A Static Geometric Medial Axis Domain Decomposition in 2D Euclidean Space

LEONIDAS LINARDAKIS and NIKOS CHRISOCHOIDES
College of William and Mary

We present a geometric domain decomposition method and its implementation, which produces good domain decompositions in terms of three basic criteria: (1) The boundary of the subdomains create good angles, i.e., angles no smaller than a given tolerance Φ_0, where the value of Φ_0 is determined by the application which will use the domain decomposition. (2) The size of the separator should be relatively small compared to the area of the subdomains. (3) The maximum area of the subdomains should be close to the average subdomain area. The domain decomposition method uses an approximation of a Medial Axis as an auxiliary structure for constructing the boundary of the subdomains (separators). The N-way decomposition is based on the "divide and conquer" algorithmic paradigm and on a smoothing procedure that eliminates the creation of any new artifacts in the subdomains. This approach produces well shaped uniform and graded domain decompositions, which are suitable for parallel mesh generation.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; J.6 [**Computer-Aided Engineering**]:

General Terms: Algorithms

Additional Key Words and Phrases: domain decomposition, parallel mesh generation, Delaunay triangulation

1. INTRODUCTION

Although the Domain Decomposition (DD) problem has been studied for more than 20 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. [Smith et al. 1996]. Here we examine the Geometric Domain Decomposition problem (GDD) in the context of parallel mesh generation. We focus on the formulation, solution and implementation of the GDD problem for a continuous 2-dimensional (2D) domain Ω into non-overlapping subdomains Ω_i, so that the subdomains Ω_i create no new artifacts, such as small angles between the separators $\partial\Omega_i$, and the separators and

This work was partially sponsored by the National Science Foundation under Grant No. 0049086, 0085969. Funding support was also provided through the Virginia Institute of Marine Science and the Southeastern Universities Research Association under the grants ONR N00014-05-1-0831 and NOAA NA04NOS4730254. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the aforementioned institutes.

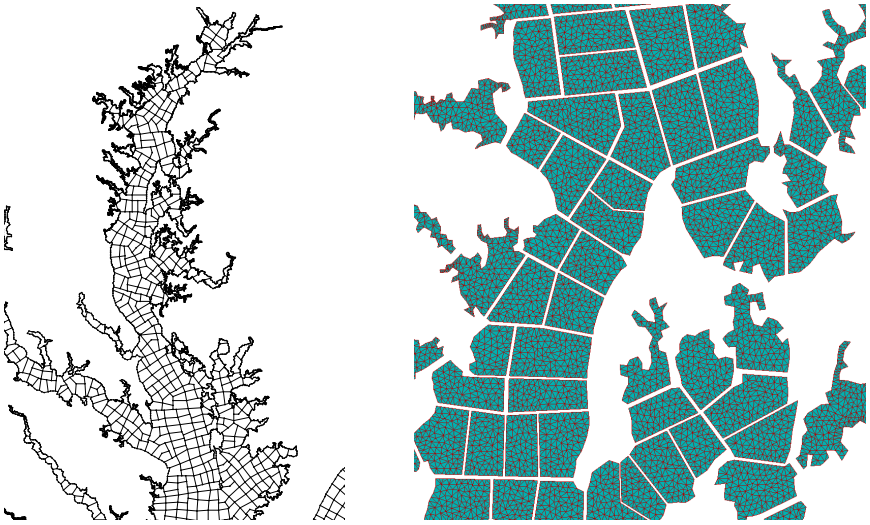

Fig. 1. **Left:** Part of the Chesapeake bay geometry decomposed uniformly by the MADD method. The angles created by the separators are greater than 70°. **Right:** Detail of the Delaunay mesh of the decomposed geometry. The decompositions produced by the MADD are suitable for stable parallel graded mesh generation.

the external boundary $\partial\Omega$. These decompositions are suitable for stable parallel graded mesh generation procedures (see Fig. 1), 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.

Parallel mesh generation methods decompose the original meshing problem into smaller subproblems that can be solved (i.e., meshed) in parallel. There are two approaches that can be employed in order to decompose the problem: *mesh data decomposition* and *geometric domain decomposition* techniques. Mesh data decomposition techniques compute data-subsets of the mesh that can be processed in parallel [Chrisochoides and Nave 2003; Kadow and Walkington 2003; Chernikov and Chrisochoides 2004]. The decomposition for these approaches is an easier problem than the geometric domain decomposition problem, but communication and local synchronization are unavoidable during the parallel mesh generation in order to maintain the conformity and quality of the distributed mesh.

Geometric domain decomposition techniques partition the domain 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 [Chew et al. 1997], or no communication at all [Galtier and George 1996; Said et al. 1999; Linardakis and Chrisochoides 2006], and thus are very efficient. When communication is required by the parallel mesh generation procedure, this will be analogous to the lengths of the separators. Hence, one of the

goals of the domain decomposition is to produce small separators. On the other hand, the separators will be part of the geometry, and consequently part of the final mesh, so the decomposition has to meet certain quality criteria (like the size of the angles that it imposes), so that the quality of the mesh will not be distorted.

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 [Karypis and Kumar 1995b]. On the other hand, geometry–based techniques take into account the geometric characteristics of the domain. For example, the Recursive Coordinate Bisection approach [Berger and Bokhari 1985] recursively bisects the domain along the axes, while the Inertial method [Nour-Omid et al. 1986] uses the inertia axis of the domain to produce a decomposition. Finally, libraries like Chaco [Hendrickson and Leland 1995a] provide both topology and geometry-based approaches.

2. THE GEOMETRIC DOMAIN DECOMPOSITION PROBLEM

We examine the GDD problem in the context of parallel mesh generation. In the rest of this paper we define as a domain Ω 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. Formally a 2-way domain decomposition is defined as follows. We assume the domain Ω is the closure of an open connected bounded set and the boundary $\partial\Omega$ is a PSLG that formed a set of linear segments which do not intersect[1]. 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 Ω_1, Ω_2 of Ω, such that: Ω_1 and Ω_2 are connected sets, with $\Omega_1 \cup \Omega_2 = \Omega$, and for every path $P \subset \Omega$, which connects a point of Ω_1 to a point of Ω_2, we have $P \cap \mathcal{H} \neq \emptyset$.

Guaranteed quality mesh generation algorithms [Chew 1989; 1993; Ruppert 1995] produce elements with good aspect ratio and good angles. These algorithms require that the initial boundary angles are within certain good bounds. For example, Ruppert's algorithm [Ruppert 1995] requires boundary angles (the angles formed by the boundary edges) no less than 60°, in order to guarantee the termination. When these algorithms are used in parallel, domain decomposition based, mesh generation procedures, the separators are treated as external boundary of each subdomain. So, the domain decomposition should create separators that meet the requirements of the mesh generation algorithm. Therefore the constructed separator 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. Even in cases like [Triangle], where a mesh generator can handle small input angles, these angles are artifacts and will be permanent, distorting the quality of the final mesh produced by the parallel mesh generator.

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,

[1]This definition does not allow internal boundaries. The algorithm we present can be extended to handle internal boundaries, if needed.

