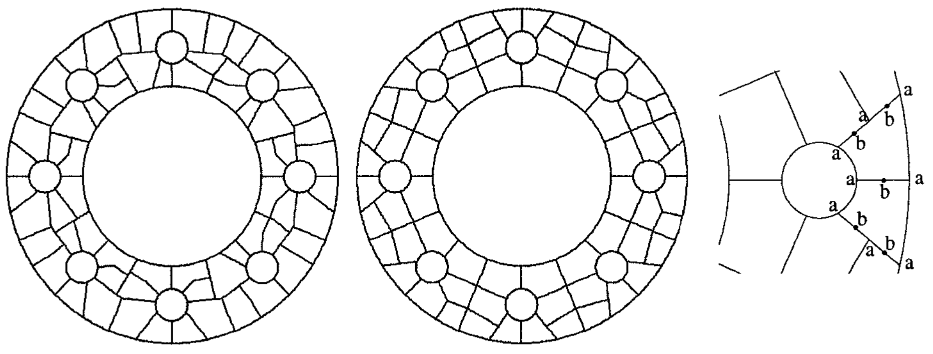


Figure 4.1: The Pipe domain decomposed in 64 subdomains using the MADD algorithm. On the **left** no smoothing is used. Most of the separators don't meet at their end-points, and they create small segments on their common boundaries. On the **middle** the smoothing procedure is used, giving conforming separators. **Right**, the points of the first type (a) and second type (b) are depicted.

4.2 Separator Smoothing Procedure

The independent computation of the separators might create small segments along their common boundary (see Fig. 4.1, left). The size of these segments depends on the level of the refinement of the extrenal and internal boundaries. As we increase the number of subsegments (thus decreasing the size of them), we also increase the probability of creating these small segments. On the other hand, the graph partitioner has information only about the size of the separators, and not about their quality, i.e., the angles that they form. Although all the permissible separators form angles greater than a predefined lower bound Φ_0, we would like to choose the ones that are not only small, but also form the best possible angles (close to $\pi/2$). In order to deal with these two issues we introduce a smoothing procedure that improves the quality of the decomposition.

The smoothing procedure is performed in two steps. The first step takes place during the construction of the graph $G_{\mathcal{T}}$. In this step we incorporate into the weight of the graph edges two types of additional information: (a) the quality of the angles that the corresponding separators form, and (b) the conformity with existing separators (i.e. if the separator's end-points meet at the end-points of an existing separator). The weight of each graph edge is multiplied by a coefficient f_0, which reflects the quality of the minimum angle ϕ that the corresponding separator forms. This coefficient is computed as $f_0 = \frac{1}{\phi - \Phi_0 + 1}$, for $\phi \leq \pi/2$, and $\frac{1}{\pi/2 - \Phi_0 + 1}$, for $\phi \geq \pi/2$. The coefficient f_0 takes values from $\frac{1}{\pi/2 - \Phi_0 + 1}$, when $\phi \geq \pi/2$, up to 1, when the minimum angle is equal to the minimum acceptable bound $\phi = \Phi_0$. So, the weight of the graph edge is decreased proportionally to the quality of the minimum angle.

We would also like to encourage the graph partitioner to choose separators that conform with existing separators, i.e., that meet on the common boundary with the existing partial separators of the adjacent subdomains. To this end we identify two types of boundary points (see Fig. 4.1, right). Points of the first type are either initial points of the domain boundary, or are end-points of an existing separator. In order to encourage the graph partitioner to choose conforming separators, we decrease the weight of the graph edges when these correspond to separators defined from points of the first type. These are end-points of existing separators (or of the initial boundary), and the new separators that meet at these points are conforming with the existing separators. The second type of points are the middle points of segments defined by the first type points. We also reduce the weight of the graph edges corresponding to separators defined from second type points. In this way we increase the probability that a separator will be chosen that has end-points either on existing end-points (first type points), or away from them (second type points).

The previous step awards conforming separators, and the ones that form better angles, but it does not guarantee that these will be chosen by the graph partitioner. In order to improve further the quality of the separator we introduce a second smoothing step, an ad hoc heuristic, after the graph partitioning procedure. Instead of inserting the partial separators chosen by the graph partitioner, we examine all the possible separators that are close to the initial ones, and insert the optimal, according to an optimality function. The neighboring separators are defined by the neighboring points to the end-points of the initial separator. The optimality function computes the degree of quality based on : (a) the size of the separator, (b) the minimum angle that it forms, and (c) the type of its end-points. The value f_0 of the angle quality is computed as described above. The value f_1 reflects the conformity and is set to 0.5 for including first type points, 0.85 for including second type points, and 1.0 otherwise. The normalized separator length is represented by f_2. Finally, the imbalance, measured as the bigest subdomain area over the total area, is represented by f_3. Minimizing the four values f_0, f_1, f_2, f_3 is an multiobjective optimization problem. A common way to explore the solutions of multiobjective optimization problems is by employing the notion of the Pareto surfaces (c.f. [25]). This approach provides a set of solutions where a tradeof between the objectives occurs. An optimal balance between the two subdomains is not a strict requirement, because the over-decomposition of the domain, and also the divide and conquer way we create the subdomains. So we are satisfied to keep the value f_3 less than a bound[1] $L_b < 1$. Then we select through a local search the partial separator that minimizes the product $f_0 f_1 f_2$.

The smoothing procedure, almost always, gives conforming separators that form good angles. This depends though on the initial partition of the graph, the balance of the decomposition, and of course, the geometric characteristics of the domain.

4.3 Parallel MADD

The divide and conquer approach we use for decomposing the domain provides the way to parallelize the MADD procedure. In the case of static decomposition with no smoothing, each subdomain is decomposed independently. In this case it is straight forward to parallelize the MADD procedure. When we use the dynamic approach and also the smoothing procedure, then additional information must be communicated between neighboring subdomains. We have implemented a parallel MADD (PMADD) for the first case, where no communication is needed. The second case is still parallelizable, but it requires additional communication procedures.

Processors	1	8	16	32	48	64
Key domain						
Subdomains	12	96	192	384	576	768
PMADD (secs)	0.20	0.37	0.44	0.60	0.83	1.05
Time/subdomain	0.017	0.004	0.0023	0.0016	0.0014	0.0014
Pipe domain						
Subdomains	16	128	256	512	768	1024
PMADD (secs)	0.27	0.51	0.60	0.89	1.07	1.47
Time/subdomain	0.016	0.004	0.0023	0.0017	0.0014	0.0014

Table 4.1: Performance results for the parallel MADD for the Key and the Pipe geometries.

[1]The default value for L_b is set to 0.75.

The PMADD method is implemented using a master/worker model. Let P be the number of processors, and N the number of subdomains we want to create. Processor 0 is used as the master processor, while all the processors, including processor 0, are used as worker processors. The master processor maintains a sorted list of the areas assigned to each processor. In each iteration of the PMADD procedure a decomposition request is sent from the master processor to the processors assigned with larger total areas. The processors that receive such requests decompose their larger subdomain in two subdomains using MADD. One of the two created subdomains is sent to a processor with small total assigned area. The procedure is repeated until all N subdomains are created.

During the PMADD phase, the first P subdomains are created in $\log(P)$ iterations. The total number of iterations for the parallel MADD phase is

$$\frac{N-P}{P/2} + \log(P) = 2(M-1) + \log(P),$$

where $M = \frac{N}{P}$ is the average number of the final subdomains per processor. Typical values for M in our experiments vary between 12 and 20. Each iteration is using on average $\frac{N-1}{2(M-1)+\lg(P)} = \frac{PM-1}{2(M-1)+\log(P)}$ processors, requiring communication volume of the same order. The experimental results indicate a small slope linear time, as Table 4.1 shows.

4.4 N-way Graded Decomposition

The N-way decomposition procedure we have described so far produces uniform domain decompositions, i.e. the areas of the subdomains are approximately equal. This approach is well suited for uniform mesh generation, but in many cases we would like to have a graded, locally refined, mesh. In these cases a uniform decomposition will result imbalance during the parallel mesh generation, and also during the parallel FD/FEM procedure. In this section we describe a procedure that produces graded domain decompositions using the MADD algorithm.

N-way graded domain decompositions can be produced in a similar way as the non-graded ones, by recursively applying the MADD procedure. The only step that needs to be modified is the way we choose the subdomain to be decomposed. In the uniform case, the

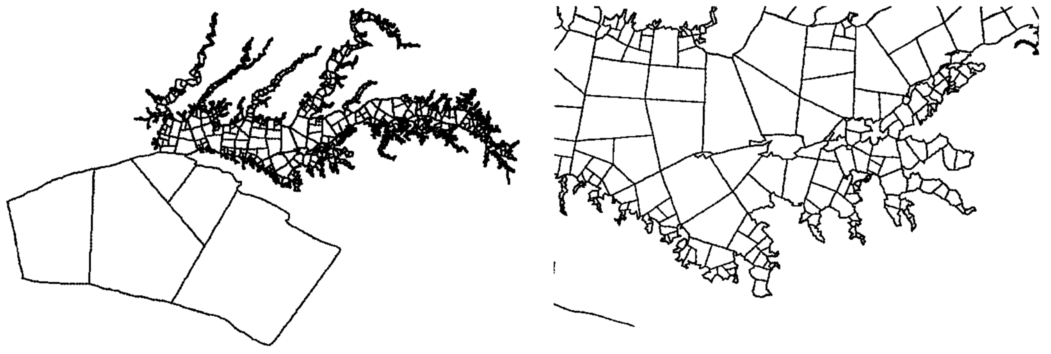


Figure 4.2: Graded MADD based on boundary weights. **Left**, a model of the Chesapeake bay is decomposed in 1250 subdomains, with weights on all the boundary points and interpolation factor set to zero. **Right**, detail of the decomposition, the irregular inner polygons represent islands and are part of the initial domain.

subdomain with the larger area is decomposed, while in the graded case the subdomains are "weighted", and the subdomain with the largest weight is decomposed. There are two ways to define the "weight" of the subdomains. The first is to define an area bound for each subdomain. The second is to assign a relative density weight for each subdomain, and use it as a gradation criterion. In the first case, the subdomain with the greater area to area bound ratio will be decomposed, and no subdomain with area ratio greater than a user-defined bound will be in the final decomposition. In the case of using a density weight criterion, the subdomain with the greatest density weight is chosen to be decomposed, and the parts of the geometry with greater density weights will be decomposed more intensively. The subdomain weight is computed as the sum of a uniform weight, reflecting the area of the subdomain, and the graded weight, reflecting the assigned weight to the subdomain. The formula is

$$\text{weight} = u \times \text{subdomain\_area} + a \times \text{subdomain\_weight},$$

where u and a are user defined weight factors.

While the subdomain area bound approach is a natural extension of the existing approaches for defining the element size of the mesh, it does not allow the user to predefine the number of subdomains he wants to create. The number of subdomains depends not only on the expected size of the mesh, which can be estimated through an area criterion, but also by the number of processors that we want to utilize and the available memory. Using density weights allows us to produce graded decompositions and at the same time to predefine the number of subdomains that will be created.

In our implementation the decomposition procedure can be controlled in the following three ways:

1. Using density weights on the boundary points.

2. Using density-weight or required-area values over an unstructured background mesh.

3. Using a density-weight function or a required-area function over a structured background grid.

Case (1) The use of density weights on the boundary points is the simplest case, and can be viewed a sub-case of the case (2). We describe it separately because it is simple to define, and in some cases (like crack propagation) we need a better refinement near the boundary. The weights assigned to the boundary are defined in the PSLG file that describes the geometry. Each point, in addition to its coordinates, is assigned an integer density weight value. A value of zero means that the point will not contribute to the density. Each subdomain is assigned a density weight value, which is the sum of its boundary weights. An interpolation factor allows the user to define the weights of the created internal boundaries; we use a linear interpolation procedure. An interpolation factor of zero will assign zero weights to the interfaces. Examples of this approach are depicted in Fig. 4.2.

Case (2) In this case we use a density-weight or required-area background mesh. A set of points in the interior, or on the boundary, of the geometry is assigned either with density weights, which indicate the required level of refinement at the neighborhood of these points, or with required area values, which indicate the area of the subdomain including this point. The points typically would be vertices of a previous mesh (see Fig 4.4, left). The density weight of each subdomain is computed as the sum of the weights of the points included in the subdomain. An example of this approach is depicted in Fig. 4.3, which is a model used to study the incompressible turbulent flow past a circular cylinder [33], and in Fig. 4.4. The

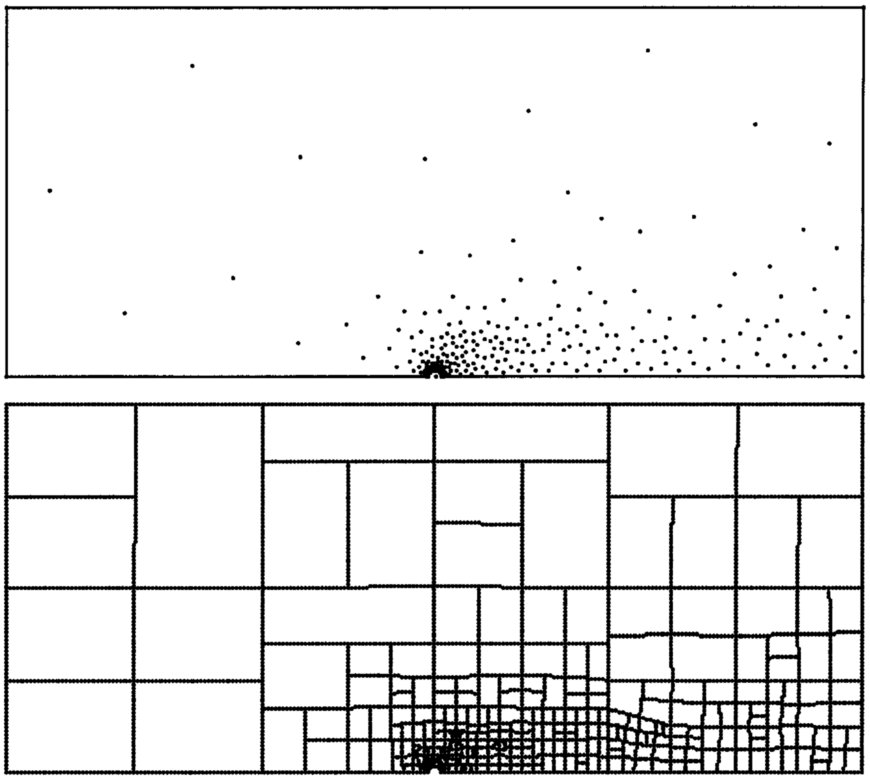


Figure 4.3: Graded MADD based on weighted background mesh. **Top** is the weight background mesh vertices of the Cylinder domain, and **bottom** is the corresponding decomposition in 280 subdomains.

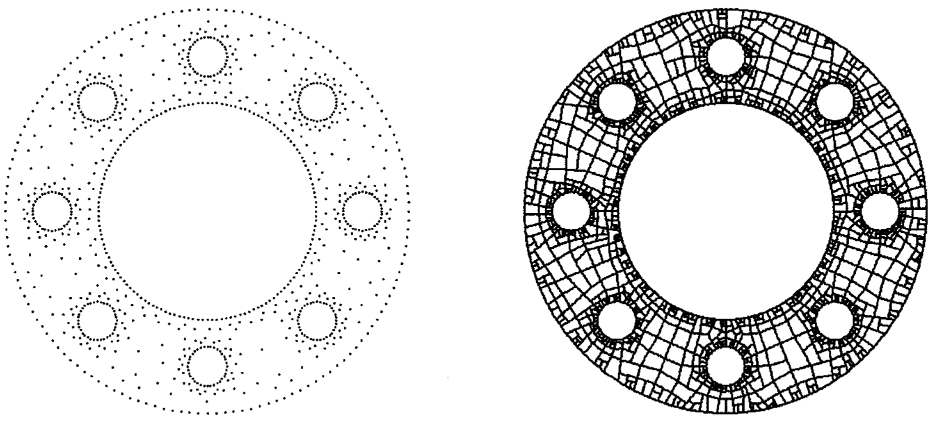


Figure 4.4: Graded MADD based on weighted background mesh. **Left** is the weight background mesh vertices of the Pipe domain, and **right** is the corresponding decomposition into 1250 subdomains.

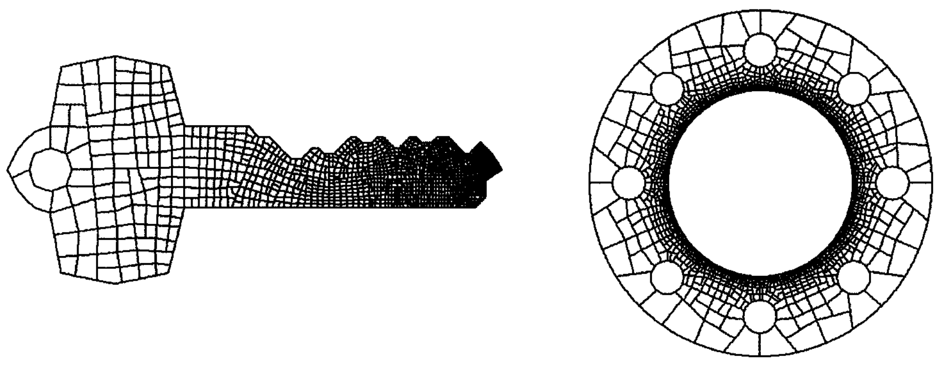


Figure 4.5: **Left**, the Key is decomposed in 1250 subdomains using a linear weight function, proportional to x coordinate. **Right**, the Pipe decomposed in 1315 subdomains using an area function proportional to ρ^{12}, where ρ is the distance from the center of the inner circle.

size of the background mesh should be proportional to the number subdomains we want to create. Creating a large number of subdomains using few background points will result poor quality of the subdomain gradation, with much larger subdomains adjacent to small ones. This will increase the subdomain connectivity and the cost for the start-up in the communication of the FEM solver. On the other hand, too many background points will unnecessarily slow the procedure, without improving the quality of the gradation.

Case (3) In this case we use a density weight function, or a required area function, to control the gradation of the decomposition. These functions are evaluated over a structured gird created on the fly during the decomposition procedure. The density-weight function assigns a weight to each point of the created background mesh, and, as in case (2), the density weight of each subdomain is computed as the sum of these weights. An example of this approach is depicted in Fig. 4.5 (left).

The required-area function assigns to each point the maximum subdomain area that is expected for the subdomain that includes this point. The required-area for a subdomain is computed as the minimum of the required-area function values of all the mesh points contained in the subdomain. In each step the subdomain with the highest ratio of area over required area is chosen to be decomposed. The procedure is repeated, until no ratio is greater than a user-defined bound (default is 1), or until a maximum number of subdomains is reached. An example of this approach is depicted in Fig. 4.5 (right).

Chapter 5

MADD Implementation and Experimental Results

5.1 Implementation

The programming language for our implementation is C. The Triangle library ([86, 77]) was used for the creation of the Delaunay triangulation during the MADD procedure. The Metis library ([57, 45]) was used for the graph partitioning step in the MADD procedure. Metis does not always produce connected components, while the MADD method requires a graph partition into two connected subgraphs. A routine was implemented that identifies these cases and restores the connectivity. There are also cases where the graph partition will result the insertion of two partial separators that meet in the same boundary point. The angle formed between these two separators might be less than the bound Φ_0, giving a non-acceptable decomposition. We have added a procedure that checks for these cases, modifies and repartitions the graph, so that only angles $\geq \Phi_0$ are created during the insertion of separators. In general these cases correspond to high cut costs, due to the length of the two intersecting separators, and in our experiments they rarely occurred.

5.2 Experimental Results

For our experiments we used three model domains. The *Pipe* model is an approximation of a cross section of a regenerative cooled pipe geometry. It consists of 576 boundary segments and 9 holes. The *Key* is a domain provided with Triangle [77], and has 54 boundary segments and 1 hole. The *Chesapeake bay* (Cbay) model defined from 13,524 points and it

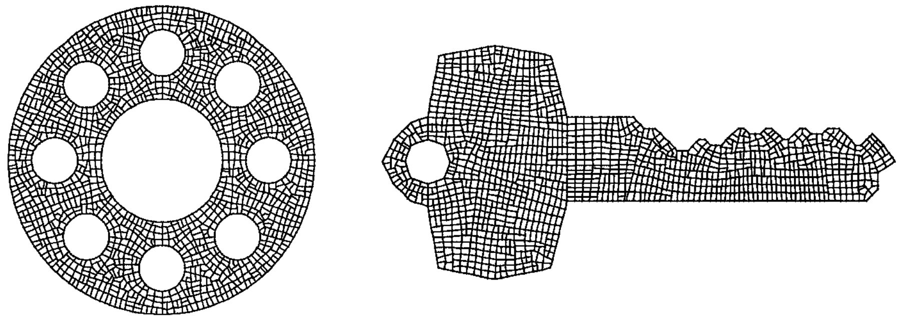


Figure 5.1: On the left the *Pipe* domain is depicted, and right the *Key* domain. Both are decomposed uniformly into 1250 subdomains by the MADD procedure.

has 26 islands.

We ran three sets of sequential experiments. In the first set we produced uniform decompositions for the three test domains using the static MADD. For the second set of experiments we used the dynamic MADD on the Key domain, and we we compare the results to the ones obtained by Metis, which is a state of the art graph partitioner. In the third set of experiments we assess the performance of the graded MADD. The experiments were performed on a Dual Pentium 3.4GHz processor.

Static MADD. In the first set of experiments we used the static MADD with a lower angle bound of 60°. The results show that the time to decompose a domain is directly related to the size of the domain (measured in number of segments), and the level of the refinement we apply on it (see Figs. 5.2 -5.3). The problem size for all the major routines (Delaunay triangulation, graph creation and partition) is proportional to the number of the input segments, and thus we should expect this behavior. The level of refinement is analogous to $\sqrt{N}$, where N are the number of subdomains. The refinement level, and the decomposition times, for the Pipe and the Chesapeake bay tend to reflect this “square root” behavior. This is not the case for the Key, which has few initial segments, and requires more intense refinement in order to get good decompositions.

Dynamic MADD. For the second set of experiments we partitioned the Key geometry up to 2,000 subdomains uniformly, using the dynamic MADD, and we compare the results to those obtained by Metis, which a state-of-art partitioner often used for parallel mesh gen-

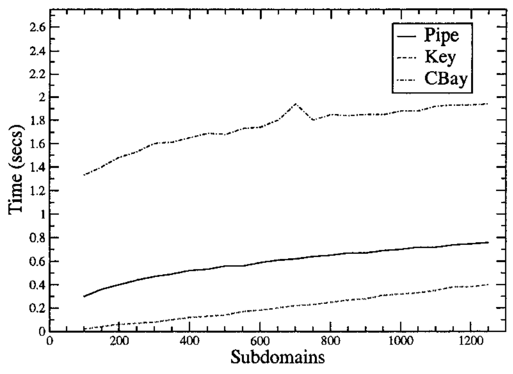


Figure 5.2: Decomposition times for the uniform static MADD.

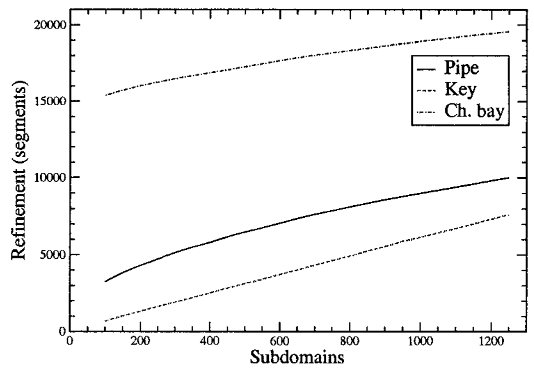


Figure 5.3: The refinement (number of segments) for the uniform static MADD

eration. A background Delaunay mesh, of size approximately 120 triangles per subdomain, was used for the Metis decompositions. The Delaunay mesh generation procedure is the only one that provides quality guarantees, creating angles no less than 30°. The background mesh was translated into a weighted graph, with weights reflecting the edge lengths and the triangle areas, and then Metis was called to partition the graph. The dynamic MADD implements the adaptive local refinement procedure described in Section 4.1. The lower angle bound was set to 70°.

Figure 5.4 depicts the minimum, median and 90% quantiles of the angles created by MADD. As expected, the minimum angles are no less than 70°, while most of the angles are close to 90°. In comparison, Metis gives minimum angles as small as the ones in the background mesh (see Fig. 5.5). The efficiency of the MADD depends on the geometry (Fig. 5.2), while the efficiency of Metis depends on the size of the background mesh. For the Key geometry MADD performs better (see Fig. 5.6), for the Pipe the decomposition times had small differences, while for the Cbay domain Metis performed better. The average length of the separators per subdomain is almost the same (Fig. 5.7), with MADD being slightly better. The maximum ratio of the subdomain separator length to the subdomain area is the same for the two methods, see Fig. 5.8. The maximum subdomain area is close to the average subdomain area for the MADD method (Fig 5.9), while Metis results almost perfect maximum subdomain area due to the near perfect balancing that it produces.

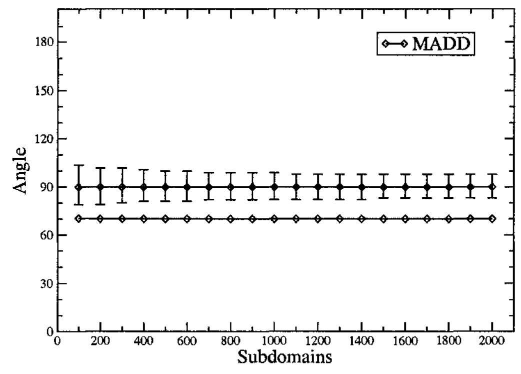


Figure 5.4: The angles created by MADD. The minimum, median and 90% quantiles are depicted.

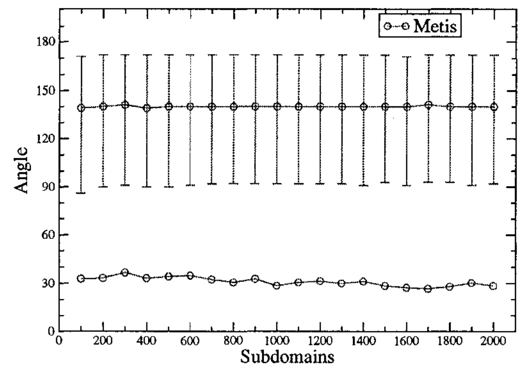


Figure 5.5: The angles created by Metis. The minimum, median and 90% quantiles are depicted.

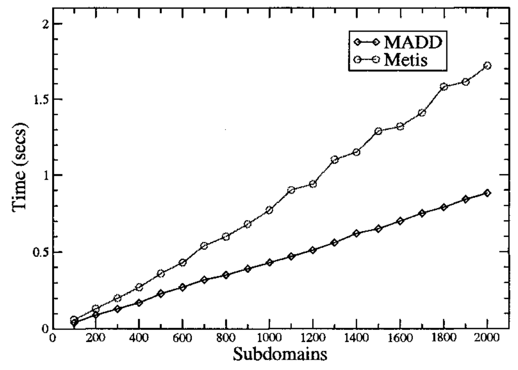


Figure 5.6: The decomposition times for MADD and Metis. The mesh generation time is included.

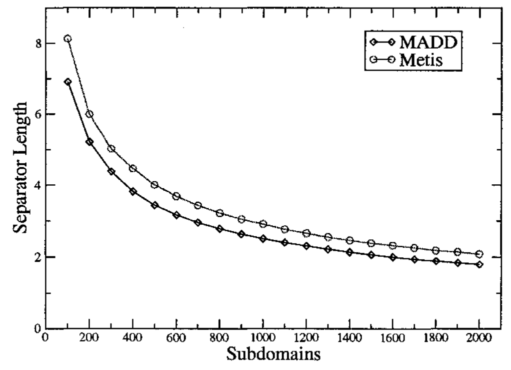


Figure 5.7: The average separator length per subdomain.

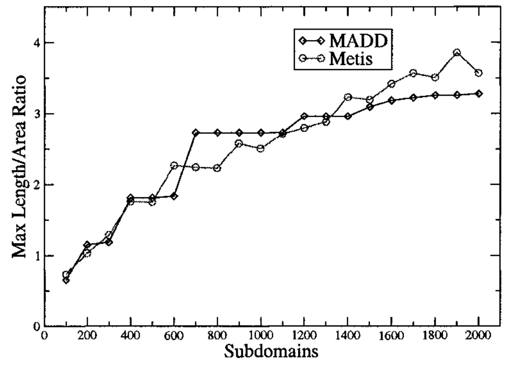


Figure 5.8: The maximum ratio of subdomain separator length/subdomain area.

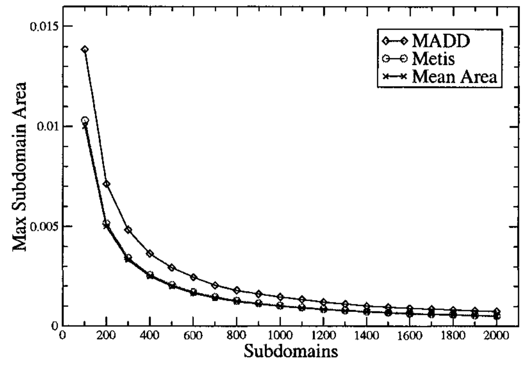


Figure 5.9: The maximum area of the subdomains. Metis gives identical maximum to the mean area, a result of practically perfect balance.

All figures refer to uniform dynamic decompositions of the Key geometry.

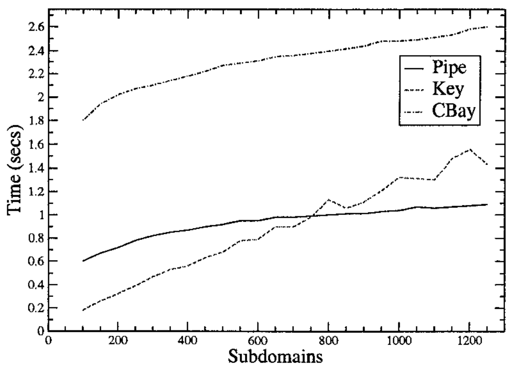


Figure 5.10: Decomposition times for the weighted boundary MADD.

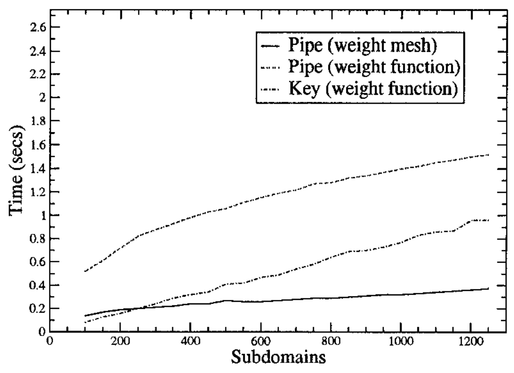


Figure 5.11: Decomposition times for the graded MADD using a weighted background mesh and weight functions.

Graded MADD. For the first group of graded decomposition experiments we used boundary weights on the three domains. The user can control the gradation level, by setting the gradation factor a, the uniform weight factor u, and the boundary interpolation weight, p, that will be applied on the interfaces (see Section 4.4. The parameters for the Pipe and the Key were $u = 3, a = 3, p = 0.5$, while for the Cbay they were $u = 2, a = 3, p = 0$.

The second group of experiments was performed on the Pipe domain using a background mesh of 1,010 points. Both area and weight values over the background mesh were used, and they produced similar decompositions for the same number of subdomains (see Fig. 4.4). The quality of the gradation depends on the ratio of the number of mesh points to the number of subdomains, as well as the gradation of the background mesh. Domain decomposition into a large number of subdomains, while using a small number of background mesh points, will result poor gradation.

We also tested the Pipe and the Key domains using weight and area functions, evaluated over a structured grid. This grid is created on the fly, when each subdomain is created; it includes a total of 21,684 points for the Pipe domain and 8,115 points for the Key. This high number of the points results in a good approximation of the density for each subdomain (the decompositions are depicted in Fig. 4.5), while the cost to create them is small (see Fig. 5.11). Of course, defining the functions analytically has the advantage of avoiding the interpolation procedure, which can have a significant cost. The weight and area functions are defined by the user and are linked dynamically, during the execution of the program.

Chapter 6

The Uniform Decoupling Method

6.1 The Decoupling Zone

The separators and the subdomains created by the MADD procedure have good quality in terms of the shape and size. Our goal though is to be able to create Delaunay meshes independently for each subdomain, and the previous procedure cannot guarantee this. In order to create the mesh independently in each subdomain we have to ensure that the final mesh will be Delaunay conforming. Blelloch et al [8] describe a projective method for decoupling the parallel Delaunay triangulation procedure for a set of points. A study of conditions for a priori conformity for constrained Delaunay triangulations is presented by Pebay and Pascal [66]. A projective separator approach is used by Galtier and George [37] for generating a Delaunay mesh independently in each subdomain. This approach though does not always guarantee a priori Delaunay conformity, and may suffer form set-backs. Said et al [73] describe a procedure for generating independently a 3D Delaunay mesh on a distributed memory environment, again with no quality guarantees.

In order to ensure the Delaunay conformity in the mesh generation context we will refine the separators using conditions derived from the mesh refining algorithm. A special "zone" around the segments of the separators (see Figure 6.1) will guarantee that the mesh generation procedure can be applied independently on each subdomain, giving a Delaunay conforming mesh for the whole domain, formed by the union of all the submeshes.

Let $\mathcal{M}$ be a Delaunay mesh generation procedure. Let $B = \partial\Omega$ be a PSLG, where Ω is the domain we wish to mesh, as defined in Section 1.3. Let $\mathcal{P}$ be the set of piecewise linear separators that decompose the domain Ω in n subdomains D_i and let $B_i = \partial D_i$ be the boundaries of the subdomains.

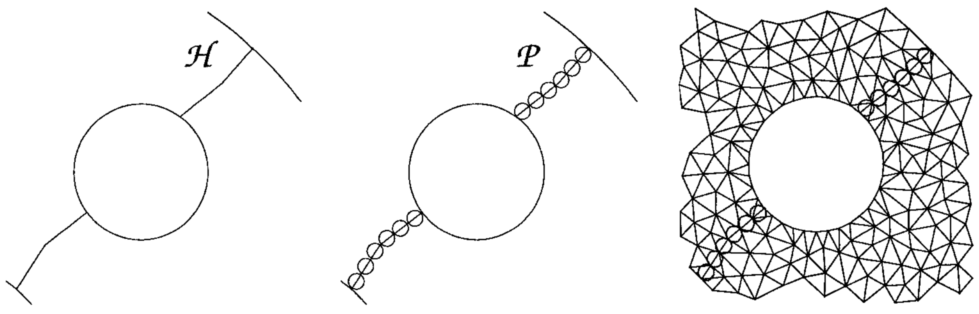


Figure 6.1: A fraction of the pipe intersection. **Left:** Part of the separators $\mathcal{H}$ inserted by MADD. **Middle:** Refining $\mathcal{H}$ gives a decoupling path $\mathcal{P}$; the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ is depicted. **Right:** Ruppert's algorithm was applied on the subdomains with an element area restriction; $\mathcal{Z}_{\mathcal{P}}$ is empty and $\mathcal{P}$ is invariant. The final mesh is Delaunay conforming.

Definition 5. *The set of the open diametral circles of all the segments that form $\mathcal{P}$ is be called the* decoupling zone *of $\mathcal{P}$ and is denoted by $\mathcal{Z}_{\mathcal{P}}$.*

Definition 6. *$\mathcal{P}$ is a* decoupling path *with respect to $\mathcal{M}$, if after applying $\mathcal{M}$ independently on the subdomains D_i, $i = 1, ..., n$, the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ is empty.*

Proposition 7. *Let M_i the mesh produced by $\mathcal{M}$ on the subdomain D_i. If $\mathcal{P}$ is a decoupling path with respect to $\mathcal{M}$, then the union $\cup M_i$ is a conforming Delaunay triangulation.*

Proof. Let M be the Delaunay triangulation of the vertices $V_M = \cup V_{M_i}$ of $\cup M_i$. We will prove that $M = \cup M_i$, by showing that the set of edges S of M are identical to the set of edges $\cup S_i$ of $\cup M_i$, thus the two triangulations are the same and $\cup M_i$ is a conforming Delaunay triangulation.

First we observe that $\mathcal{P}$ is a subset of both S and $\cup S_i$, because its decoupling zone is empty. For any edge $ab \in S$ there are two cases: (i) Both end points a, b belong to the same subdomain M_j, $a, b \in V_{M_j}$. (ii) $a \in M_i$ and $b \in M_j \setminus M_i$.

Case (i). Suppose $a, b \in V_{M_j}$. From the local Delaunay property, there is an empty circumcircle C of ab which does not include any points in V_M. Because $V_{M_j} \subseteq V_M$, C must be empty in the set V_{M_j}. Thus $ab \in S_j$ and $ab \in \cup S_i$.

Case (ii). We will show that this case cannot occur, there is no edge $ab \in S$ such that $a \in M_i$ and $b \in M_j \setminus M_i$. Suppose we have such an edge ab. Then $ab \subset \Omega$ and, since the subdomains M_i and M_j are separated by $\mathcal{P}$, a and b are separated by $\mathcal{P}$. So $ab \cap \mathcal{P} \neq \emptyset$.

On the other hand we have $\mathcal{P} \subseteq S$, which means that two edges of the triangulation M intersect. This contradicts the definition of a mesh (rule 2 in the Introduction).

Since case (ii) cannot occur, we conclude from case (i) that $S \subseteq \cup S_i$. The two triangulations M and $\cup M_i$ must have the same number of edges, so we have $S = \cup S_i$, and thus $M = \cup M_i$. This proves the proposition. $\square$

Proposition 8. *If the algorithm $\mathcal{M}$ is a mesh refinement algorithm, then the decoupling path $\mathcal{P}$ is invariant during the steps of $\mathcal{M}$, in which the Delaunay property is maintained.*

Proof. Suppose that during the procedure $\mathcal{M}$ an edge $s \in \mathcal{P}$ is destroyed. That means that the diametral circle C_s of s includes some point. Since $\mathcal{M}$ does not remove points, C_s will not be empty after the termination of $\mathcal{M}$. This contradicts the definition of the decoupling path. $\square$

Proposition 7 proves that, provided that we have constructed a decoupling path, the subdomains can be meshed independently and the final mesh will be Delaunay conforming. Observe that these results are true for a geometry in any n-dimensional Euclidean space Our next step will be to construct a 2D decoupling path from the separators created by MADD.

The decoupling path is defined with respect to a mesh generation procedure and, in many cases [17, 72], the stopping conditions of the mesh generation algorithm allow us to compute the length of the edges of the separators, so that these edges will form a decoupling path. Then we only have to refine the segments of the separators, acquiring this predefined length.

For the sequential mesh procedure we will consider Ruppert's algorithm [72], and Theorem 1 will be used for the decoupling procedure. The only requirement for Ruppert's algorithm is that the boundary angles must be at least 60°[1]. Provided that our initial boundary Ω satisfies this criterion, we can apply MADD to decompose Ω using an angle bound $\Phi_o = 60^\circ$. So, both the constructed separators and the external boundaries form angles $\geq 60^\circ$. Consequently the created subdomains are acceptable for this mesh generation algorithm.

[1]This condition is relaxed in improved versions of the algorithm.

6.2 Construction of the Decoupling Path

Let $B = \partial\Omega$ be the boundary of the domain Ω, and $\mathcal{H}$ the set of separators in Ω created by the MADD method using an angle bound of $\Phi_o = 60^\circ$. Let $B_i = \partial D_i$ be the boundaries of the created subdomains and $D_{\mathcal{H}} = D \cup \mathcal{H}$.

In order to construct a decoupling path $\mathcal{P}$ from the separators $\mathcal{H}$ we will refine $\mathcal{H}$ by inserting points along its edges, obtaining a desirable segment length. The calculation of this length is based on a parameter k. Let $L = \min\{|s|/\ s \text{ is a segment of } \mathcal{H}\}$. Let k be a real constant parameter, such that

$$0 < k \leq \min(\mathrm{lfs}_{\min}(D_{\mathcal{H}}), L/4). \tag{6.1}$$

The parameter k will be calculated from the conditions of the algorithm, so that it can be guaranteed that no edge will be created with length less than k.

The following lemma describes the refining procedure of $\mathcal{H}$.

Lemma 9. *Let s be a segment of $\mathcal{H}$. Then there is $\nu \in \mathbf{N}$ such that, after inserting $\nu - 1$ points b_i on s, we have $\frac{2}{\sqrt{3}}k \leq |b_i b_{i+1}| < 2k$ for any two consequent points b_i, b_{i+1}.*

Proof. Let l be the length of the segment s and ν such that $2(\nu - 1)k \leq l < 2\nu k$. Then, by dividing the s into ν equal subsegments, we have for the length l' of the subsegments: $\frac{2(\nu-1)}{\nu}k \leq l' < 2k$. For $\nu \geq 3$, we have $\frac{2(\nu-1)}{\nu} > \frac{2}{\sqrt{3}}$, and this proves the lemma. □

Let $\mathcal{P}$ be the separators $\mathcal{H}$ after we have inserted the points b_i, as described in the previous lemma, and let $D_{\mathcal{P}} = D \cup \mathcal{P}$. The following lemmata hold.

Lemma 10. *Let b_i, b_{i+1} two consequent points inserted on a segment s of $\mathcal{H}$. Then the diametral circle of $b_i b_{i+1}$ is empty.*

Proof. The diametral circle C of $b_i b_{i+1}$ is contained in the diametral circle of s, which by the MADD construction does not include any of the points of $D_{\mathcal{H}}$.

The remaining points to be examined are the inserted points b_j. We have that all the angles are greater than 60^o and, from Lemma 9, no created segment is less than half of any other created segment. Consequently, C cannot contain a point b_j created by the refining procedure. □

Lemma 11. *The following inequality holds: $lfs_{\min}(D_{\mathcal{P}}) \geq k$.*

Proof. We have from the relation 6.1 that $\mathrm{lfs}_{\min}(D_{\mathcal{H}}) \geq k$. We will examine the distances created by the inserted points.

Let b_i be a point inserted in a segment s of $\mathcal{H}$. For the distance d of b_i from a non incident to s segment we have $d \geq \mathrm{lfs}_{\min}(D_{\mathcal{H}}) \geq k$. The same holds for the distance d' from points that are not incident to s, because we have $d' \geq d \geq k$.

For the distance d between b_i and an incident segment we have $d \geq \sin 60^o \cdot \frac{2}{\sqrt{3}}k = k$. Finally, the distance between b_i and a point that belongs to an incident segment is greater than the distance d of the previous relation, and this completes the proof. □

The previous lemma demonstrates the property that will be used to prove that $\mathcal{P}$ is a decoupling path. Our next step will be to calculate the parameter k.

Ruppert's algorithm can be applied using either the quality criterion for the circumradius to shortest edge ratio, or by adding a criterion for the maximum area of the created elements. We will calculate k for this two cases separately. We will prove that $\mathcal{P}$ is a decoupling path for the two cases: (I) When Ruppert's algorithm is applied with only the quality criterion of the circumradius to shortest edge ratio. (II) When it is applied with an additional criterion for the maximum triangle area.

6.3 Proof of Correctness

Case I: The ratio criterion

In this case we are only interested for the circumradius to shortest edge ratio. Since k gives a bound for the size of the created segments, we would like k to be as big as possible and at the same time satisfy the relation 6.1. Proposition 1 and Lemma 11 indicate that we can define $k = \min\{\mathrm{lfs}_{\min}(D_{\mathcal{H}}), L/4\}$.

Proposition 12. *Define $k = \min\{lfs_{\min}(D_{\mathcal{H}}), L/4\}$ and let $\mathcal{P}$ be the piecewise linear separators as constructed in Lemma 9. Then $\mathcal{P}$ is a decoupling path with respect to Ruppert's algorithm.*

Proof. According to Theorem 1, Ruppert's algorithm when applied to a subdomain B_i, will not create segments less than $\mathrm{lfs}_{\min}(B_i)$. We will show ad absurdo that the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ is empty after the termination of the algorithm.

Suppose that $\mathcal{Z}_{\mathcal{P}}$ is not empty after the mesh procedure and some points have been inserted in it. That means that some boundary segments of $\mathcal{P}$ have been encroached and thus have been split in half. From Lemma 9 the length of the segments of $\mathcal{P}$ is less than $2k$ and by splitting them the created segments will have length less than k. This contradicts to Proposition 1 because, from Lemma 11, we have $\mathrm{lfs}_{\min}(B_i) \geq \mathrm{lfs}_{\min}(D_{\mathcal{P}}) \geq k$.

Thus the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ is empty after applying Ruppert's algorithm, and $\mathcal{P}$ is a decoupling path with respect to this algorithm. □

Corollary 13. *$\mathcal{P}$ remains invariant during Ruppert's algorithm execution.*

Proof. Ruppert's algorithm does not remove points and maintains the Delaunay property after inserting a point. The corollary is a direct consequence of the previous proposition and of Proposition 8. □

Proposition 12 states that we can process the subdomains independently, using Ruppert's algorithm, and the final mesh will be Delaunay conforming and of guaranteed quality. Next we will examine the case where we have an additional condition for the area of the triangles.

Case II: The ratio and max area criteria

In this case, besides the circumradius to shortest edge ratio condition, we have an additional criterion for the maximum triangle area. In many cases we want to construct Delaunay meshes, not only with good quality of angles, but also of a desired maximum size. Let A be a bound to the maximum triangle area, then all the triangles of the final mesh will have an area at most A. To achieve this, the mesh generation algorithm will split the triangles in two cases: (a) Because of the bad circumradius to shortest edge ratio. (b) Because the area of the triangle is greater than A.

We will calculate k so that the previous results will remain valid.

Lemma 14. *Let l be the smallest edge of a triangle with area greater than A and circumradius to shortest edge ratio at most $\sqrt{2}$. Then $l > \sqrt{\frac{A}{\sqrt{2}}}$.*

Proof. Let r be the circumradius of the triangle. Then $\frac{r}{l} \leq \sqrt{2}$ and $A < r \cdot l$. So, $A < r \cdot l \leq \frac{l^2}{\sqrt{2}} \Rightarrow l > \sqrt{\frac{A}{\sqrt{2}}}$. □

We want to define k in such a way that the mesh generation procedure will not create edges smaller than k. The previous lemma indicates that we should have $k \leq \frac{1}{2}\sqrt{\frac{A}{\sqrt{2}}}$. We will take $k = \min\{\mathrm{lfs}_{\min}(D_{\mathcal{H}}),\ L/4,\ \frac{1}{2}\sqrt{\frac{A}{\sqrt{2}}}\}$. Then Lemma 11 holds, and we have the following theorem:

Theorem 15. *Let $k = \min\{lfs_{\min}(D_{\mathcal{H}}),\ L/4,\ \frac{1}{2}\sqrt{\frac{A}{\sqrt{2}}}\}$ be the parameter for the point insertion procedure in Lemma 9, and $\mathcal{P}$ the produced set of separators. Then $\mathcal{P}$ is a decoupling path with respect to Ruppert's algorithm with the criteria of maximum circumradius to shortest edge ratio $\sqrt{2}$ and maximum triangle area A.*

Proof. There are two cases for splitting a triangle: a) because of its circumradius to shortest edge ratio, or b) because of its area.

When Ruppert's algorithm splits a triangle because of its circumradius to shortest edge ratio it does not create edges smaller than $\mathrm{lfs}_{\min}(D_{\mathcal{P}}) \geq k$. If a triangle is split because of its size, then from Lemma 14 we have that the smaller created edge will be no less than $\frac{1}{2}\sqrt{\frac{A}{\sqrt{2}}} \geq k$. In both cases no edge smaller than k will be created.

It is easy to see now that the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ will be empty, after Ruppert's algorithm has been applied on the subdomains B_i with the additional condition of a maximum triangle area A. If this was not so, then some edge of $\mathcal{P}$ would be encroached and split. From Lemma 9 the new edges will be smaller the k, which is a contradiction. □

In summary, the procedure of preprocessing the separators created by MADD, as described in Lemma 9, creates a decoupling path with respect to Ruppert's algorithm, in both cases of the quality and the size criteria. In the first case, the construction is based on the minimum local feature size, while in the second the maximum area of the triangles is taken into account.

The size optimality (times a constant) of the mesh produced by Ruppert's algorithm, when only the the angle criterion is used, is based on to local feature size [72, 78]. The size optimality, combined with the angle quality, provides the basis of the adaptivity to the geometry, that the Delaunay mesh displays. On the other hand, the insertion of separators by itself changes the geometry to be meshed, and the uniform refinement of the separators alternates the local feature size of the geometry. After applying the decoupling procedure, the size optimality of the mesh is not any more guaranteed. The use of the local feature size, instead of the global minimum, in creating the decoupling path, would improve the gradation and mesh size, especially when there are big differences in the local feature size. In cases though where the geometry is very simple but h-refinement is important [33], we would like to limit the area of the triangles, and in these cases the optimality of the mesh size is not based on the local feature size. The meshes produced using the area restriction are usually much larger, and thus more prompt for parallel processing. The experiments that we ran show that the over-refinement imposed by the decoupling procedure is insignificant (see Section 6.5), when the area criterion is used.

The creation of the decoupling path allows us to generate Delaunay meshes, independently for each subdomain, with good angle quality and of the desired size. The final mesh, formed by the union of the submeshes, is Delaunay conforming. As a result, this procedure decouples the domain and enables us to parallelize the mesh generation procedure, while eliminating the communication between the processors.

6.4 The Parallel Delaunay Decoupling Procedure

The procedure for the parallel mesh generation consists of two steps:

1. *The parallel MADD (PMADD) phase*: In this step the domain is decomposed using the parallel MADD method in a master/worker processor scheme (see Section 4.3), and the subdomains are distributed to the processors.

2. *The mesh generation phase*: This step is performed independently for each subdomain and includes two sub-steps:

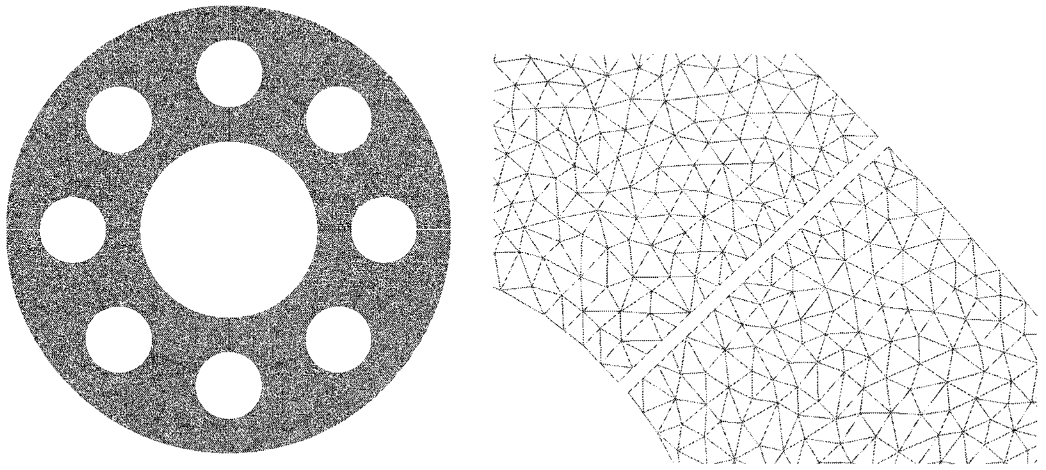


Figure 6.2: The *Pipe* domain uniformly meshed by the Decoupling procedure. Right is depicted a detail of the mesh, where the decoupling path is shown.

(a) The decoupling of the subdomains by refining the interfaces, as described in Section 6.2.

(b) The mesh generation on the subdomains. In this step the sequential mesh generator is applied independently on each subdomain.

During the PMADD phase the domain is over-decomposed (i.e. we create $N >> P$ subdomains, where P is the number of processors), in order to achieve good load balancing (see Section 6.4). The created subdomains are are assigned a priori to the processors and no data movement takes place after the PMADD phase. After the requested number of subdomains have been created, the master processor sends requests to all processors to mesh the subdomains assigned to them. Each processor iterates through its subdomains and performs two steps:

(a) Refines the interfaces, where the separators created by the MADD are refined by inserting vertices, as described in the decoupling procedure in Section 6.2, according to the given mesh quality criteria. The parameter k, that determines the refinement of the separators, is computed before the mesh generation phase begins, and is used to refine the internal boundaries of all the subdomains, independently for each subdomain. Although each interface is refined independently for the two subdomains where it belongs, the result is conforming, because the same parameter k is used, the same orientation for the interfaces, and of course the same algorithm.

(b) The mesh generation procedure is applied on the subdomains independently. The sequential mesh generator is used as is, in the form of a library. The interfaces, since they are refined from the decoupling procedure, will not be further refined form the mesh generation procedure and they will remain unchanged, as proved in Section 6. So, no communication is required, and the created meshes are Delaunay conforming. The procedure terminates when all the meshes for subdomains have been created. The parallel procedure is described by Algorithm 6.1

Static Load Balancing During the PMAAD phase the subdomains are assigned to the processors and no data movement takes place after this phase. This induces an a priori, static, load balance. Our experiments show that more than 99% of the total time is spent in the meshing phase (see Section 6.5), which does not suffer from communication or synchronization cost. Thus, the work-load balance among the processors is the main parameter that affects the performance of the method. The load balancing problem for mesh refinement is a difficult problem, because of the unpredictable computational behavior of the meshing procedure. The problem becomes more approachable by the use of the PMADD for over-decomposing the domain. The over-decomposition approach creates much more work-loads than the avaliable processors [47]. This results higher granularity of the work-loads, and thus achieves better load balancing among the processors [22]. The goals of the PMADD is to minimize the larger area and to distribute the subdomains uniformly to the processors. The obtained subdomains have similar geometric shapes, and their area is proved to be a good measure for estimating the work load for the mesh generator.

Our experimental data show, for the geometries we tested so far, that the parallel MADD procedure creates subdomains with similar "good" shape (see Figure 5.1), when the number N of subdomains is large. Figure 6.3 shows that, as we increase N, and thus decrease the area of the subdomains, the meshing time converges, with very small differences between subdomains of similar size. This result demonstrates that the area of the subdomain can be used to estimate the work-load of the mesher for this subdomain. Of course this depends on the geometry of the original domain, which is one of the parameters that determine the

Algorithm 6.1.

```
     **Master Processor:**
1.    Read the definition of the domain $\Omega$
2.    Initialize and maintain a sorted list of the areas of the subdomains
3.    while the current number of subdomains is less than $N$ do
4.        send decompose requests to processors that are assigned
          large area of subdomains
5.        receive replies about decomposition and area information
6.    endwhile
7.    send requests to processors to decouple and mesh their subdomains
8.    receive replies until all processors completed meshing
9.    send requests for termination

     **Worker Processors:**
10.   while not terminate do
11.       receive request from Master and/or other workers
12.       if request is to decouple then
13.           Apply MADD on the largest subdomain
14.           send reply to Master
15.           send a new subdomain to other processor
16.       endif
17.       if request is to receive a subdomain then
18.           Add the new subdomain to this worker's mesh-queue
19.           send reply to Master
20.       endif
21.       if request is to start meshing then
22.           for each assigned subdomain do
23.               Refine the separators according to the decouple procedure
24.               Apply the sequential mesh generator on the subdomain
25.           endfor
26.           send completion message to master
27.       endif
28.   endwhile
```

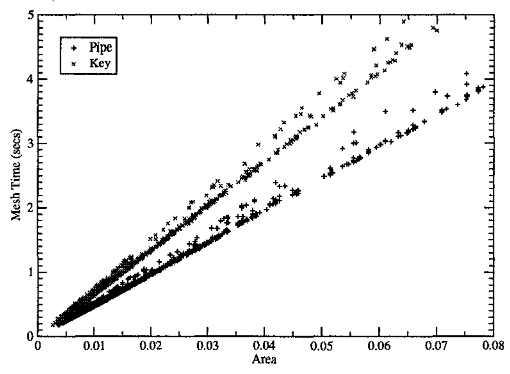


Figure 6.3: Mesh time for different sizes of subdomains of the key and the pipe geometry.

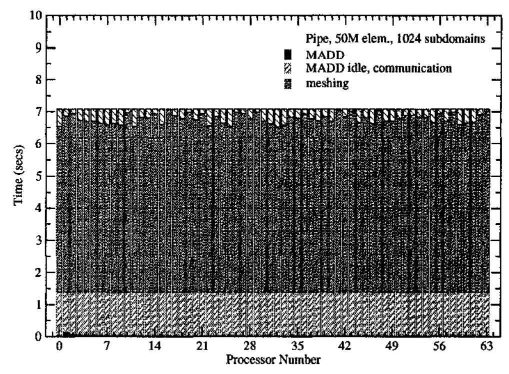


Figure 6.4: The work balance for 64 procs, 50M elem.

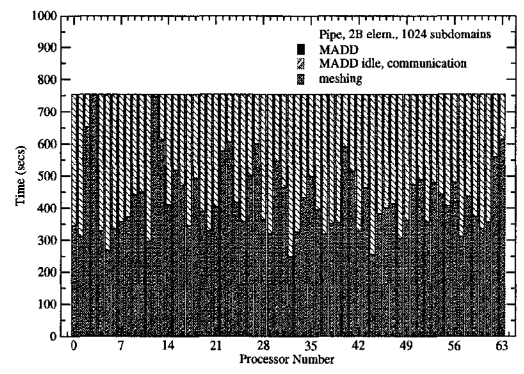


Figure 6.5: The work balance for 64 procs. 2B elem., 1024 subdomains for the pipe.

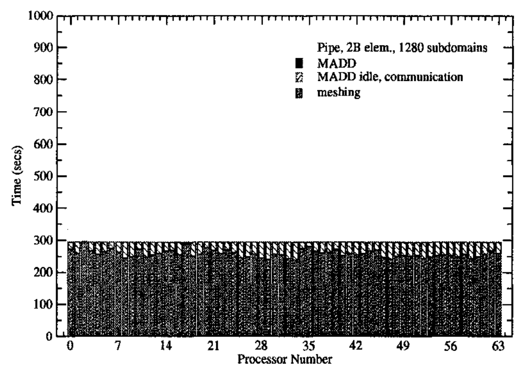


Figure 6.6: The work balance for 64 procs. 2B elem. 1280 subdomains for the pipe.

level of required decomposition.

The load balance among the processors is achieved by balancing the total area of the subdomains assigned to each processor. The first effort to create subdomains with similar sizes takes place during the graph partition. This result though is not guaranteed, and the obtained subdomains can have differences in size. By over-decomposing we have the ability to distribute the subdomains, so that each processor is assigned approximately the same total size. Moreover, the random distribution of the subdomains gives a more uniform assignment of subdomains that differ from the average in terms of size and geometry. The results of this simple approach are good. Figure 6.4 depicts the load balance among 64 processors for the pipe geometry, for 1024 subdomains and 50M mesh size. This picture is typical in most cases. However, we have observed that the load balance does not depend only on the geometry and the size of the subdomain, but also on size of the created mesh.

Figure 6.5 shows the load balance for the same decomposition of the pipe, as in Fig. 6.4, this time for a mesh size of 2 billion elements. We see that the good load balance of the Figure 6.4 is destroyed. The reason for this is that the time for creating larger meshes is much more sensitive to area and geometry differences. The answer to this problem is to increase N. In this way we improve two parameters: *i*) the size of the mesh for each subdomain is decreased, and thus the time to create it is less sensitive to the differences, and *ii*) a more uniform assignment of the subdomains can be accomplished. Figure 6.6 shows the balance for the same mesh size, 2 billion elements, by decomposing it into 1280 subdomains. This small increase of the number of subdomains gives an impressive improvement, the load balance is satisfactory and the total time is decreased in less than half, the reasons are described in Sections 6.5, 6.5.

The previous example shows that the load balance is sensitive to the size of the final mesh. The level of the required decomposition depends not only on the geometry and the number of the processors, but mainly on the size of the final mesh. Let E be an estimation for the final size of the mesh in millions of elements. From our experiments we found that, for our setup, the number of subdomains should be at least $N = \frac{E}{1.6}$. This means that in average 1.6M elements will be created for each subdomain. A higher decomposition has, of course, higher time cost, but this cost is insignificant against the gain, Figures 6.5 and 6.6, as well as the results in the next section demonstrate it. A dynamic load balance approach is described in Section 7.6.

6.5 Performance Evaluation

We evaluate the Parallel Delaunay Decoupling (PDD) method with respect to three requirements: (1) stability, (2) parallel efficiency, and (3) code re-use. Our experimental data indicate that the PDD method is stable i.e., the elements of the distributed mesh retain the same good quality of angles as the elements generated by the Triangle (see Figures 6.8 and 6.13 (right)); at the same time it is very efficient as our fixed and scaled speedup data (see Figures 6.12, and 6.13 (left)) indicate. Finally it is based on 100% code re-use i.e., existing sequential libraries like Metis and Triangle are used without any modifications for the parallel mesh generation.

Experimental Setup. We have used two model domains (see Figure 5.1, with relatively simple geometries[2]: The *Pipe*, a cross section of rocket from a NASA model problem where the peripheral pipes are used to cool the main cylinder in the center that contains combustion gases, and the *Key*, a domain provided with Triangle. We ran three sets of experiments: (1) to observe the behavior of the MADD and Decoupling method in sequential execution for small meshes, 4-5 million (M) elements, (2) to calculate the fixed speedup for fixed size meshes of the order of 40-50M elements, and (3) to compute the scaled speedup for meshes whose size range from 12M to 2 billion (B) elements.

The programming language for our implementation was C++ and DMCS [3] was used as the communication substrate. The Triangle [77] library was used for the mesh generation procedure as well as for the creation of the Delaunay triangulation during the MADD procedure. The parameters passed to Triangle for the mesh generation were two: (*a*) for the quality the elements (Ruppert's algorithm is used to achieve circumradius to shortest edge ration less then $\sqrt{2}$), and (*b*) for the maximum area of the generated elements. Also, Metis [45] was used for the graph partitioning step in the MADD procedure. The cases that Metis returned non-connected subgraphs were recognized and discarded. All the libraries where used without modifications, minimizing the cost for the parallel implementation and achieving 100% code-reuse.

The experiments ran on SciClone, a high-performance computing environment in the College of William and Mary. SciClone is a heterogeneous cluster of Sun workstations which use Solaris 7 operating system. For our experiments we have used a subcluster of 32 dual-cpu Sun Ultra 60 workstations 360 MHz, with 512 MB memory and 18.2 GB local disk. Networking was provided by a 36-port 3Com Fast Ethernet switch (100Mb/sec).

[2]The complexity of the geometry will challenge the PMADD and in particular the Delaunay triangulation procedure. Provided the efficiency of *Triangle*, this shouldn't be a problem. The mesh refinement procedure will be applied on the created subdomains, which have simple geometries. However, for three dimensional cases the complexity of the geometry is a much more serious issue.

Sequential Experiments

We ran a set of sequential experiments in order to compare the sequential Delaunay decoupling method, where we over-decompose the domain, with Triangle, the best known publicly available sequential guaranteed quality Delaunay mesh generation code for two dimensional domains. In these experiments we examine the affects of the decoupling procedure with respect to the performance of the mesh procedure, the size of the final mesh, which indicates that the over-refinement we introduce is insignificant, and the quality of the elements in terms of the angle distribution. The size of the meshes we created is limited by the size (5.5M) we were able to generate with Triangle due to memory limitations. However, using the Delaunay decoupling method we were able to generate more than 30M on a single processor.

Figure 6.7 shows the ratio of the size of the decoupled meshes over the size of the non-decoupled mesh, which is a measure of the over-refinement we introduce when we decouple the domains. Similarly, Table 6.1 presents the number of elements for different levels of decoupling. The over-refinement is insignificant, it is less than 0.4%, despite the intense over-decomposition (less than 90K elements per subdomain).

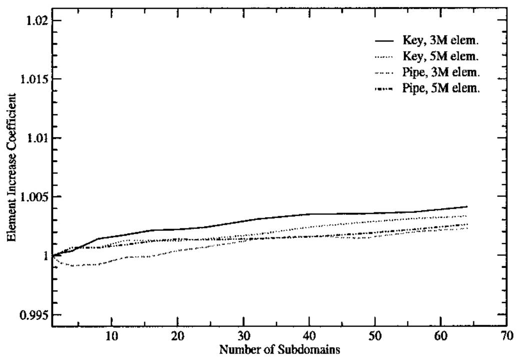


Figure 6.7: The increase of number of elements for decoupling into different number of subdomains.

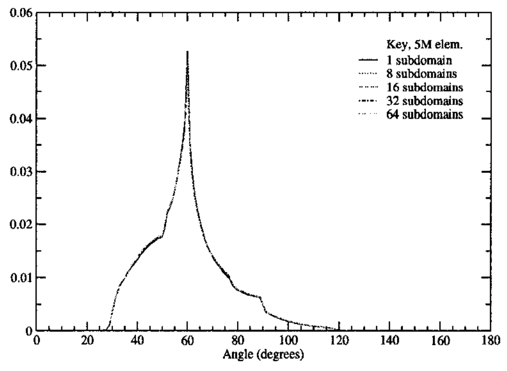


Figure 6.8: The angle distribution for different number of subdomains.

The overhead of the sequential MADD method is approximately linear with respect to the number of subdomains, see Figure 6.9. This overhead is small compared to the mesh generation time. The total execution time using the sequential decoupling procedure is

Subdomains	1	8	16	32	48	64
Key elements	5,193,719	5,197,066	5,200,395	5,203,023	5,208,215	5,210,857
Total time	46.146	38.414	38.204	37.590	37.322	37.333
Pipe elements	5,598,983	5,602,668	5,605,819	5,607,055	5,609,404	5,613,624
Total time	59.263	41.342	41.046	40.370	40.352	40.147

Table 6.1: The number of elements and the total time (in seconds) for the same mesh generation parameters and for different levels of decoupling. The times do not include the mesh merging procedure.

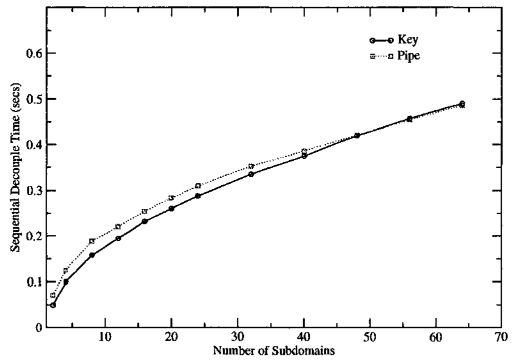


Figure 6.9: The time for the sequential MADD.

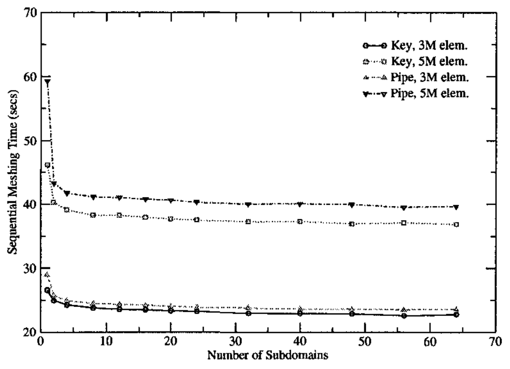


Figure 6.10: The time for sequential meshing after decoupling into subdomains. The times do not include the mesh merging procedure.

decreased up to 68% of the time it takes for Triangle to generate a mesh with the same quality. As the size of the mesh increases the performance of the decoupling procedure compared to Triangle is improving even further, because the size of the working set for each subdomain is smaller and the Delaunay mesh algorithm used in Triangle has a non-linear time complexity [77].

The quality of the elements produced after the decoupling of the domain into subdomains is evaluated by comparing the distribution of angles. We compare the angles of the elements from both the non-decoupled mesh generated by Triangle and the decoupled ones generated by our method. Figure 6.8 shows that the distribution is the same. The above results hold as we scale the mesh size in our parallel experiments.

In summary, the decoupling method demonstrates merits even for sequential mesh generation. The gains in the performance from the better memory utilization cover the small overheads due to decoupling and over-refinement, while the element quality is independent of the decoupling, which shows that our method is stable regarding the quality of the mesh.

Parallel Experiments

We performed two sets of experiments in order to calculate the fixed and scaled speedup using 8, 16, 32, and 64 processors. With 64 processors we were able to generate 2.1 billion (B) high quality elements for the Pipe in less than 3.5 minutes, while using Triangle [77] on a single workstation we were able to generate 5.5 million (M) elements in about one minute (see Tables 6.1 and 6.3).

In the rest of the section we present performance data for both the parallel medial axis domain decomposition (PMADD) method and the parallel mesh generation. The PMADD procedure is evaluated in terms of its total parallel execution time which includes some communication and idle time and the maximum computation time spend on a single processor. The parallel mesh generation phase does not require communication and its performance is measured in terms of maximum and average computation time of processors. The ratio of these two numbers is used to measure the load imbalance of the parallel meshing phase.

Finally, we evaluate the scalability of the method in terms of two performance criteria: (1) the **average time** that it takes for one element to be created on a single processor, over all the processors and elements that are created, and (2) the **overhead** cost (due to decomposition and parallelism) for each processor we use. Both criteria indicate that the parallel mesh generation method we present here is scalable and that we can generate billions of elements with insignificant overheads (see Table 6.3).

Fixed Size Mesh Experiments In the fixed size set of parallel experiments we used a mesh of 40M elements for the Key domain and 50M for the cross section of the Pipe. For the key domain we created 12 subdomains for each processor while for the pipe 16 subdomains. The maximum triangle area is fixed throughout the experiments for each domain.

The results are presented in Table 6.2. The data again indicate an unimportant increase in the number of elements for the different levels of over-decomposition, which shows that the over-refinement we introduce is insignificant. The total execution time and the computation time for the actual mesh generation are depicted in Figure 6.11. These times are very close,

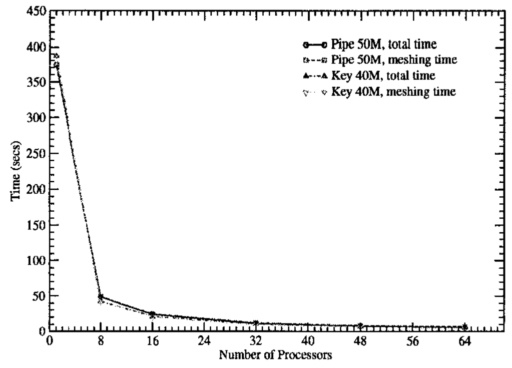


Figure 6.11: The performance for fixed size mesh.

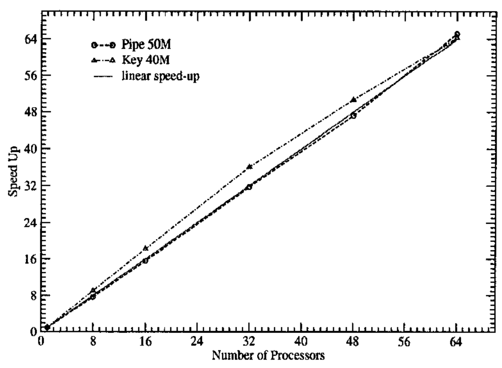


Figure 6.12: The speedup for fixed size mesh.

No of processors	1	8	16	32	48	64
The Key Domain						
No of subdomains	12	96	192	384	576	768
Mesh size (M)	43.32	43.34	43.37	43.41	43.43	43.45
PMADD time	0.20	0.37	0.44	0.60	0.83	1.05
Meshing time	386.32	42.35	20.72	10.12	6.79	4.96
Total time	386.52	42.72	21.16	10.72	7.62	6.01
The Pipe Domain						
No of subdomains	16	128	256	512	768	1024
Mesh size (M)	50.93	50.97	51.00	51.05	51.08	51.11
PMADD time	0.27	0.51	0.60	0.89	1.07	1.47
Meshing time	374.15	48.80	24.03	11.80	7.93	5.74
Total time	374.42	49.29	24.63	12.69	9.00	7.21

Table 6.2: Performance data for the key and the pipe geometry for a fixed maximum element area. All times are in seconds and mesh sizes are in millions (M).

because the PMADD overhead cost is very small. This cost is neutralized by the effect of over-decomposition, which along with the good load balancing and zero communication during the parallel meshing, lead to superlinear speedup, see Figure 6.12. The speedup is calculated against the total time it takes to create the mesh on one processor, as it is presented in Table 6.2.

Scaled Size Mesh Experiments A more practical way to evaluate the scalability and true performance of a parallel algorithm and software is to scale the size of the problem in proportion to the number of processors used. In the following experimental data we use the same level of decomposition for every configuration of processors, i.e., we keep the average number of subdomains per processor constant, and thus we eliminate the effect of over-decomposition in the resulting performance data. Theoretically we should be able to achieve the same creation time per element per processor for all the parallel configurations independently of the number of processors used. However, this is not feasible for the following two reasons: (1) the decomposition overhead, which increases very slowly but nevertheless there is an increase in the overhead as the number of processors increases and (2) load imbalances due to unpredictable and variable computation of the mesh generation kernel.

Table 6.3 shows some performance indicators for the two model problems we use, the key and the pipe geometry. In the experiments for the key model we created 12 subdomains per processor and generated on average 1.6M elements per subdomain i.e., total 20M per processor. For the pipe model we created 20 subdomains per processor and generated on average 1.6M elements per subdomain i.e., total 32M per processor. Small differences exist in the size of the mesh because our stopping criteria are based on the quality and size of elements, and thus the mesh size cannot be exactly predefined. It is clear from the Table 6.3 that for larger processor configurations, like 64 processors, the 99.5% of the total execution time is spent in the meshing phase by the Triangle. This suggests that for realistic problems the PMADD overhead is about 0.5% of the total execution time.

We observe that, while the max PMADD time on one processor remains almost constant, the time for PMADD phase increases as the number of processors increases. This is in agreement with the analysis in Section 6.4. As the number of processors increases, the number of PMADD iterations increases, although the number of the subdomains per processor is constant. In each PMADD iteration all the processors finish the decomposition, before the next iteration begins. This synchronization imposes an additional cost in the PMADD time. Moreover, the communication during this phase increases, as the number of processors increases. Fortunately, the communication and synchronization cost is less than 0.02 secs per processor. In comparison with the total execution time this cost is very small.

The load imbalance is measured by the ratio of the maximum meshing time on one processor and the average meshing time for all the processors. In Table 6.3 we observe that the load balance for the pipe is very good, 1.14 for 64 procs, while for the key is satisfactory,

No of processors	1	8	16	32	48	64
The Key Domain						
No of subdomains	12	96	192	384	576	768
Mesh Size	20M	160M	320M	650M	860M	1.3B
Total time	152.43	177.31	192.41	213.91	166.10	205.26
Max meshing Time	152.23	176.92	191.93	213.26	165.25	204.19
Aver. meshing Time	152.23	165.75	168.04	170.31	137.70	163.14
Imbalance	1	1.067	1.142	1.252	1.200	1.252
MADD Phase time	0.20	0.38	0.44	0.63	0.84	1.05
Max MADD time	0.20	0.14	0.13	0.13	0.12	0.13
Tot. time/(elem./procs)	7.33	8.73	9.47	10.54	9.20	10.11
Additional Cost /procs	0%	2.4%	1.8%	1.4%	0.5%	0.6%
The Pipe Domain						
No of subdomains	20	160	320	640	960	1280
Mesh size	32M	240M	500M	1B	1.4B	2.1B
Total time	236.00	247.10	245.32	279.59	246.59	294.39
Max meshing time	235.71	246.53	244.65	278.56	245.09	292.71
Aver. meshing time	235.71	226.78	231.15	253.59	218.56	255.87
Imbalance	1	1.087	1.058	1.098	1.121	1.144
MADD phase time	0.29	0.55	0.67	1.01	1.48	1.66
Max MADD time	0.29	0.19	0.17	0.17	0.16	0.18
Tot. time/(elem./procs)	7.30	8.23	7.94	8.51	8.45	8.96
Additional Cost /procs	0%	1.6%	0.6%	0.5%	0.3%	0.4%

Table 6.3: Performance data for the key and the pipe geometry. The meshing time includes the time of the decoupling procedure (MADD). The MADD phase includes the load balance estimation procedure and the distribution of the subdomains to the processors. The imbalance is measured as ratio of the max meshing processor time over the average. All times are in seconds except for the time/(elem./procs) which is in microsecs.

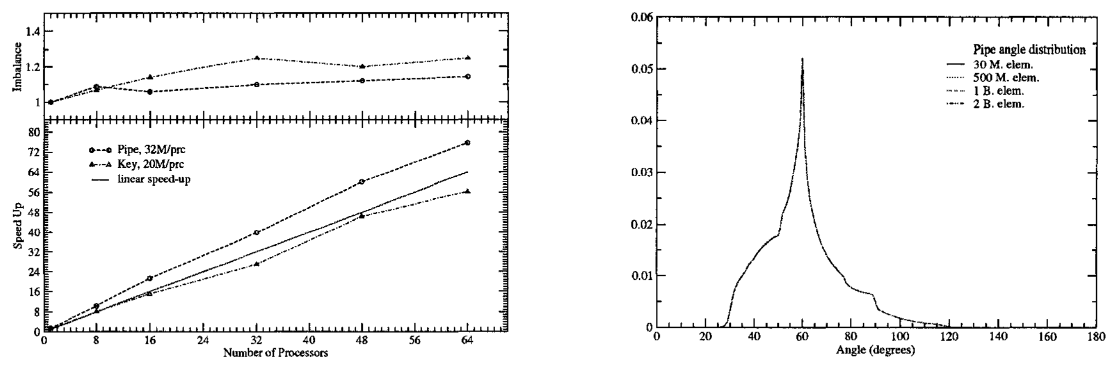


Figure 6.13: **Left: Top** is presented the imbalance and **down** the speedup for the scaled experiments. The speedup is measured against the sequential creation of 5M elements and is based on the overall time it takes for one element to be created. Observe the direct impact of the imbalance to the speedup. **Right:** The angle distribution for scaled mesh sizes of the pipe.

1.25. The load-balance is based on overdecomposing the domain and equidistributing the areas, and although it depends on the size of the mesh as we saw in Section 6.4, it also depends on the geometry and the number of the processors.

An important measure for evaluating the efficiency of a parallel meshing method is the (total) time spent for creating one element on one processor. Let $T^{(P)}$ be the total time running on P processors in order to create a mesh of size $S^{(P)}$. Then, the time per element, per processor is $T_e^{(P)} = \frac{T^{(P)} \cdot P}{S^{(P)}}$. This measure eliminates the differences in the mesh size, providing a more objective view of the scaled performance. We see in Table 6.3 that this time is almost constant, and thus the method is scalable. The slight increase of this time is mainly due to the imbalance increase, while the contribution of the overhead time cost is very small. This is evident in Figure 6.13, where the imbalance is depicted on the top and the scaled speedup down. The scaled speedup for P processors is measured as $U_P = \frac{T_e^s \cdot P}{T_e^{(P)}}$, where T_e^s is the time to create sequentially one element for a non-decomposed mesh of size 5M. We again observe the superlinear speedup for the same reasons as in the fixed size experiments. It is obvious in this figure the direct impact of the imbalance to the speedup.

Another measure for evaluating the scalability is the additional cost time cost for each processor that we use, relatively to the total time when running on one processor. The additional cost C_P per processor, when using P processors, is computed as $C_P = \frac{T_e^{(P)} - T_e^{(1)}}{T_e^{(1)} \cdot P}$. Taking into account that the mesh size $S^{(P)}$ is approximately proportional to the number of processors P, we have $C_P \simeq \frac{T^{(P)}-T^{(1)}}{T^{(1)}.P}$. We can consider the quantity $T^{(1)}$ as the ideal time for creating on P processors a mesh of size $S^{(P)} \simeq P \cdot S^{(1)}$, since the effect of over-decomposition is eliminated. In this way the additional cost C_P measures the distance from the ideal speedup, distributed to the number of processors used.

The time $T_e^{(P)}$ is increasing as P increases, the reasons were explained above. This increase though is small for the key and even smaller for the pipe domain. It is interesting to observe that the additional cost C_P tends to decrease, as P increases. Although we have to pay a (small) cost in the performance for each additional processor we use, this cost tends to decrease, when measured in scale. This result underlines the scalability of the method.

Finally we should compare the quality of the elements of scaled meshes that the decoupling procedure produces. In Figure 6.13 right is depicted the distribution of the angles of the elements, for meshes varying from 30M triangles to 2.1B. The quality is obviously the same.

Chapter 7

The Graded Decoupling Method

7.1 Graded Mesh Generation

Delaunay mesh generation procedures, as the ones proposed by Chew [17, 18], Ruppert [71, 72], and further developed by Shewchuk [78, 81], create boundary conforming triangular meshes of good quality. The area of the elements in Delaunay meshes can grow fast, as we move away from the boundary, resulting meshes of optimal size (up to a constant factor)[72, 60]. The gradation reflects the geometric properties of the domain, but it does not reflect the computational characteristics of the model. Regions of the domain where is harder to approximate the solution, or we desire hinger accuracy, should be meshed more intensively. These parts can be determined in advance, based on the properties of the geometry and the model, or as a result of an error estimation function from a previous FEM procedure.

The problem of determining the element size, and thus the gradation of the mesh, during the mesh generation and refinement has been studied extensively (cf. [10, 54, 62, 30, 90]). Usually the size of the elements is computed as a function of: (a) the geometry of the domain, (b) the distance from sources of activity in the model (like heat sources), (c) a gradation control bound, and (d) error estimators, typically computed from a previous solution over a coarse mesh. The common way to control the element size of a mesh is to employ a sizing function that determines the element size. In the anisotropic case this function can be viewed as a tensor field over the domain [10], while in the isotropic case as a real function. In this work we consider only the isotropic case. The sizing function can be defined over the whole domain, or over a background mesh of the domain (alternatively the sizing function can be defined over a control space). The objectives of the sizing function are two-fold: to capture the complexity of the geometry, and to optimize the quality of the mesh with regard to a specific model.

The complexity of the geometry lies on the properties of the boundary, which in turn can be used to specify a sizing function. Some of these geometric properties used to determine a sizing function are the angle variation between boundary faces [53, 54], the curvature of the boundary [28, 62, 30, 63, 49, 90], and the proximity between different boundary entities [90, 68, 70].

The behavior of the model can be assessed based on previous experience and error estimations. Sources of activity can be translated to geometrical entities, which in turn give sizing functions [53, 54, 90] usually in terms of the distance from the source. Another way is to utilize an initial, relatively coarse, mesh to obtain error estimations. This mesh can be used as a background mesh for generating a new mesh, with element sizes governed by the error estimations. The element size at each point of the domain can be determined through an interpolation procedure [62, 63]. Alternatively, Cartesian [30], and octree based background grids have been proposed to control the element sizes.

The final mesh should demonstrate bounded the gradation, in order to be of good quality, and several methods have been proposed for smoothing the sizing function and bounding the gradation. For the discrete cases the use of interpolation smoothing methods is common [11, 62, 51], while for the continuous case gradient limiting methods can be applied [67].

7.2 The Graded Decoupling Approach

In Chapter 6 we described the decoupling procedure for uniform parallel guaranteed quality Delaunay mesh generation. In this chapter we extend the decoupling method for generating large graded Delaunay meshes in parallel. A continuous sizing function, or discrete function on a background mesh, is used to control the mesh element size, and thus the gradation of the mesh. The continuous sizing function is considered to be a real (hence isotropic) positive function defined over the whole domain. On the other hand, the background mesh consists of a set of nodes in the domain, that store the desired element size. As in the uniform case, we target the elimination of the communication during the mesh generation procedure, by applying a sequential mesh generator independently on each subdomain.

The approach for the graded decoupling procedure is similar to the uniform decoupling procedure. The separators are prerefined in a way that guarantees the conformity of the subdomain meshes, when they are created independently. The refinement in the graded case will be controlled by the sizing function. The notions of the *decoupling zone* and *decoupling path*, as they are given in definitions 5 and 6, will be used for the graded case too, since they are independent of the way the mesh is created. Propositions 7 and 8, which guarantee the mesh conformity, hold in general for any Delaunay mesh generation procedure, and consequently are also true for the graded case. Thus it is sufficient to identify and construct a decoupling path for the graded Delaunay mesh generation procedure. To this end we will use the gradation of the mesh to control the decomposition procedure. This will allow the refining size (i.e. the separator lengths) to be bounded along the boundary of every subdomain.

In the next section we describe a gradation controlled domain decomposition that will accommodate the graded decoupling method. In Section 7.4 the properties of the graded decoupling path are identified, and in Section 7.5 we describe the construction of the decoupling path.

7.3 Gradation Controlled Domain Decomposition

The gradation produced by the sizing function should be bounded, and, especially in the case of large meshes, we expect the mesh to be locally near uniform. Our goal during the domain decomposition is to identify bounded gradation regions of the domain. This can be achieved by imposing a constant upper bound to the gradation of the sizing function inside each subdomain. Moreover, neighboring parts of the mesh should not present large size difference, and so the gradation among neighboring subdomains should also be bounded. Decompositions with the above properties can be used to decouple the mesh generation procedure.

We formulate the above two conditions as follows. For any subdomain D_i, let $m(D_i)$ denote the minimum element area and $M(D_i)$ the maximum element area inside D_i, as these are defined by the sizing function or the background mesh.

Condition 1. *For a predefined constant $R_1 > 1$ we should have*

$$M(D_i) \leq R_1 m(D_i),$$

for all subdomains D_i.

Condition 2. *For a predefined constant $R_2 > 1$ we should have*

$$m(D_i) \leq R_2 m(D_j),$$

for all neighboring (sharing a common internal boundary) subdomains D_i and D_j.

We expect the gradation inside a subdomain to be at most as large as among neighboring subdomains, so we require $R_1 \leq R_2$. An interesting theoretical problem is to find an optimal domain decomposition, in terms of the number of subdomains, that satisfies the above two conditions.

In the following we describe a geometric domain decomposition procedure that satisfies the conditions that where formulated above. The procedure is based on the Medial Axis domain decomposition, which is applied iteratively until the conditions 1 and 2 are met. We examine both the cases of a sizing function f and of a background mesh G as control mechanisms for the maximum size of the elements. The sizing function f is considered to be positive and continuous over the whole domain Ω, while the background mesh G is an unstructured mesh over the domain.

Domain Decomposition Controlled by a Sizing Function

We will apply the MADD procedure iteratively, so that the final decomposition satisfies the conditions 1 and 2. Given a decomposition $\mathcal{D}_n$, we identify the set $\mathcal{B}_n$ of subdomains that do not satisfy either condition 1 or condition 2. Namely, if for some subdomain $D_i \in \mathcal{D}_n$ we have $M(D_i) > R_1 m(D_i)$, then $D_i \in \mathcal{B}_n$, and if for two neighboring subdomains $D_i, D_j \in \mathcal{D}_n$ we have that $m(D_i) > R_2 m(D_j)$, then $D_i, D_j \in \mathcal{B}_n$. The largest subdomain of $\mathcal{B}_n$ is decomposed using the MADD procedure, giving a new decomposition $\mathcal{D}_{n+1}$. The procedure is repeated until all subdomains satisfy these two conditions. Algorithm 7.1 outlines this iterative procedure.

Algorithm 7.1.
```
1.  input initial decomposition $\mathcal{D}_1 = \{\Omega\}$
2.  identify the set $\mathcal{B}_1 \subseteq \mathcal{D}_1$ of non-acceptable subdomains
3.  $i = 1$
4.  while $\mathcal{B}_i \neq \emptyset$ do
5.     let $B \in \mathcal{B}_i$ be the largest subdomain
6.     apply MADD to $B$
7.     $i = i + 1$
8.     let $\mathcal{D}_i$ be the new decomposition
9.     identify the set $\mathcal{B}_i \subseteq \mathcal{D}_i$ of non-acceptable subdomains
10. endwhile
```

The termination of the algorithm will guarantee that the produced decomposition satisfies both the conditions 1 and 2. In order to prove the termination we will use the observation that the MADD produces decomposition topologies equivalent to the Euclidian topology, i.e., the maximum diameter of the subdomains tends to zero, when we apply iteratively the MADD on the largest subdomain. This notion is formally expressed as follows: Let $\mathcal{D}_n$ be a sequence of decompositions, each produced from the previous by applying the MADD to the largest subdomain. Then, $\max_{D \in \mathcal{D}_n} \delta(D) \to 0$, where $\delta(D) = \max \|y - x\|, x, y \in D$ is the diameter of the subdomain D.

Commonly the objectives of graph partitioner are two-fold. The first objective is to create balanced decompositions, a property that can be described as follows: There is a constant $b_1 < 1$, so that after we decompose any subdomain D into the subdomains D_i, D_j, we have $\max\{|D_i|, |D_j|\} \leq b_1|D|$. The second objective is the creation of small separators, which is usually formulated as minimizing the ratio $\frac{|\partial D_i|}{|D_i|}$. These objectives allow us to prove that the MADD produces decomposition topologies equivalent to the Euclidean.

Lemma 16. *Let $\mathcal{D}_n$ be a sequence of decompositions, each produced from the previous by applying the MADD to the largest subdomain, for which the following two conditions hold: There is a constant $b_1 \in \mathbb{R}$, $b_1 < 1$, such that $\max\{|D_i|, |D_j|\} \leq b_1|D|$, for any subdomains D_i, D_j obtained by decomposing a subdomain D. There is a constant $b_2 \in \mathbb{R}$ such that $\frac{|\partial D_i|}{|D_i|} \leq b_2$ for any subdomain D_i. Then we have $\max_{D \in \mathcal{D}_n} \delta(D) \to 0$, where*

$\delta(D) = \max \|y - x\|, x, y \in D$ is the diameter of the subdomain D.

Proof. The proof is performed in three steps:

Step 1. Let $A_n = \max_{D \in \mathcal{D}_n} |D|$. We will show that $A_n \to 0$. The sequence $\{A_n\}$ is clearly decreasing, so we only need to find a subsequence $\{A'_n\} \subseteq \{A_n\}$, such that $A'_n \to 0$.

Let $A'_n = \max_{D \in \mathcal{D}_{2^n}} |D|$. We will prove by induction that $A'_n \leq b_1^n |\Omega|$. For $n = 0$, and $\mathcal{D}_1 = \{\Omega\}$, the relation is obviously true. Suppose the claim is true for some $n = m$, we will prove it is true for $n = m + 1$.

For any subdomain $D \in \mathcal{D}_{2^m}$ decomposed into two subdomains D_i, D_j we have

$$\max\{|D_i|, |D_j|\} \leq b_1 |D| \leq b_1 b_1^m |\Omega| = b_1^{m+1} |\Omega|.$$

Next we will show that any subdomain $D \in \mathcal{D}_{2^m}$, with $|D| > b_1^{m+1}|\Omega|$, will be decomposed. Observe that the decomposition $\mathcal{D}_{2^m}$ contains $|\mathcal{D}_{2^m}| = 2^m$ subdomains, and any new subdomain will have area less or equal to $b_1^{m+1}|\Omega|$. The decomposition $\mathcal{D}_{2^{m+1}}$ is obtained from $\mathcal{D}_{2^m}$ after decomposing $2^m = |\mathcal{D}_{2^m}|$ subdomains. So, all the subdomains $D \in \mathcal{D}_{2^m}$, with $|D| > b_1^{m+1}|\Omega|$, will be decomposed.

From the above we conclude that for any subdomain $D \in \mathcal{D}_{2^{m+1}}$ we have $|D| \leq b_1^{m+1}|\Omega|$, and thus $A'_n \leq b_1^{m+1}|\Omega|$.

From the induction we have $A'_n \to 0$, and consequently $A_n \to 0$.

Step 2. It is easy to see that $\max_{D \in \mathcal{D}_n} |\partial D| \to 0$. For any subdomain D we have $|\partial D| \leq b_2 |D|$, and so

$$\max_{D \in \mathcal{D}_n} |D| \to 0 \Rightarrow \max_{D \in \mathcal{D}_n} |\partial D| \to 0.$$

Step 3. The subdomains are connected, so we have $\delta(D) \leq |\partial D|$. Consequently

$$\max_{D \in \mathcal{D}_n} |\partial D| \to 0 \Rightarrow \max_{D \in \mathcal{D}_n} \delta(D) \to 0,$$

and the lemma is proved. □

There are no strict mathematical proofs, in general, that a graph partitioner will achieve the two objectives mentioned above. In practice though, we have observed that state-of-art partitioner, like Metis [57], will give decompositions that meet these two objectives for all the geometries we have tested, and for very large scale decompositions. We proceed to give a proof of termination of the algorithm under the conditions of the previous lemma.

Lemma 17. *If for a sequence of decompositions $\mathcal{D}_n$ we have* $\max_{D\in\mathcal{D}_n}\delta(D)) \to 0$*, then there is a decomposition $\mathcal{D}_k$ that satisfies the conditions 1 and 2.*

Proof. We have that f is continuous over a compact domain, and thus is uniformly continuous. Moreover f is bounded below by a constant positive number. Then for any $\epsilon > 0$ there is a $\delta > 0$, such that if $\|x - y\| \leq \delta$, we have $\frac{f(x)}{f(y)} \leq 1 + \epsilon$. Let δ be such that the inequality is satisfied for $1 + \epsilon = \min(R_1, R_2)$.

If $\mathcal{D} = \{D_i\}$ is a decomposition such that $\max\{\delta(D_i)\} \leq \delta$, then $\mathcal{D}$ obviously satisfies the conditions 1 and 2. Let $\mathcal{D}_k$ such that $\max_{D\in\mathcal{D}_k}\delta(D) < \delta$. Such decomposition exists, because $\max_{D\in\mathcal{D}_n}\delta(D) \to 0$, and satisfies the conditions 1 and 2. $\square$

Theorem 18. *Under the conditions of Lemma 16, Algorithm 7.1 terminates, giving a decomposition that satisfies the conditions 1 and 2.*

Proof. If Algorithm 7.1 terminates, then by the construction it will produce a decomposition that satisfies the conditions 1 and 2. We will prove the termination by contradiction.

We observe that if $B' \in \mathcal{B}_{n+1}$, then $B' \subseteq B$ for some $B \in \mathcal{B}_n$. We have from Lemma 16 that $\max_{B\in\mathcal{B}_n}\delta(B) \to 0$. Suppose that the algorithm does not terminate, then for some k we will have $\max_{B\in\mathcal{B}_k}\delta(B) < \delta$, where δ is defined in Lemma 17. Then, from the same lemma, $\mathcal{B}_k$ satisfies the conditions 1 and 2, which contradicts the definition of $\mathcal{B}_k$. $\square$

Domain Decomposition Controlled by a Background Mesh

Another way for controlling the size of the elements is to use a background mesh. This approach is common when error estimations on an existing mesh are used to govern the creation of a new mesh. We use an unstructured background mesh $G = \{g_i\}$, where each of its nodes g_i is assigned a sizing value $f(g_i)$. This value determines the element size at the neighborhood of the node. In the cases where the sizing value is assigned to the elements,

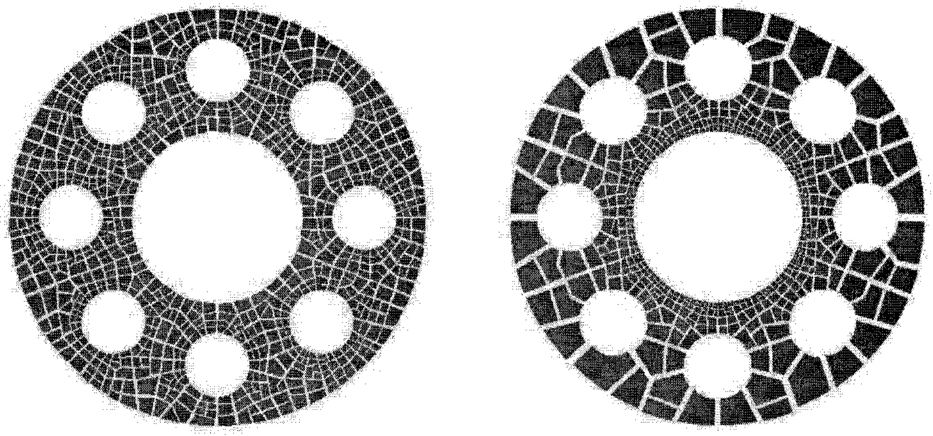


Figure 7.1: **Left:** The Pipe domain decomposed into 804 subdomains using a sizing function that corresponds to sources at the centers of the holes. **Right:** The Pipe domain decomposed into 487 subdomains using a background mesh, the element size being smaller near the center.

instead of the nodes, of the background mesh, we can use an interpolation procedure to obtain sizing values on the nodes.

As in the case of a function, we assign to each subdomain D two values, $m(D) = \min\{f(g) | \ g \in D \cap G\}$ and $M(D) = \max\{f(g) | \ g \in D \cap G\}$. The decomposition should satisfy the conditions 1 and 2, stated in the begginning of this section. The procedure described by Algorithm 7.1 will produce such a decomposition for a background mesh. There are though two questions we should answer, in order to show that this algorithm can be used for a background mesh: 1. What the values $m(D)$ and $M(D)$ should be, when no mesh nodes of G are in D? 2. The termination of the algorithm in the case of a continuous function f is based on the continuity of f; can it be guaranteed in the case of the background mesh? Both questions are addressed by employing an interpolation scheme, which we use when no node of G is contained in a subdomain D.

Subdomain Interpolation Procedure. When no background node is contained in a subdomain D, then the minimum and the maximum element size in D will be computed using interpolation. Let D such a subdomain, with $D \cap G = \emptyset$. We compute the desirable area of the elements in D by geometric interpolation, using the values of its neighboring subdomains. Let $m_1 = \min m(D')$ and $m_2 = \max m(D')$, where the minimum and maximum values are taken over all the neighboring (sharing common boundary) subdomains D' of D. Then we assign $m(D) = M(D) = \sqrt{m_1 m_2}$. We choose the geometric mean to compute the new values because it best complies with the nature of conditions 1 and 2. Specifically, the value $\max \frac{m(D_1)}{m(D_2)}$, where $D_1, D_2 \in \{D, D'\}$, is minimized when $m(D)$ is obtained by the geometric mean. Moreover, the geometric interpolation induces a continuous function in the following sense: as the size of the decomposition grows, the values $\frac{m(D_i)}{m(D_j)}$ and $\frac{M(D_i)}{M(D_j)}$ for neighboring subdomains tend to 1. In other words, the discrete sizing values given by the geometric interpolation procedure approximate a continuous function, and following the arguments in Section 7.3, Algorithm 7.1 terminates.

7.4 The Graded Delaunay Decoupling Path

Let $\mathcal{H}$ be the set of the piecewise linear separators produced by the domain decomposition procedure. The decoupling path is constructed by refining the initial separators $\mathcal{H}$, so that they form a decoupling path $\mathcal{P}$. The termination conditions of the Delaunay mesh generation allow us to compute a length size that should be used for refining $\mathcal{H}$ into a decoupling path $\mathcal{P}$. For uniform meshes it is we have proved in 15 that a decoupling path can be constructed, allowing the Delaunay meshes to be generated independently. We restate the theorem in comprehensive form, which will be useful for developing the ideas on the graded decoupling method.

Theorem 19. *Let* $k = \min\{lfs_{\min}(\Omega_{\mathcal{H}}), \frac{1}{2}\sqrt{\frac{A}{\sqrt{2}}}\}$*, where* $lfs_{\min}(\Omega_{\mathcal{H}})$ *is the minimum local feature size of* $\Omega \cup \mathcal{H}$ *and* A *is a constant bounding below the maximum triangle area. If for all the edges* $E \in \mathcal{P}$ *of the refined separators we have* $\frac{2}{\sqrt{3}}k \leq |E| < 2k$*,* $|E|$ *being the length of* E*, then* $\mathcal{P}$ *is a decoupling path with respect to Ruppert's algorithm, under the constrains of maximum circumradius to shortest edge ratio less or equal to* $\sqrt{2}$ *and maximum triangle area bound greater or equal to* A*.*

Theorem 19 assumes that the triangle area bound A is constant, and thus cannot be applied as is in the case of graded meshes. It can still be used though in the graded case, for the construction of the decoupling path in the following way. Let $\mathcal{D} = \{D_i\}$ a decomposition of Ω and $m(D_i)$ as defined in Sections 7.3 and 7.3. We will assume that the sizing function captures the minimum local feature size in the following way: $\frac{1}{2}\sqrt{\frac{m(D_i)}{\sqrt{2}}} \leq \mathrm{lfs}_{\min}(D_i)$. In Section 8.2 we describe a procedure that constructs such a sizing function[1]. In the following we will discuss the decoupling procedure under a sizing function bound, omitting the minimum local feature size. We can apply Theorem 19 on each individual subdomain D_i, obtaining the following result.

Proposition 20. *Let $k_i = \frac{1}{2}\sqrt{\frac{m(D_i)}{\sqrt{2}}}$. If for all the edges $E \in \mathcal{P} \cap D_i$, that belong to the internal boundary of D_i, we have $\frac{2}{\sqrt{3}}k_i \leq |E| < 2k_i$, then the edges of $\mathcal{P} \cap D_i$ will remain invariant after applying Ruppert's algorithm on D_i, with the constrains of maximum circumradius-to-shortest-edge ratio equal to $\sqrt{2}$ and maximum triangle area bound greater or equal to $m(D_i)$.*

Each separator E of $\mathcal{P}$ is shared by two subdomains, and in order to prove that the whole set of separators $\mathcal{P}$ forms a decoupling path, we have to examine if E remains invariant after applying the mesh generator independently to both subdomains. By applying Proposition 20 to each of the neighboring subdomains we obtain the following result.

Proposition 21. *Let $E \in \mathcal{P}$ be any edge of the separators, with $E \in D_i \cap D_j$ and length $|E| = l$. If both relations $\frac{2}{\sqrt{3}}k_i \leq l < 2k_i$ and $\frac{2}{\sqrt{3}}k_j \leq l < 2k_j$ hold (for k_i, k_j as defined in Proposition 20), then $\mathcal{P}$ is a decoupling path.*

Figure 7.2 depicts a graded Delaunay mesh created by decoupling the subdomains, and also the decoupling zone for one subdomain.

We proceed to examine the prerequisites under which the hypothesis of the above proposition is true. Let D_i, D_j be two neighboring subdomains; without loss of generality we assume $k_i \leq k_j$. Then there exists l that satisfies both conditions of Proposition 21, if and only if, $\frac{k_j}{k_i} < \sqrt{3}$. If $\frac{k_j}{k_i} \geq \sqrt{3}$, it is obvious that no such l exists. On the other hand, if $\frac{k_j}{k_i} < \sqrt{3}$, then there is such l that satisfies both conditions (for example we can choose $l = k_j$). More general, for any $l = \sqrt{3}k_i - \epsilon$, with $0 < \epsilon \leq \sqrt{3}k_i - k_j$, both conditions are true. From the definition of k_i, k_j we observe that $\frac{k_j}{k_i} < \sqrt{3} \Leftrightarrow \frac{m(D_j)}{m(D_i)} < 3$. Thus, by taking $R_2 < 3$ in Condition 2, Section 7.3, the relation $\frac{k_j}{k_i} < \sqrt{3}$ holds, and thus the decoupling path $\mathcal{P}$ exists.

[1]Constructions of sizing functions that capture the local feature have been studied in the past, mostly in the context of advancing front methods. [90, 68, 70].

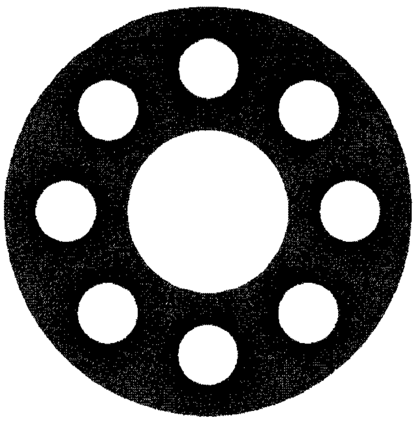
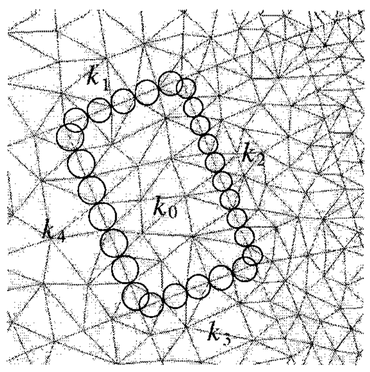



Figure 7.2: **Left:** Graded Delaunay mesh based on decoupling the subdomains. The sizing function reflects sources at the centers of the holes. **Right:** Detail of the mesh; the decoupling zone $\mathcal{Z}_{\mathcal{P}}$ for one subdomain is depicted by the circles.

7.5 Construction of the Graded Delaunay Decoupling Path

The condition $R_2 < 3$ allows the theoretical existence of a decoupling path, but we have to take into account that the decoupling path $\mathcal{P}$ will be constructed by refining the existing separators $\mathcal{H}$, which were created by the domain decomposition procedure. Let $E' \in \mathcal{H}$ be an edge of the separator shared by the subdomains D_i, D_j, which must be refined, so that the resulting subsegments satisfy the conditions of Proposition 21. The refining procedure will break E' into, say, ν subsegments. Then the conditions $\frac{2}{\sqrt{3}}k_j \le \frac{|E'|}{\nu} \Leftrightarrow \nu \le \frac{\sqrt{3}|E'|}{2k_j}$ and $\frac{|E'|}{\nu} < 2k_i \Leftrightarrow \nu > \frac{|E'|}{2k_i}$ must hold, where $k_j \ge k_i$. In other words, an integer value should exist between the values $\frac{|E'|}{2k_i}$ and $\frac{\sqrt{3}|E'|}{2k_j}$. A sufficient condition for the above relation to be true is $\frac{\sqrt{3}|E'|}{2k_j} - \frac{|E'|}{2k_i} \ge 1$. In result, we have for the length $|E'|$ the condition

$$|E'| \ge \frac{2k_j k_i}{\sqrt{3}k_i - k_j} = \frac{2k_j}{\sqrt{3} - \frac{k_j}{k_i}} \tag{7.1}$$

in order for the created separators to satisfy the above relation, we have to keep the denominator of the right side fraction bounded below. This can be done by defining the R_2 constant to be small enough. In our experiments we use the value $R_2 = 1.5$, so that the denominator is always greater than 0.5. Then, the relation 7.1 is satisfied if

$$|E'| \ge 4k_j.$$

The decomposition is controlled by the gradation of the sizing function, and not by the sizing values, so it is invariant when we decrease the sizing function by a constant factor. While the values $|E'|$ are kept invariant, the values k_j decrease, and thus, for large meshes, the relation 7.1 holds.

We sum our results for constructing the decoupling path in the following theorem.

Theorem 22. *Let the relation 7.1 hold for all separators $E' \in \mathcal{H}$, which were created by the domain decomposition procedure. Then the refined set of separators $\mathcal{P}$ is a decoupling path for Ruppert's algorithm, with the constrains of maximum circumradius to shortest edge ratio $\sqrt{2}$, and maximum triangle area bounded by the sizing function f.*

Proof. The above discussion shows that relation 7.1 guarantees that the refinement of the separators will satisfy the hypothesis of Proposition 21. The conclusion is driven by Proposition 21. □

The above theorem allows the creation of graded meshes in parallel and with no communication, since the subdomains can be meshed independently after they have been decoupled. The final mesh will be globally Delaunay, satisfying the area constrains defined by the sizing function, as well as the quality constrain of having maximum circumradius to shortest edge ratio less or equal to $\sqrt{2}$.

7.6 The Graded Delaunay Decoupling Procedure

We implemented the graded decoupling method, as it is described in Sections 7.2 through 7.5[2]. We use a master/worker scheme for the parallel decoupled Delaunay mesh generation procedure. The master processor reads the domain Ω and over-decomposes it (i.e. we create $N >> P$ subdomains, where P is the number of processors). The subdomains are queued in reverse order of their expected mesh size. The master controls the assignment of the subdomains to the processors following a greedy approach. The next subdomain to be processed is sent to the next free worker processor. The procedure is described by Algorithm 7.2

[2]The sizing function is assumed to capture the geometric features of the domain.

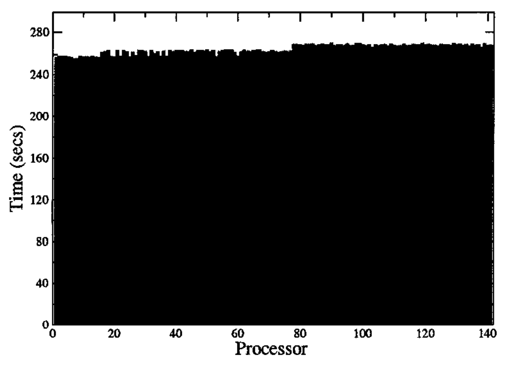
Algorithm 7.2.

```
    Master Processor:
1.      read the definition of the domain $\Omega$
2.      create the gradation cotrolled MADD
3.      create a sorted queue of the subdomains in reverse area order
4.      while the queue id not empty do
5.          send the next subdomain to the next free processor
6.          receive replies from completed meshes
7.      endwhile
8.      wait until all processors have finished
9.      send termination requests

    Worker Processors:
10.     while not terminate do
11.         receive subdomain from Master
12.         decouple the subdomain
13.         apply the sequential mesh generator on the subdomain
14.         send completed reply to Master
15.     endwhile
```

The subdomains structure contains the information (i.e. the k_i) for the decoupling procedure, so each worker processor can decouple independently the received subdomain. A parameter allows the subdomains to be packed into groups, reducing the communication (and the workload for the master processor). Moreover, each processor maintains a work buffer allowing asynchronous communication, and thus minimizing the communication cost

The decomposition is performed sequentially by the master processor and is controlled by the gradation as described in Section 7.3. An important parameter that affects the parallel performance is the good balance of the work-loads among the processors. Over-decomposition of the domain, i.e. creating much more subdomains than the number of processors, has proved to be an effective approach [52]. This approach allows work-load differences for processing each subdomain to be absorbed, by assigning a set of subdomains to each processor. Over-decomposition though is less effective when the work-loads for some of the subdomains are much larger than the average work-load of all the subdomains. Moreover, the created meshes for each subdomain should fit into the avaliable memory. An additional condition of bounding the subdomain area is applied, in order to bound the workload for each subdomain and also the memory requirements. This condition is



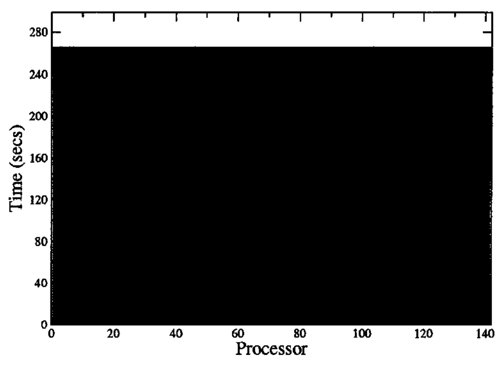


Figure 7.3: Load balance on heterogenous environment of 141 cpus. **Left:** The load balance for the d function. The decomposition is 2,064 subdomains and the created mesh is 5 billion elements. **Right:** The load balance for the d^4 function. The decomposition is 19,847 subdomains and the created mesh is 5 billion elements.

formulated as follows.

Condition 3. *For a predefined constant T, that designates the maximum number of elements per subdomain, we should have*

$$|D| \leq m(D_i)T,$$

where D_i is any subdomain, and $|D_i|$ denotes the area of D_i.

The above condition can be met by further decomposing the subdomains that do not satisfy it. Following the arguments of Section 7.3, the decomposition procedure will terminate. The constant T depends on the machines to be used.

The assignment of the subdomains is done on the fly in greedy way, resulting a *dynamic load-balance* greedy scheme. This approach is effective, even for heterogenous environments, provided we have a large enough over-decomposition. Figure 7.3 depicts the load balance for two different decompositions. The load balance on the left is for 2,064 subdomains, and although is good, it is not perfect. The load balance on the right figure is for a much larger decomposition, 19,847 subdomains where used, and it is almost perfect. Of course, higher over-decomposition implies a higher overhead cost, and also higher communication cost. A study of optimal load balancing strategies, while keeping the overhead and communication cost small, is part of our future work.

7.7 Performance Evaluation

The meDDec software [56] implements the parallel graded Delaunay decoupling procedure. It is written in c99 standard C using the LAM/MPI library. The Triangle library [77, 86] was used for the creation of the Delaunay triangulation during the MADD procedure. Triangle was also used as the off-the-shelf sequential mesh generator on each subdomain for the parallel decoupled Delaunay mesh generation. The Metis library [45, 57] was used for the graph partitioning step in the MADD procedure.

We ran three sets of experiments. A sequential set of experiments was performed to assess the stability of the decoupling method, and specifically the resulting over-refinement. A set of parallel experiments was performed on a homogenous environment in order to assess the efficiency of the method, and in particular the parallel speedup. Another set of parallel experiments was performed on a heterogenous environment in order to examine the efficiency of the method on an environment consisting of machines with different processing power and memory.

Experimental Set-up. The domain used for our experiments is the *Pipe* model (Figs. 7.2, 7.4), which is an approximation of cross section of a rocket geometry. We tested the performance for four sizing functions. The function f_s reflects sources at the centers of the holes and is analogous to fourth power of the distance of the centers (see Fig. 7.2 left). The functions d, d^2 and d^4 are analogous to the distance from the inner hole, raised to the power of one (Fig. 7.4 left), two and four (Fig. 7.4 right), respectively. The gradation constant R_2 was set to 1.5, while R_1 was set to 1.425.

Our experiments were performed on the SciClone cluster [74]. For the homogenous environment experiments we used the tempest subcluster, consisting of 32 dual cpus at 2.4 GHz, 4 GB memory. The heterogenous environment is composed by the subclusters whirlwind (64 single cpus, 650 MHz, 1 GB memory), twister (31 dual cpus, 900 MHz, 2 GB memory) and vortex (4 quad cpus, 1.28 GHz, 8 GB memory), giving a total of 142 cpus.

Sequential Experiments. We have ran a set of sequential experiments to observe the number of additional elements created by the decoupling procedure for different sizing

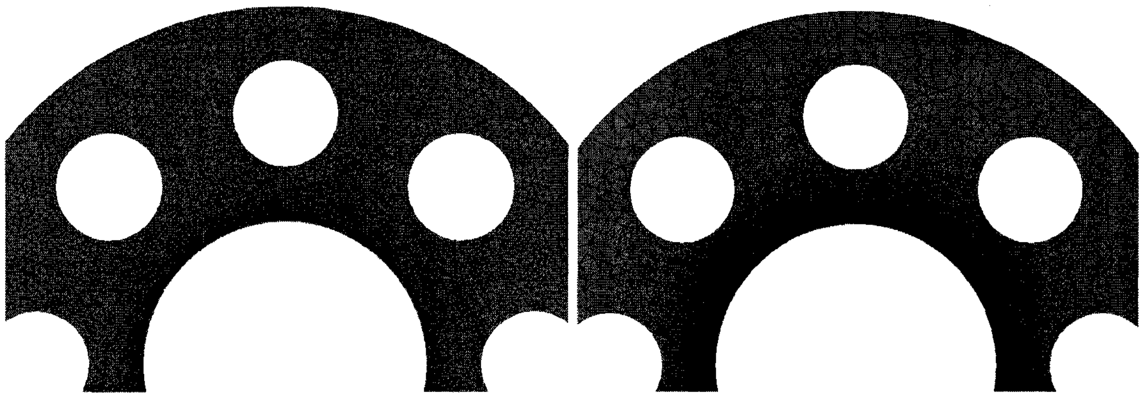


Figure 7.4: Part of the Pipe domain meshed by the decoupling procedure according to two sizing functions. **Left:** The element size is given by the function d, which is analogous to the distance from the inner hole. **Right:** The element size is governed by d^4, which is analogous to the fourth power of the distance from the inner hole.

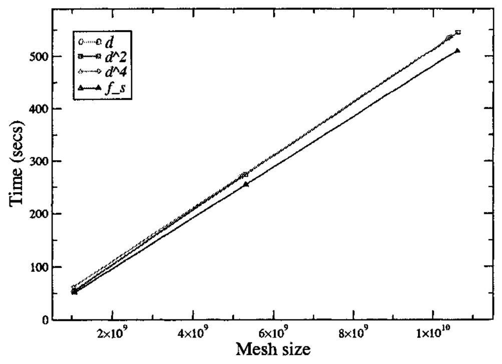
functions. The results are described in Table 7.1. The over-refinement is analogous to the length of the separators, which in turn is analogous to the number of the subdomains. The gradation of the sizing function controls the decomposition, and the number of the created subdomains increases, as the local gradation gets larger. The over-refinement is relatively small, even for sizing functions that show large gradation. For the function d^4, the global gradation is 1/707281, while the additional elements after decoupling are 2.28% of the non-decoupled sequentially generated mesh size.

Size Function	Sub-domains	Triangle elements	Decouple elements	% Add. elements	Global Size gradation
d	1310	69221990	69625612	0.58	1/29
d^2	4965	70787036	71685252	1.27	1/841
d^4	19448	69614458	71198934	2.28	1/707281
f_s	10214	70761174	72032140	1.80	1/77

Table 7.1: The number of additional elements created by the decoupling procedure, as compared to the elements created by the sequential, non-decoupled, procedure.

Parallel Experiments. The time performance of the decoupled mesh generation procedure in the heterogenous environment is depicted in Figure 7.5. The times are independent of the sizing function, and appear to be linear in terms of the created mesh size.

The performance for the homogenous environment is presented in Table 7.2. The results show that we can create 2 billion elements in less than one minute. The speedup is depicted



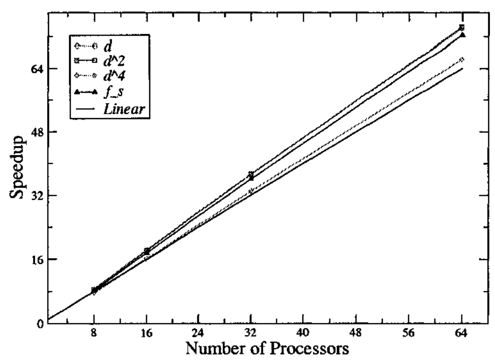


Figure 7.5: The time performance for the heterogenous environment (142 cpus).

Figure 7.6: The speedup for the homogenous environment.

Sizing function	d	d^2	d^4	f_s
Mesh Size	10.4 B	10.6 B	10.4 B	10.6 B
Subdomains	2,526	5,086	19,839	10,404
Decomposition Time	1.23	2.55	14.4	5.92025
Meshing Time	277.57	278.76	271.58	288.801

Table 7.2: Performance results for the homogenous environment (64 cpus). The times are in seconds and the mesh size is in billions of elements. The meshing time includes the decomposition read and distribute time.

in Figure 7.6. We have created about 81 million elements per processor, and calculated the speedup against the sequential run of Triangle for a mesh of 30 million elements (with no disk swapping). The parallel times include the decomposition cost. The decoupling procedure gives super-linear speedup, a result commonly observed for decoupling approaches. This is due to the slightly non-linear time of the mesh generation procedure, and probably because of the larger accumulative cache size. Moreover, we observe better speedup as we increase the number of processors. This is explained by the fact that we always define one processor to be the master, and dedicate it to control the mesh generation procedure.

Chapter 8

The Decoupling Method for Domains with Small Angles

In our study of the graded decoupling method in the previous chapter we have assumed the sizing function to capture the geometric features of the domain, namely the local feature size. Furthermore, we have not addressed the problem of generating meshes for domains with small angles. Ruppert's algorithm is guaranteed to terminate, if the angles of the domain are larger than 60°. Modifications of the algorithm guarantee the termination of the procedure when arbitrary small input angles are present. These modifications though alter the behavior of the algorithm near the boundary, and also the termination conditions, i.e. the minimum triangle edge size. In this chapter we address the problem of integrating the geometric features of the domain into the sizing function, and we enable the graded decoupling procedure to be applied for domains with small angles.

Ruppert describes in [72] the Delaunay refinement algorithm, and also a procedure to handle small input angles. The geometry is pre-processed, and protecting circles with radius $\mathrm{lfs}(p)/3$ are centered at the vertices p of the small angles. The small angles are "shielded" by shield edges defined by the protecting circles and the edges of the small angles. The domain outside the shielded triangles can be meshed by the standard refining algorithm, while the shielded triangles can be refined using templates. This procedure will work for domains with holes[1], which is this case in our study. Shewchuk describes a "Terminator" algorithm [80] based an concentric circular cells, which always terminates and guarantees a circumradius to shortest edge ratio lower bound of $1/[\sqrt{2}\sin(\theta/2)]$, where θ is the smallest input angle.

[1]It may not work for domains with internal boundaries [80].

Cohen-Steiner et al. [26] describe a Delaunay triangulation procedure for 3D PLC domains with possibly small angles ($< 90°$ in the case of 3D). The procedure constructs protecting balls centered on the vertices, or along the edges of the domain. No points are inserted into the protecting balls, instead a "split-on-sphere" strategy is followed. Cheng and Poon [15] propose an elaborative procedure for meshing 3D polyhedra with small angles. A set of protecting spheres forms a protective buffer zone along the edges. The centers of the protecting spheres are on the edges and are used to refine them. The determination of these centers requires the calculation of their local feature size, and also their *local gap size*, which is the radius of the smallest ball intersecting two entities of the domain. The procedure provides quality and termination guarantees, but appears to be impractical [14]. A more practical algorithm for 3D Delaunay mesh generation is presented in [14] by Cheng et al. Small angles are protected by balls, and as in [26], a split-on-sphere strategy is applied to protect the elements defined by the vertex balls.

The above approaches require the computation of the local feature size. A 3D Delaunay mesh generation method, for domains with small angles, that does not require an a priori computation of the local feature size, is described in [65] by Pav and Walkington. The algorithm builds-up information about the local feature size, starting from an initial Delaunay tetrahedralization, which provides the distances between closest input vertices. A "grooming" procedure refines the input edges according to this information. A set of arcs is constructed along the refined edges, so that adjacent arcs meet at obtuse angles. Finally, the tetrahedralization is refined, while the arcs are split by a split-on-sphere strategy. The edges may need to be refined further, creating new arcs.

8.1 A Shielding Procedure for Small Input Angles

The method we describe in this section shield the small input angles ($< 60°$), and is suitable for geometric domain decomposition based parallel Delaunay 2D mesh generation. The subdomains with small angles are preprocessed, and the small angles are shielded via shield edges (see Fig. 8.1), in a similar way to Ruppert's approach. The shield edges form isosceles triangles that include the small angles, and are guaranteed to be invariant when we apply

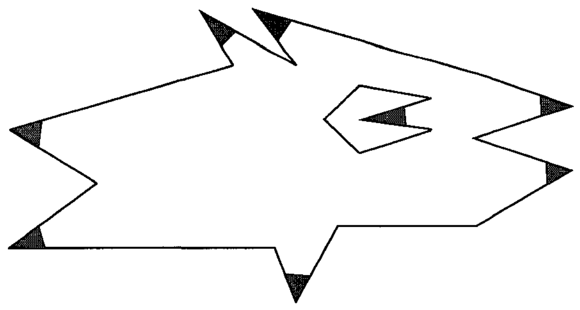


Figure 8.1: A subdomain D, enclosed by a set of boundaries with small angles. The shielded triangles T_D are depicted by the shaded areas. The remaining domain $D - T_D$ does not include angles less than 60°.

Ruppert's algorithm to the rest of the subdomain. For the construction of the shield edges we follow an analogous approach to the decoupling procedure. The minimum local feature size will be used to determine their size. In addition, a sizing function can be used to bound the areas of the shielded triangles. The procedure we propose is embarrassingly parallel, and provides termination and quality guarantees.

The decomposition $\mathcal{D} = \{D_i\}$ of the domain Ω is the starting point of the algorithm. Each subdomain D_i will be preprocessed independently, and the small angles will be shielded. Provided that the decomposition does not create angles less than 60°, as it is the case for the MADD, we only have to consider subdomains D_i that include external boundaries. Moreover, no subdomain can be a triangle with two small angles. Indeed, at least one edge of a triangular subdomain must be a separator, thus the two angles it forms will be greater than 60°. In the following discussion we will only consider subdomains that are *not triangles with two small angles.* We remind the reader that the domain Ω may include holes, but not internal boundaries (Definition 2).

An alternative definition of the minimum local feature size will be employed, which only allows to take into account the features inside the domain.

Definition 7. *The* minimum local feature size $lfs_{\min}(\Omega)$ *of a domain Ω is defined as the minimum distance between two non incident entities, when the straight line connecting these entities is in Ω (including the boundary).*

This definition does not alter the proof of termination of Ruppert's algorithm, neither the previous proofs we have given. Let D be a subdomain that includes some small angles.

We first describe the algorithm when no sizing function is used for the mesh generation. The initial step is to compute the minimum local feature size of D,

$$l = \mathrm{lfs}_{\min}(D).$$

This can be done even by a $\mathbf{O}(n^2)$ brute force approach, since the size of the subdomain is small[2]. Let $0 < \theta = \widehat{bac} < 60°$ be a small angle of D (see Fig. 8.2). We insert points b', c' in the segments ab and ac respectively, such that

$$|ab'| = |ac'| = \frac{l}{2\sin(\theta/2)} \frac{1}{(\cos(\theta/2) + \sin\theta)}. \tag{8.1}$$

We insert the shield edge $b'c'$ in the subdomain D, which forms the shielded isosceles triangle $b'ac'$. Observe that this insertion does not affect the decomposition, since ab and ac will not be part of a separator. The procedure is repeated for all small angles in D, and for all subdomains D that include external boundaries. We summarize the process in Algorithm 8.1

Algorithm 8.1.
```
1.  for all subdomains $D$ with external boundaries do
2.      calculate $l = \mathrm{lfs}_{\min}(D)$
3.      for all angles $\theta = \widehat{bac} < 60°$ in $D$ do
4.          insert points $b', c'$ in the segments $ab$ and $ac$ such that
5.          $|ab'| = |ac'| = \frac{l}{2\sin(\theta/2)} \frac{1}{(\cos(\theta/2)+\sin\theta)}$
6.          form the shielded triangle $t = b'ac'$
7.          add triangle $t$ to the list $T_D$
8.      endfor
9.      form the new subdomain $D' = D - T_D$
10. endfor
```

Let T_D the set of the shielded triangles, after we have processed all small angles of D. Let $D' = D - T_D$ be the new subdomain, without the shielded triangles. Then the following proposition holds.

Lemma 23. *Let $l = lfs_{\min}(D)$ be the minimum local feature size, as defined in 7, of a subdomain D. Let the subdomain D include some angles $\theta < 60°$. After applying Algorithm*

[2]Typically, the subdomains will not have more than a few hundred points.

8.1 on D, we obtain a new subdomain D', with no angles $< 60°$. Then, the minimum local feature size $l' = lfs_{\min}(D')$ of the processed subdomain D' satisfies the following relations:

$$l' = \min_{\theta<60°} \frac{l}{\cos(\theta/2)+\sin\theta}, \tag{8.2}$$

and

$$\frac{l}{\sqrt{3}} < l' < l. \tag{8.3}$$

Proof. In order to compute the new minimum local feature we only have to examine the new features induced by the points b', c' and the edge $b'c'$ (see Fig. 8.2). These features will be the edge $b'c'$, the edges $b'b$, $c'c$, or created from the edge $b'c'$ and some point q not on $b'b$, $c'c$. Finally we must examine the case of two inserted points that do not form a shield edge.

Thus we need to (i) calculate the length $|b'c'|$, (ii) calculate the lengths $|b'b|$, $|c'c|$, (iii) examine the case of a point q that may create with $b'c'$ a new minimum local feature size, and (iv) examine the distance between two inserted points which do not form a shield edge.

(i) The length $|b'c'|$. We have (see Fig. 8.2 left)

$$|b'c'| = 2\sin(\theta/2)\cdot|ab'|.$$

From the construction we have $|ab'| = |ac'| = \frac{l}{2\sin(\theta/2)}\frac{1}{(\cos(\theta/2)+\sin\theta)}$ (relation 8.1), and so

$$|b'c'| = 2\sin(\theta/2)\cdot\frac{l}{2\sin(\theta/2)}\frac{1}{(\cos(\theta/2)+\sin\theta)} \Rightarrow$$

$$|b'c'| = \frac{l}{(\cos(\theta/2)+\sin\theta)}.$$

(ii) The lengths $|b'b|, |c'c|$. Without loss of generality we assume that $|b'b| \leq |c'c|$. Let q be the projection of b on AC. Then $|bq| \geq l$. We have

$$|ab| = \frac{|bq|}{\sin\theta} \geq \frac{l}{\sin\theta}.$$

So,

$$|b'b| = |ab| - |ab'| \geq \frac{l}{\sin\theta} - \frac{l}{2\sin(\theta/2)}\frac{1}{(\cos(\theta/2)+\sin\theta)} \Rightarrow$$

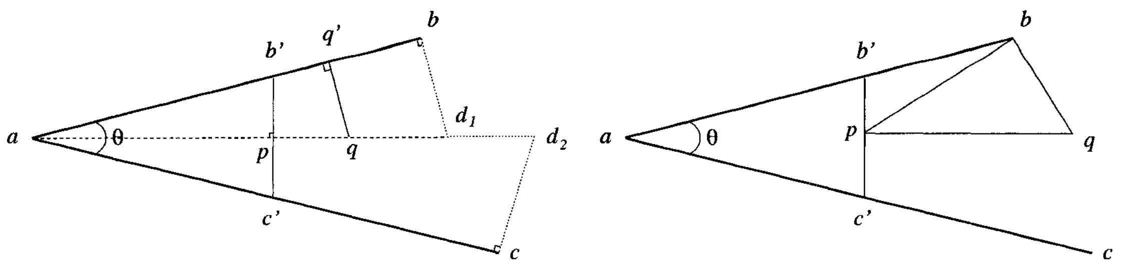


Figure 8.2: **Left:** The distance from $b'c'$ of a point q inside $b'bd_1d_2cc'$ cannot be less than $|b'c'|$. **Right:** The distance from $b'c'$ of a point q outside $b'bd_1d_2cc'$ cannot be less than $|b'c'|$.

$$|b'b| \geq \frac{l}{\sin\theta}\left(1 - \frac{1}{1+2\sin(\theta/2)}\right) = \frac{l}{(\sin\theta)}\frac{2\sin(\theta/2)}{(1+2\sin(\theta/2))} \Rightarrow$$

$$|b'b| \geq \frac{l}{\cos(\theta/2)+\sin\theta}.$$

Obviously the same relation holds for the length $|c'c|$.

(iii) Arbitrary existing point q. We will show that no existing point q of the subdomain D can be at distance less than $\frac{l}{(\cos(\theta/2)+\sin\theta)}$ from the shield edge $b'c'$. We will examine two cases: (a) the point q is inside the area enclosed by $b'bd_1d_2cc'$ (see Fig. 8.2 left), and (b) q is outside this area (Fig. 8.2 right).

Case (a) Let q be a point exists inside the area enclosed by $b'bd_1d_2cc'$. Let p be the projection of q onto $b'c'$. We will prove by contradiction that the distance $|qp|$ of q from the shield edge $b'c'$ must be greater than $|b'c'| = \frac{l}{\cos(\theta/2)+\sin\theta}$. Suppose this is not true, and $|qp| \leq \frac{l}{\cos(\theta/2)+\sin\theta}$. Without loss of generality, we assume th point q to be closer to to the segment ab than to segment ac, or equidistant. Take the projection q' of q on the segment ab, or ac, such that the distance $|qq'|$ is minimum. The length $|qq'|$ is maximized when the segment aq bisects θ, and this is the case we will examine. Then the length $|aq|$ is

$$|aq| = |ap| + |pq| = \cos(\theta/2)|ab'| + |pq| \leq \cos(\theta/2)|ab'| + \frac{l}{(\cos(\theta/2)+\sin\theta)} \Rightarrow$$

$$|aq| \leq \cos(\theta/2)\frac{l}{2\sin(\theta/2)}\frac{1}{(\cos(\theta/2)+\sin\theta)} + \frac{l}{(\cos(\theta/2)+\sin\theta)} \Rightarrow$$

$$|aq| \leq \frac{l}{\cos(\theta/2)+\sin\theta}\left(\frac{\cos(\theta/2)}{2\sin(\theta/2)}+1\right).$$

Calculating the length $|qq'|$ we get

$$|qq'| = \sin(\theta/2)|aq| \leq \frac{l\sin(\theta/2)}{\cos(\theta/2)+\sin\theta}\left(\frac{\cos(\theta/2)}{2\sin(\theta/2)}+1\right) \Rightarrow$$

$$|qq'| \leq \frac{l}{\cos(\theta/2)+\sin\theta}\left(\frac{\cos(\theta/2)}{2}+\sin(\theta/2)\right).$$

For $0 < \theta < 60°$ we observe that

$$\frac{\cos(\theta/2)}{2}+\sin(\theta/2) < 1,$$

and also

$$1 < \cos(\theta/2)+\sin\theta < \sqrt{3}. \tag{8.4}$$

So we get

$$|qq'| < l,$$

which contradicts the fact that $l = \mathrm{lfs}_{\min}(D)$. Thus $|qp| > |b'c'| = \frac{l}{\cos(\theta/2)+\sin\theta}$.

*Case (b)*Next we examine the case of a point outside the area $b'bd_1d_2cc'$. We will show that $\widehat{pbq} > \widehat{bpq}$, and thus, from the law of sines, $|pq| > |bq|$. Indeed, observe that $\widehat{b'bq} > \pi/2$, and also from the result in (ii), $\widehat{b'bp} < \widehat{b'pb}$. So

$$\widehat{pbq} > \pi/2 - \widehat{b'bp} > \pi/2 - \widehat{b'pb} = \widehat{bpq}.$$

We conclude using relation 8.4,

$$|pq| > |bq| \geq l > \frac{l}{\cos(\theta/2)+\sin\theta}.$$

(iv) Inserted points that do not form a shield edge. For inserted points that do not form a shield edge we have to examine three cases:

(a) If two inserted points are not on incident lines, it is obvious that their distance will be greater than $l = \mathrm{lfs}_{\min}(D)$.

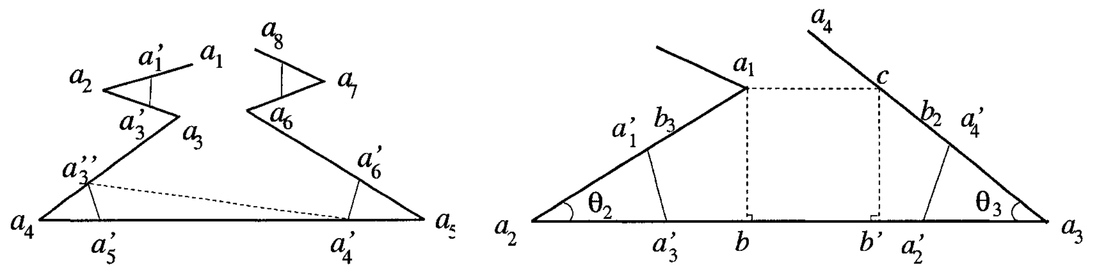


Figure 8.3: **Left:** The distance $|a_3''a_4'|$ is greater than $|a_3''a_5|$. **Right:** The distance $|a_1c|$ is less than the distance $|a_2'a_3'|$.

(b) If two inserted points belong on two incident lines, and do not form a shield edge, then their distance will be obviously greater than the length of the shield edge (see Fig. 8.3 left), and thus do not create the smallest feature. Here we note that we only consider distances inside the domain (Definition 7), and we do not have to examine distances like $|a_3'a_3''|$ in Fig. 8.3 left.

(c) We will examine the case when two points will be inserted on the same boundary edge (see Fig. 8.3 right). Let c be the intersection[3] of the line from a_1 parallel to the segment a_2a_3. We will prove that $|a_2b| > |a_2a_3'|$ and $|a_3b'| > |a_3a_2'|$. First we observe that $|a_1b| = |cb'| \geq l$. We have for $|a_2b|$

$$|a_2b| = |a_1b|\cot(\theta_2) \geq l \cdot \cot(\theta_2).$$

We examine the right hand part.

$$\begin{aligned}
l \cdot \cot(\theta_2) &\geq |a_2a_3'| \Leftrightarrow \\
l \cdot \cot(\theta_2) &\geq \frac{l}{2\sin(\theta_2/2)}\frac{1}{(\cos(\theta_2/2)+\sin\theta_2)} \Leftrightarrow \\
\frac{\cos\theta_2}{\cos(\theta_2/2)}(\cos(\theta_2/2)+\sin\theta_2) &\geq 1 \Leftrightarrow \\
\cos\theta_2(1+2\sin(\theta_2/2)) &\geq 1 \Leftrightarrow \\
(1-2\sin^2(\theta_2/2))(1+2\sin(\theta_2/2)) &\geq 1 \Leftrightarrow \\
1-2\sin^2(\theta_2/2)+2\sin(\theta_2/2)-4\sin^3(\theta_2/2) &\geq 1 \Leftrightarrow \\
-\sin(\theta_2/2)+1-2\sin^2(\theta_2/2) &\geq 0 \Leftrightarrow
\end{aligned}$$

[3]One of the two parallels, either from a_1 or from a_4, will intersect the opposite segment.

$$\cos\theta_2 - \sin(\theta_2/2) \geq 0.$$

Observe that $\cos\theta_2 - \sin(\theta_2/2)$ is a decreasing function, and equals zero at $\theta_2 = 60°$. Thus the last inequality is true for $0 \leq \theta_2 \leq 60°$. We obtain that

$$|a_2 b| \geq l \cdot \cot(\theta_2) \geq |a_2 a_3'|.$$

In a similar way we get

$$|a_3 b'| \geq l \cdot \cot(\theta_3) \geq |a_3 a_2'|.$$

Since $|bb'| \geq l$, we conclude

$$a_3' a_2' \geq bb' \geq l.$$

We summarize our results. From the relation 8.4 we have

$$\frac{l}{\sqrt{3}} < \frac{l}{(\cos(\theta/2) + \sin\theta)} < l.$$

From (i) we have that the shield edges have length $\frac{l}{(\cos(\theta/2)+\sin\theta)}$, while from (ii), (iii) and (iv), no new local feature size will be less than this value. So, $l' = \min_{\theta \in D} \frac{l}{\cos(\theta/2)+\sin\theta}$ and $\frac{l}{\sqrt{3}} < l' < l$. The proof of the lemma is complete.

□

The new subdomain D', without the shielded triangles, includes no angles less than 60°. So Ruppert's algorithm can be applied with the same quality and termination guarantees (Theorem 1, Section 1.1). Moreover, from relation 8.3 we have for any shield edge $b'c'$

$$l' \leq |b'c'| < \sqrt{3}l'.$$

Following the same argument as in the case of the decoupling procedure, no shield edges will be splitted. Thus, the resulting mesh will be a conforming Delaunay mesh with circumradius to shortest edge ratio at most $\sqrt{2}$ for all the triangles that do not include a small input angle.

The circumradius to shortest edge ratio for the shield triangles is

$$\frac{|ab'|/[2\cos(\theta/2)]}{|b'c'|} = \frac{1}{2\sin\theta}.$$

If $\theta \geq 20.7°$, we have $\frac{|b'c'|}{|ac'|} < \sqrt{2}$, and the quality of the whole mesh is the same as the one that Ruppert's algorithm guarantees.

Triangle area controlled by a sizing function. The circumradius to shortest edge ratio criterion corresponds to the smallest angle of the triangle in 2D, and thus cannot be controlled in the shielded triangles, for which the input angle cannot be improved. The only improvement that can be applied to the shielded triangles is to reduce their area by enforcing a sizing function. In the rest of the section we will expand the procedure of preprocessing the subdomains with small input angles, so that the all triangle areas are bound by a sizing function $f(x)$.

The area E of a shielded triangle $b'ac'$ is $E = |ap| \cdot |c'b'|/2$ (see Fig. 8.2 left). We calculated the length $|b'c'|$ in (i) of Lemma 23 as

$$|b'c'| = \frac{l}{(\cos(\theta/2) + \sin\theta)}.$$

From the same lemma in (iii) we calculated $|ap|$ as

$$|ap| = \cos(\theta/2)|ab'| = \frac{\cos(\theta/2)}{2\sin(\theta/2)} \cdot \frac{l}{(\cos(\theta/2) + \sin\theta)}.$$

We have for the area E of the shielded triangle $ab'c'$

$$\begin{aligned} E &= \frac{l}{(\cos(\theta/2) + \sin\theta)} \cdot \frac{\cos(\theta/2)}{4\sin(\theta/2)} \cdot \frac{l}{(\cos(\theta/2) + \sin\theta)} \\ &= \frac{l^2}{2} \cdot \frac{\cos(\theta/2)}{2\cos^2(\theta/2)\sin(\theta/2)} \cdot \frac{1}{[1 + 2\sin(\theta/2)]^2} \\ &= \frac{l^2}{2} \cdot \frac{1}{\sin\theta} \cdot \frac{1}{[1 + 2\sin(\theta/2)]^2}. \end{aligned}$$

We obtain

$$l = [1 + 2\sin(\theta/2)] \cdot \sqrt{2E\sin\theta}. \tag{8.5}$$

An upper area bound to the triangle area can be enforced, by enforcing an upper bound to l through the relation 8.5. If A is a constant upper bound to the triangle area, we know from Theorem 15 that no edge less than $\sqrt{\frac{A}{\sqrt{2}}}$ will be split by Ruppert's algorithm (unless it is greater or equal of two times the minimum local feature size).

If we take

$$l = \min_{\theta<60^\circ} \left\{ [1+2\sin(\theta/2)] \cdot \sqrt{2A\sin\theta}, \quad \sqrt{\frac{A}{\sqrt{2}}} \right\},$$

then the area E of the shielded triangle will be at most A, and the shield edges will be less than $\sqrt{\frac{A}{\sqrt{2}}}$. Simpler formulas can be obtained that satisfy both conditions, although they will not be tight[4].

Algorithm 8.2.

```
1.   let $\mathcal{D} = \{D_i\}$ be a decomposition of $\Omega$
2.   let $m(D_i) = \min_{x\in D_i} f(x)$, where $f(x)$ is a sizing function
3.   let $\mathcal{D}' = \{D_i \in \mathcal{D} \ / \ D_i$has external boundaries forming some angles $< 60^\circ\}$
4.   for all subdomains $D_i \in \mathcal{D}'$ do
5.      set $l_i = \min_{\theta<60^\circ}\{[1+2\sin(\theta/2)] \cdot \sqrt{2m(D_i)\sin\theta}$,
            $\sqrt{\frac{m(D_i)}{\sqrt{2}}}$, $\mathrm{lfs}_{\min}(D_i)\}$.
6.      for all angles $\theta = \widehat{bac} < 60^\circ$ in $D_i$ do
7.         insert points $b', c'$ in the segments $ab$ and $AC$ such that
8.         $|ab'| = |ac'| = \frac{l_i}{2\sin(\theta/2)} \frac{1}{(\cos(\theta/2)+\sin\theta)}$
9.         form shielded triangle $t = b'ac'$
10.        add triangle $t$ to the list $T(D_i)$
11.     endfor
12.     form the new subdomain $D_i' = D_i - T(D_i)$
13.     $\mathcal{D} = \mathcal{D} - \{D_i\} \cup \{D_i'\}$
14.  endfor
```

Let $f(x)$ be a sizing function defining a maximum triangle area in the subdomain D, and $m(D) = \min_{x\in D} f(x)$. We will take $A = m(D)$, and also

$$l = \min_{\theta<60^\circ} \left\{ [1+2\sin(\theta/2)] \cdot \sqrt{2m(D)\sin\theta}, \quad \sqrt{\frac{m(D)}{\sqrt{2}}}, \quad \mathrm{lfs}_{\min}(D) \right\}. \tag{8.6}$$

The previous proofs for Lemma 23 remain valid for this new value of l, since it will be less or equal to $\mathrm{lfs}_{\min}(D)$. In addition, the areas of the shielded triangles will be bounded by

[4]For example, we have $\sqrt{2}\sin(\theta/2) < [1+2\sin(\theta/2)]^2 \cdot 2\sin\theta$, and also $\sqrt{\sqrt{2}\sin(\theta/2)} < \sqrt{\frac{1}{\sqrt{2}}}$. when $\theta < 60^\circ$. So a value $l = \sqrt{A\sqrt{2}\sin(\theta/2)}$ would satisfy both the area bound and the edge no-splitting bound.

$m(D)$. The shield edges $b'c'$ will not be split, since we have

$$|b'c'| < 2 \cdot \text{lfs}_{\min}(D) \quad \text{and} \quad |b'c'| < \sqrt{\frac{m(D)}{\sqrt{2}}}.$$

We restate Algorithm 8.1 in Algorithm 8.2, to include an area bound enforced by an area function f. The preprocessing of the subdomains with small angles can obviously be done in parallel. Each subdomain $D_i \in \mathcal{D}$ can be processed independently, executing the steps 5 to 12 of Algorithm 8.2.

We summarize the results of this section in the following theorem.

Theorem 24. *Let D_i be a subdomain created by the MADD, which includes some angles angles $\theta < 60°$. Apply the steps 4-14 of Algorithm 8.2 on D_i, obtaining a subdomain D'_i. Define l_i as in line 5 of Algorithm 8.2, and $l'_i = lfs_{\min}(D'_i)$. Then the following propositions hold:*

1. *All angles in D'_i are $\geq 60°$, and thus Ruppert's algorithm can be applied on D'_i, with largest circumradius to shortest edge ratio $\sqrt{2}$, and maximum triangle area defined by the sizing function $f(x)$.*
2. $l'_i = \min_{\theta < 60°} \frac{l_i}{\cos(\theta/2) + \sin\theta}$.
3. $\frac{l_i}{\sqrt{3}} < l'_i < l_i$.
4. *For any shield edge $b'c'$ we have*

$$l'_i \leq |b'c'| < l'_i\sqrt{3}.$$

5. *No shield edge will be split by Ruppert's algorithm.*
6. *All shielded triangles will have area less. or equal to $m(D_i) = \min_{x \in D_i} f(x)$, where $f(x)$ is a sizing function.*
7. *The circumradius to shortest edge ratio for any shielded triangle having an angle $\theta < 60°$ is*

$$\frac{1}{2\sin\theta}.$$

This ratio is less than $\sqrt{2}$, if $\theta \geq 20.7°$.

8.2 Construction of the Sizing Function

In the discussion of the graded decoupling procedure we assumed the sizing function to capture the geometric features of the domain. In this section describe a smoothing procedure for constructing a new sizing function $F(x)$ from an existing one $f(x)$, so that $F(x)$ is less or equal to $f(x)$ and also takes into account the geometric features (the minimum local feature size and the small input angles). $F(x)$ is constructed so that it has bounded gradation inside each subdomain and along neighboring subdomains.

Let $f(x)$ be an initial sizing function, and $\mathcal{D} = \{D_i\}$ the decomposition of the domain Ω as described in Section 7.3. We define $m_f(D_i) = \min_{x \in D_i} f(x)$ and $M_f(D_i) = \max_{x \in D_i} f(x)$. Then, the decomposition will satisfy the gradation conditions

$$M_f(D_i) \leq R_1 m_f(D_i), \tag{8.7}$$

and

$$m_f(D_i) \leq R_2 m_f(D_j), \quad \text{for any neighboring subdomains } D_i, D_j, \tag{8.8}$$

with $R_1 \leq R_2$. These conditions allow the existence of a graded decoupling path when $R_2 < 3$ (Section 7.4).

Our goal is to construct a new sizing function $F(x)$ such that (i) $F(x) \leq f(x)$, (ii) $F(x)$ captures the the geometric features of the subdomains (the minimum local feature size and existing small angles), (iii) $F(x)$ has bounded gradation inside each subdomain, and (iv) has bounded gradation along neighboring subdomains. While it is straightforward to express conditions (i), (iii) and (iv) mathematically, some calculation is required for condition (ii). The first step is to compute a lower bound for the minimum Delaunay edges for each subdomain, taking into account the sizing function $f(x)$ and the presence of small angles. We define

$$m_i = \left\{ \frac{1}{2}\sqrt{\frac{m_f(D_i)}{\sqrt{2}}}, \ \operatorname{lfs}_{\min}(D_i) \right\}. \tag{8.9}$$

Having in mind the discusion in the previous section, and specifically the definition of l (relation 8.6), we further define

$$e_i = \min_{\theta<60^\circ} \left\{ \frac{[1+2\sin(\theta/2)]\cdot\sqrt{2m_f(D_i)\sin\theta}}{\sqrt{3}}, \frac{m_i}{\cos(\theta/2)+\sin\theta} \right\} \qquad (8.10)$$

when small angles $< 60^\circ$ are present, and

$$e_i = m_i, \qquad (8.11)$$

when no small angles are present in D_i. We have constructed e_i so that no Delaunay edge less than e_i will be created in the subdomain D_i, when the mesh is generated taking into account the sizing function f and the geometric features of D_i (and not those of neighboring subdomains). Let $m_F(D_i) = \min_{x\in D_i} F(x)$, and $M_F(D_i) = \max_{x\in D_i} F(x)$, be the extremes of our new sizing function inside a subdomain D_i. Then, from Proposition 20, the sizes of the Delaunay edges in D_i will be bounded below by

$$k_i = \frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}.$$

In order for the new sizing function to capture both f and the geometric features, it is sufficient to take $k_i \le e_i$. Thus, we will require

$$m_F(D_i) \le 4\sqrt{2}\cdot e_i^2. \qquad (8.12)$$

We now can express the four conditions for the new sizing function $F(x)$ in a mathematical formulation. $F(x)$ should satisfy the following relations.

(i) $F(x) \le f(x)$.

(ii) $m_F(D_i) \le 4\sqrt{2}\cdot e_i^2$.

(iii) $M_F(D_i) \le R_1 m_F(D_i)$.

(iv) $m_F(D_i) \le R_2 m_F(D_j)$, for any neighboring subdomains D_i, D_j.

We will transform locally $f(x)$ in each subdomain, obtaining a new sizing function $F(x)$, so that the above relations hold. Let $F_i(x) = F|_{D_i}(x)$ and $f_i(x) = f|_{D_i}(x)$ be the restriction on D_i of $F(x)$ and $f(x)$ respectively. Define

$$g_i = 4\sqrt{2}\cdot e_i^2, \qquad (8.13)$$

and also

$$F_i(x) = f_i(x)\frac{g_i}{m_f(D_i)}. \tag{8.14}$$

By the construction, $F_i(x)$ will satisfy relations (i), (ii), and (iii). Indeed, we have that

$$m_F(D_i) = m_f(D_i)\frac{g_i}{m_f(D_i)} = g_i = 4\sqrt{2}\cdot e_i^2, \tag{8.15}$$

and condition (ii) is satisfied. Condition (iii) is satisfied, because the same relation holds for f,

$$M_F(D_i) = M_f(D_i)\frac{g_i}{m_f(D_i)} \leq R_1 m_f(D_i)\frac{g_i}{m_f(D_i)} = R_1 m_F(D_i). \tag{8.16}$$

Finally, condition (i) is satisfied from the definition of e_i,

$$F_i(x) = f_i(x)\frac{4\sqrt{2}\cdot e_i^2}{m_f(D_i)} \leq f_i(x)\frac{m_f(D_i)}{m_f(D_i)} = f_i(x). \tag{8.17}$$

More work is needed in order for F to satisfy condition (iv). The construction of F is local for each domain, and no guarantees are provided for F along neighboring subdomains. A procedure that checks neighboring subdomains, and reconstructs F in the cases where condition (iv) does not hold, is required.

The subdomains D_i with the smaller sizes F_i may force neighboring subdomains D_j to decrease their required value g_j, in order to maintain the gradation bound. Since the propagation of small sizes can only happen from smaller to larger sizes, it implies a procedure that will result bounded gradation among neighboring subdomains. Let us order the subdomains D_i in decreasing order of g_i. Modification of the smaller area g_j will only be forced by neighboring subdomains D_i, with $i < j$. This fact is the basis of our algorithm. Let $\mathcal{D} = [D_i]$ be the sorted list of the subdomains in increasing order of the values g_i. For each subdomain D_i, in increasing order, let D_j be a subdomain adjacent to D_i with $g_j > R_2 g_i$. Obviously $j > i$. We set a new value $g_j = R_2 g_i$, and we reposition D_j in the ordered list, according to its new value. Still we have $j < i$, beacause $R_2 > 1$. We repeated the procedure until we have scanned all of the list. The procedure is described by Algorithm 8.3.

Proposition 25. *The constructed function $F(x)$ by Algorithm 8.3 satisfies conditions (i) - (iv).*

Algorithm 8.3.

```
        // compute the values $g_i$
1.      let $\mathcal{D} = \{D_i \ / \ i = 1, .., N\}$ be a decomposition of $\Omega$
        satisfying the conditions 8.7 and 8.8
2.      set $g_i = 4\sqrt{2} \cdot e_i^2$
3.      sort $\mathcal{D} = \{D_i \ / \ i = 1, .., N\}$, so that $g_i \leq g_j$ when $i < j$
4.      for $i$=1 to $N - 1$ do
5.        for all adjacent to $D_i$ subdomains $D_j$ with $i < j$ do
6.          if $g_j > R_2 \cdot g_i$ do
7.            set $g_j = R_2 \cdot g_i$
8.            reposition $D_j$ in the list $\mathcal{D}$ according to the new $g_j$
9.          endif
10.       endfor
11.     endfor
        // compute the sizing function $F$
12.     Set $F|_{D_i}(x) = f|_{D_i}(x)\frac{g_i}{m_f(D_i)}$
```

Proof. The value of g_i can only being changed in step 7 of the algorithm, and in this case will only be decreased. So, the Algorithm may only reduce the values of F. Thus, the relations obtained in 8.17 and 8.15 still hold, and the conditions (i) and (ii) are satisfied. As was shown by the calculation in 8.16, condition (iii) is true.

If a value g_j is changed in step 7, say when examining subdomain D_r in step 4, then it will not be changed again, because all the consequent subdomains D_i, $i > r$, examined in step 4 will have $g_i \geq g_r$. Now, for any subdomain D_i scanned in step 4, its value g_i will not be changed in the remaining steps. Also, from steps 6 and 7, all its neighbors will have values $g_i \leq R_2 g_j$. In the next steps, the values g_j will remain unchanged. So, after step 11, we will have for any neighboring subdomains D_i, D_j

$$m_F(D_i) = g_i \leq R_2 g_j = R_2 m_F(D_j),$$

and condition (iv) is satisfied. □

Steps 7 and 8 in Algorithm 8.3 will be executed at most once for each subdomain. The repositioning in step 8 will take at most $\mathbf{O}(N)$ operations, So, Algorithm 8.3 is of $\mathbf{O}(N^2)$ complexity, where N is the number of subdomains[5].

[5]The calculation of the e_i is a pre-processing step, and is not included in the analysis of the algorithm. The calculation of the e_i can be done in parallel, independently for each subdomain. The complexity of this

8.3 The Decoupling Method for Domains with Small Angles

The construction of the sizing function $F(x)$ in previous section targets the graded decoupling procedure, as it was described in Chapter 7. In this section we will verify that the decoupling method can be applied for the constructed sizing function $F(x)$. As previously, let $m_F(D_i) = \min_{x \in D_i} F(x)$, and $M_F(D_i) = \max_{x \in D_i} F(x)$. Then F satisfies the conditions *(iii)*

$$M_F(D_i) \leq R_1 m_F(D_i),$$

and *(iv)*

$$m_F(D_i) \leq R_2 m_F(D_j),$$

for any neighboring subdomains D_i, D_j. When $R_2 < 3$, we can construct a decoupling path for the sizing function $F(x)$, with an additional condition for the size of the separators[6] (Theorem 22). The only remaining issue that we have to check is that the sizing function indeed captures the minimum local feature size, and also allows the small input angles to be shielded in a way that does compromise the decoupling method.

First we confirm that the sizing function captures the minimum local feature size of the subdomains, i.e.,

$$\frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \leq \mathrm{lfs}_{\min}(D_i).$$

We know that $F(x)$ satisfies condition (ii), and so we obtain

$$m_F(D_i) \leq 4\sqrt{2} \cdot e_i^2 \;\Rightarrow\; \frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \leq e_i \leq m_i \leq \mathrm{lfs}_{\min}(D_i),$$

as we wanted.

We also need to verify that the small angles can be shielded, in the way described in Section 8.1, without compromising the decoupling procedure. The shielding procedure should satisfy the following properties. (a) The parameter l_i in step 5 of Algorithm 8.3 for

calculation depends on the way we evaluate the sizing function $f(x)$, and on the complexity of finding the $\mathrm{lfs}_{\min}(D_i)$.

[6]This condition can also be captured by the sizing function. We describe this more general construction of the sizing function for simplicity.

shielding the angles is not greater than the minimum local feature size of the subdomain. (b) The shield edges will not be split under the sizing function. (c) The shield edges will not create features not captured by the sizing function.

Let D_i be a subdomain with some small angles $\theta < 60°$. Let $\Phi_i = \max_{\theta<60°} \theta$ be the larger of the smallest angles in D_i. We define

$$l_i = \frac{\cos(\Phi_i/2) + \sin\Phi_i}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}.$$

We are reminded that in the presence of small angles the parameter e_i is defined by the relation 8.10

$$e_i = \min_{\theta<60°}\left\{\frac{[1+2\sin(\theta/2)]\cdot\sqrt{2m_f(D_i)\sin\theta}}{\sqrt{3}}, \frac{m_i}{\cos(\theta/2)+\sin\theta}\right\},$$

and also that $F(x)$ satisfies condition (ii)

$$m_F(D_i) \le 4\sqrt{2}\cdot e_i^2.$$

(a) From condition (ii) we have

$$\begin{aligned} l_i &= \frac{\cos(\Phi_i/2) + \sin\Phi_i}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \\ &\le [\cos(\Phi_i/2 + \sin\Phi_i]\cdot e_i \\ &\le [\cos(\Phi_i/2) + \sin\Phi_i]\frac{m_i}{\cos(\Phi_i/2)+\sin\Phi_i} \\ &= m_i \\ &\le \mathrm{lfs}_{\min}(D_i). \end{aligned}$$

So, the parameter l_i is less or equal to $\mathrm{lfs}_{\min}(D_i)$.

(b) The shield edges will not be split under the sizing function if their length is less than

$\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}$. We have for a shield edge $b'c'$ of a small angle θ that

$$\begin{aligned} |b'c'| &= \frac{\cos(\Phi_i/2) + \sin\Phi_i}{2(\cos(\theta/2) + \sin\theta)}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \\ &< \sqrt{\frac{m_F(D_i)}{\sqrt{2}}}, \end{aligned}$$

as we wanted.

(c) In order to ensure that a shield edge $b'c'$ will not create smaller features than the sizing function indicates, we have to show that $|b'c'| \geq \frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}$. We have

$$\begin{aligned} |b'c'| &= \frac{\cos(\Phi_i/2) + \sin\Phi_i}{2(\cos(\theta/2) + \sin\theta)}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \\ &\geq \frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}, \end{aligned}$$

as we wanted.

Finally, from the left part of the definition of e_i we observe that the area E of the shielded triangle is bounded by the initial sizing function $f(x)$, $E \leq m_f(D_i)$.

We summarize our conclusions:

Theorem 26. *Let $F(x)$ be the sizing function constructed by Algorithm 8.3. Let $\mathcal{D} = \{D_i\}$ be the set of subdomains, after they have been processed by Algorithm 8.2 for shielding the small angles, using*

$$l_i = \frac{\cos(\Phi_i/2) + \sin\Phi_i}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}}.$$

Then for any subdomain D_i we have that all its angles will be $\geq 60°$, and

$$\frac{1}{2}\sqrt{\frac{m_F(D_i)}{\sqrt{2}}} \leq lfs_{\min}(D_i).$$

The graded decoupling procedure is applicable to the decomposition $\mathcal{D}$, using the sizing function $F(x)$.

Chapter 9

Conclusions and Future Work

The Delaunay refinement procedure is memory intensive, with unpredictable computational behavior which depends on the input geometry and the element size requirements. The decoupling approach is a geometric decomposition based, parallel mesh generation method, that allows a sequential Delaunay mesh generator to be applied independently on each subdomain. The method allows the creation of large Delaunay meshes on parallel distributed memory environments, and at the same time eliminates the communication/synchronization. The decoupling method is effective and efficient, resulting super-linear speedups.

Parallel mesh generation procedures that are based on geometric domain decompositions require the permenant separators to be of good quality (in terms of their angles and length), in order to maintain the mesh quality. The *Medial Axis domain decomposition* we describe in this work provides domain decomposition of high quality, and it presents for the first time a decomposition method suitable for parallel meshing procedures.

Decoupling approaches have been studied in the past. However, the methods previously proposed lucked completeness; they either did not provide termination or quality guarantees, or they had to introduce communication. In this work we provide a mathematical formulation of the decoupling method that guarantees the termination of the procedure, and also the conformity and quality of the final mesh, without introducing communication. This formulation, initially given for uniform meshes on domains with no small angles, is extended for parallel graded mesh generation on domains with possibly small angles. The experimental results confirm the efficiency and stability of the decoupling procedure.

The procedure described in this work addresses the problem of parallel Delaunay mesh generation for 2D domains. The notions though of the *decoupling path* and the *decoupling zone* are defined for any dimesions, and their conformity and invariance properties (Propositions 7 and 8) remain true. Thus, these notions can form the basis for a 3D decoupling procedure. As we have mentioned above, 3D decoupling approaches have been described in the past, but with no quality guarantees. A procedure that guarantees the conformity and the quality of the mesh, based on the notion of the decoupling path, is feasible.

The two problems that we face in 2D also need to be addressed in 3D. The 3D domain decomposition should be of good quality in terms of the formed angles in order not to distort the mesh quality. The extention to 3D of the core medial axis domain decomposition algorithm is straight forward. Additional work though is required to obtain some theoretical indications about the angles formed, both with the boundary and among the faces of the separators. More challenging will be the extention of the smoothing procedure to 3D, as the optimization objectives for a separator need to take into account multiple faces. Moreover, the theoretical angle bound for the 3D version of Ruppert's algorithm is $90°$. It is unrealistic to expect a decomposer to achieve this bound, at least for discrete approaches. In real life simulations the input domain is also unlikely to comform with this bound. Therefore, the parallel procedure has to be constructed targeting a 3D meshing procedure that can address the problem of the small input angles.

The second problem is to develop a premeshing procedure for the separating surfaces that will guarantee the conformity and quality of the mesh. While computing the the required sizes does not appear to be problematic, the premeshing procedure is likeley to create entities with distance less than the minimum local feature size. This will result a dead loop, since a new iteration will be needed to capture the new minimum local feature size. The resolution of this problem lies on establishing a termination criterion in terms of face areas rather than the length of the edges.

3D Delaunay mesh generation procedures that can cope with small input angles have been studied in the last few years, and are still a subject of study (a short description of the related bibliography is given in the beginning of Chapter 8). Some of these procedures, like the one described by Cheng and Poon [15], premesh the boundary, and protect the boundary edges and faces. Such approaches naturally extend to parallel decoupling procedures. I consider this parallelization approach most attractive, as it relaxes the angle requirements of the separators. At the same time, theoretical guarantees of conformity and quality can be derived. The parallel version of such algorithms may allow improvements, like independently calculating the local feature size for each subdomain. The known sequential procedures though are fairly complicated, and more simple methods may be developed in the future.

The problems of domain decomposition and mesh generation are increasingly revealed to be coupled with the related models. Anisotropic, and in general adaptive, methods are subject of current and future research. In the context of parallel computing these approaches have to be applied locally, and the availability of decoupling methods for these adaptive approaches would allow efficient parallel adaptive mesh generation procedures to be developed.

Bibliography

[1] Cecil G. Armstrong, Desmond J. Robinson, Mike McKeag, T. Li, S. Bridgett, R. Donaghy, and C. McGleenan, *Medials for meshing and more*, Proceedings of 4th International Meshing Roundtable, Sandia National Laboratories, 1995, pp. 277–288.

[2] Kevin Barker, Nikos Chrisochoides, Andrey Chernikov, and Keshav Pingali, *A load balancing framework for adaptive and asynchronous applications*, IEEE Trans. Parallel and Distributed Systems **15** (2004), no. 2, 183–192.

[3] Kevin Barker, Nikos Chrisochoides, Jeffrey Dobbelaere, Demian Nave, and Keshav Pingali, *Data movement and control substrate for parallel adaptive applications*, Concurrency and Computation Practice and Experience **14** (2002), 77–101.

[4] Stephen T. Barnard and Horst D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience **6** (1994), 101–107.

[5] Marsha Berger and Shahid Bokhari, *A partitioning strategy for pdes across multiprocessors.*, Proceedings of the 1985 International Conference on Parallel Processing, August 1985.

[6] Marshall Bern, David Eppstein, and John R. Gilbert, *Provably good mesh generation*, Proc. 31st IEEE Symp. Foundations of Computer Science, 1990, To appear in *J. Comp. System Science*, pp. 231–241.

[7] Marshall Bern and Paul Plassmann, *Mesh generation*, Handbook of Computational Geometry (Jörg Sack and Jorge Urrutia, eds.), Elsevier Science, 1999.

[8] Guy E. Blelloch, Garry L. Miller, Jonathan C. Hardwick, and Dafna Talmor, *Design and implementation of a practical parallel Delaunay algorithm*, Algorithmica **24** (1999), no. 3/4, 243–269.

[9] Harry Blum, *A transformation for extracting new descriptors of shape*, Models for the Perception of speech and Visual Form, MIT Press, 1967, pp. 362–380.

[10] Houman Borouchaki, Paul-Louis George, Frederic Hecht, Patrick Laug, and Eric Saltel, *Delaunay mesh generation governed by metric specifications, Part I. Algorithms and Part II. Applications*, Finite Elements in Analysis and Design **25** (1997), 61–83 and 85–109.

[11] Houman Borouchaki, Frediric Hecht, and Pascal J. Frey, *Mesh gradation control*, 6th International Meshing Roundtable, Sandia National Laboratories, Oct. 1997, pp. 131–141.

[12] Jonathan Brandt, *Convergence and continuity criteria for discrete approximations of the continuous planar skeleton*, CVGIP:Image Understanding **59** (1994), 116–124.

[13] Jonathan Brandt and Ralph Algazi, *Continuous skeleton computation by Voronoi diagram*, Comput. Vision, Graphics, Image Process. **55** (1992), 329–338.

[14] Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, and Tathagata Ray, *Quality meshing for polyhedra with small angles*, Proc. 20th Annu. Sympos. Computational Geometry, 2004, pp. 290–299.

[15] Siu-Wing Cheng and Sheung-Hung Poon, *Graded conforming Delaunay tetrahedralization with bounded radius-edge ratio*, 14th annual ACM-SIAM symposium on Discrete algorithms (Baltimore, Maryland), 2003, pp. 295 – 304.

[16] Andrey Chernikov and Nikos Chrisochoides, *Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement*, Proceedings of the 18th annual international conference on Supercomputing (Malo, France), 2004, pp. 48–57.

[17] L. Paul Chew, *Guaranteed quality triangular meshes*, Tech. Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[18] ______, *Guaranteed-quality mesh generation for curved surfaces*, 9th Annual Symposium on Computational Geometry (San Diego, California), ACM, 1993, pp. 274–280.

[19] L. Paul Chew, Nikos Chrisochoides, and Florian Sukup, *Parallel constrained Delaunay triangulation*, ASME/ASCE/SES Special Symposium on Trends in Unstructured Mesh Generation (Evanston, IL), 1997, pp. 89–96.

[20] Hyeong In Choi, Sung Woo Choi, and Hwan Pyo Moon, *Mathematical theory of medial axis transform*, Pacific Journal of Mathematics **181** (1997), 57–88.

[21] N. Chrisochoides, *An alternative to data mapping for parallel PDE solvers: parallel grid generation*, Scalable Parallel Libraries Conference (Mississippi State University, Mississippi), IEEE, 1993.

[22] Nikos Chrisochoides, *Multithreaded model for the dynamic load-balancing of parallel adaptive pde computations*, Applied Numerical Mathematics **20** (1996), 349–365.

[23] ______, *Parallel mesh generation.*, Numerical Solution of Partial Differential Equations on Parallel Computers (Are Magnus Bruaset, Petter Bjorstad, and Aslak Tveito, eds.), Springer-Verlag, 2005.

[24] Nikos Chrisochoides and Demian Nave, *Parallel Delaunay mesh generation kernel*, International Journal for Numerical Methods in Engineering **58** (2003), no. 2, 161–176.

[25] Carlos A. Coello Coello, *A Short Tutorial on Evolutionary Multiobjective Optimization*, First International Conference on Evolutionary Multi-Criterion Optimization (Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, eds.), Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001, pp. 21–40.

[26] David Cohen-Steiner, Éric Colin de Verdiére, and Mariette Yvinec, *Conforming Delaunay triangulations in 3d*, SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry (New York, NY, USA), ACM Press, 2002, pp. 199–208.

[27] Hugues L. de Cougny and Mark S. Shephard., *Parallel unstructured grid generation.*, CRC Handbook of Grid Generation (J. F. Thompson, B. K. Soni, and Nigel P. Weatherill, eds.), CRC Press, Inc., Boca Raton,, 199, pp. 24–1 – 24–18.

[28] Hugues L. de Cougny and Mark S. Shephard, *Surface meshing using vertex insertion*, Proceedings of the 5th International Meshing Roundtable, 1996, pp. 243–256.

[29] ———, *Parallel volume meshing using face removals and hierarchical repartitioning*, Computer Methods in Applied Mechanics and Engineering **174** (1999), no. 3–4, 275–298.

[30] Frank Deister, Udo Tremel, Oubay Hasan, and Nigel P. Weatherill, *Fully automatic and fast mesh size specification for unstructured mesh generation*, Engineering with Computers **20** (2004), 237–248.

[31] Boris N. Delaunay, *Sur la Sphére Vide*, Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheski i Estestvennyka Nauk **7** (1934), 793–800.

[32] Ralf Diekmann, Derk Meyer, and Burkhard Monien, *Parallel decomposition of unstructured FEM-meshes*, Concurrency: Practice and Experience **10** (1998), no. 1, 53–72.

[33] Suchuan Dong and George E. Karniadakis, *DNS of flow past a stationary and oscillating rigid cylinder at re = 10,000*, Journal of Fluids and Structures **20(4)** (2005), 519–531.

[34] Herbert Edelsbrunner, *Geometry and topology for meshing*, Cambridge University Press, 2001.

[35] Pascal Jean Frey and Paul-Louis George, *Mesh generation*, Hermes Science Publishing, 2000.

[36] Adam Gaither, Dave Marcum, Donna Reese, and Nigel Weatherill, *A paradigm for parallel unstructured grid generation*, 5th International Conference on Numerical Grid Generation in Computational Field Simmulations (Mississippi State University), April 1996, pp. 731–740.

[37] Jerôme Galtier and Paul-Louis George, *Prepartitioning as a way to mesh subdomains in parallel*, 5th International Meshing Roundtable (Pittsburgh, Pennsylvania), 1996, pp. 107–122.

[38] Paul-Louis George and Houman Borouchaki, *Delaunay triangulation and meshing: Applications to finite element*, Hermis, Paris, 1998.

[39] H. Nebi Gursoy and Nicholas M. Patrikalakis, *An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part I algorithms*, Engineering With Computers **8** (1992), 121–137.

[40] Bruce Hendrickson and Robert W. Leland, *The Chaco user's guide version 2.0*, Tech. Report SAND95-2344, Sandia National Laboratories, 1995.

[41] ———, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM Journal on Scientific Computing **16** (1995), no. 2, 452–469.

[42] ______, *A multi-level algorithm for partitioning graphs*, ACM/IEEE Conference on Supercomputing (San Diego, CA), 1995.

[43] Clemens Kadow and Noel Walkington, *Design of a projection-based parallel Delaunay mesh generation and refinement algorithm*, 4th Symposium on Trends in Unstructured Mesh Generation, 2003.

[44] George Karypis and Vipin Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Report TR 95-035, Department of Computer Science, University of Minnesota, Minneapolis, 1995.

[45] ______, *MeTis: Unstructured graph partitioning and sparse matrix ordering system, version 2.0*, 1995.

[46] Brian W. Kernighan and Shen Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Systems Technical Journal **49(2)** (1970), 291–307.

[47] Ravi Konuru, Jeremy Casas, Robert Prouty, Steve Oto, and Jonathan Walpore, *A user level process package for pvm*, Scalable High-Performance Computing Conferene, IEEE, 1997, pp. 48–55.

[48] B. G. Larwood, Nigel P. Weatherill, O. Hassan, and K. Morgan, *Domain decomposition approach for parallel unstructured mesh generation*, International Journal for Numerical Methods in Engineering **58** (2003), 177–188.

[49] C. K. Lee, *On curvuture element-size control in metric surface mesh generation*, International Journal for Numerical Methods in Engineering **50** (2001), 787–807.

[50] D. T. Lee, *Medial axis transformation of a planar shape*, IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI 4** (1982), no. 4, 363–369.

[51] Xiangrong Li, Jean-Francois Remacle, Nicolas Chevaugeon, and Mark S. Shephard, *Anisotropic mesh gradation control*, Proceedings, 13th International Meshing Roundtable (Williamsburg, VA), Sandia National Laboratories, 2004, pp. 401–412.

[52] Leonidas Linardakis and Nikos Chrisochoides, *Delaunay Decoupling Method for parallel guaranteed quality planar mesh refinement*, SIAM Journal on Scientific Computing **27** (2006), no. 4, 1394–1423.

[53] Rainald Löhner, *Extensions and improvements of the advancing front grid generation technique*, Communications in Numerical Methods in Engineering **12** (1996), 683–702.

[54] Rainald Löhner, *Automatic unstructued grid generators*, Finite Elements in Analysis and Design **25** (1997), 111–135.

[55] Rainald Löhner and Juan Raul Cebral, *Parallel advancing front grid generation*, International Meshing Roundtable, Sandia National Labs, 1999.

[56] meDDec, `http://www.cs.wm.edu/~leonl01/meddec/meddec.html`.

[57] Metis, `http://www-users.cs.umn.edu/~karypis/metis/index.html`.

[58] Garry L. Miller, Steven E. Pav, and Noel J. Walkington, *Fully incremental 3D Delaunay mesh generation*, 11th International Meshing Roundtable, Sandia National Laboratory, September 2002, p. 7586.

[59] ———, *When and why Ruppert's algorithm works*, 12th International Meshing Roundtable, Sandia National Laboratory, September 2003, pp. 91–102.

[60] Scott A. Mitchell, *Cardinality bounds for triangulations with bounded minimum angle*, Sixth Canadian Conference on Computational Geometry, 1994, pp. 326–331.

[61] B. Nour-Omid, A. Raefsky, and G. Lyzenga, *Solving finite element equations on concurrent computers*, American Soc. Mech. Eng (1986), 291–307.

[62] Steven J. Owen and Sunil Saigal, *Neighborhood-based element sizing control for finite element surface meshing*, 6th International Meshing Roundtable (Park City, UT), 1997.

[63] ———, *Surface mesh sizing control*, International Journal for Numerical Methods in Engineering **47** (2000), 497–511.

[64] Steven E. Pav, *Delaunay refinement algorithms*, Ph.D. thesis, Carnegie Mellon University, May 2003.

[65] Steven E. Pav and Noel J. Walkington, *Robust three dimensional Delaunay refinement*, 13th International Meshing Roundtable (Williamsburg, VA), September. 2004.

[66] Philippe P. Pébay and J. Frey Pascal, *A-priori Delaunay-conformity*, 7th International Meshing Roundtable, 1998, pp. 321–333.

[67] Per-Olof Persson, *PDE-based gradient limiting for mesh size functions*, Proc. of the 13th Int. Meshing Roundtable, August 2004, pp. 377–387.

[68] ———, *Mesh size functions for implicit geometries and PDE-based gradient limiting*, Engineering with Computers **22** (2006), no. 2, 95–109.

[69] Mark A. Price, Clive Stops, and Geoffrey Butlin, *A medial object toolkit for meshing and other applications*, Proceedings of 4th International Meshing Roundtable, 1995, pp. 219–229.

[70] William Roshan Quadros, Steven James Owen, Mike Brewer, and Kenji Shimada, *Finite element mesh sizing for surfaces using skeleton*, 13th International Meshing Roundtable (Williamsburg, VA,), Sandia National Laboratories, 2004, pp. 389–400.

[71] Jim Ruppert, *A new and simple algorithm for quality 2-dimensional mesh generation*, 4th ACM-SIAM Symp. on Discrete Algorithms, 1993, pp. 83–92.

[72] ———, *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, J. Algorithms **18** (1995), no. 3, 548–585.

[73] R. Said, Nigel Weatherill, K. Morgan, and N. Verhoeven, *Distributed parallel Delaunay mesh generation*, Comp. Methods Appl. Mech. Engrg. **177** (1999), 109–125.

[74] SciClone, `http://www.compsci.wm.edu/SciClone/index.html`.

[75] Mark Shephard, J. Flaherty, Hugues de Cougny, C. Ozturan, C. Bottasso, and M. Beall, *Parallel automated adaptive procedures for unstructured meshes*, Parallel Computing in CFD, (1995), no. R-807, 6.1–6.49.

[76] Evan C. Sherbrooke, Nicholas M. Patrikalakis, and Franz-Erich Wolter, *Differential and topological properties of medial axis transforms*, Graphical Models and Image Processing **55** (1996), no. 1, 574–592.

[77] Jonathan Richard Shewchuk, *Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator*, Applied Computational Geometry: Towards Geometric Engineering (Ming C. Lin and Dinesh Manocha, eds.), Lecture Notes in Computer Science, vol. 1148, Springer-Verlag, May 1996, From the First ACM Workshop on Applied Computational Geometry, pp. 203–222.

[78] ———, *Delaunay refinement mesh generation*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997, Available as Technical Report CMU-CS-97-137.

[79] ———, *Tetrahedral mesh generation by Delaunay refinement generation*, Fourteenth Annual Symposium on Computational Geometry, ACM, June 1998, pp. 86–95.

[80] ———, *Mesh generation for domains with small angles*, Proceedings of the sixteenth annual symposium on Computational geometry, 2000, pp. 1–10.

[81] ———, *Delaunay refinement algorithms for triangular mesh generation*, Computational Geometry: Theory and Applications **22(1-3)** (2002), 21–74.

[82] Robin Sibson, *Locally equiangular triangulations*, The Computer Journal **21** (1978), no. 3, 243–245.

[83] Barry Smith, Petter Bjørstad, and William Gropp, *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, New York, 1996.

[84] T. K. H. Tam and Cecil G. Armstrong, *2D finite element mesh generation by medial axis subdivision*, Advances in Engineering Software and Workstations **13** (1991), no. 5-6, 313–324.

[85] T. K. H. Tam, Cecil G. Armstrong, and Mike McKeag, *Computing the critical points on the medial axis of a planar object using a Delaunay point triangulation algorithm*, Tech. report, Mesh generation Group, Dept. Mech. Eng., the Queen's University of Belfast., 1991.

[86] Triangle, `http://www.cs.cmu.edu/~quake/triangle.html`.

[87] Chris Walshaw and Mark Cross, *Mesh partitioning: A multilevel balancing and refinement algorithm*, SIAM Journal on Scientific Computing **22** (2000), no. 1, 63–80.

[88] Chris Walshaw, Mark Cross, and M. G. Everett, *Parallel dynamic graph partitioning for adaptive unstructured meshes*, Journal of Parallel and Distributed Computing **47** (1997), no. 2, 102–108.

[89] Franz-Erich Wolter, *Cut locus and medial axis in global shape interrogation and represenation*, Tech. report, MIT, Department of Ocean Engeneering, Design Laboratory, 1993.

[90] Jin Zhu, Ted Blacker, and Rich Smith, *Background overlay grid size functions*, 11th International Meshing Roundtable, Sandia National Laboratories, September 2002, pp. 65–74.

VITA

Leonidas Linardakis

Leonidas Linardakis received a B.S. and M.S. degree in Mathematics, in 1997 and 1999 respectively, from the University of Ioannina, Greece. In 2003 he earned an M.S. degree in Computer Science from the College of William and Mary.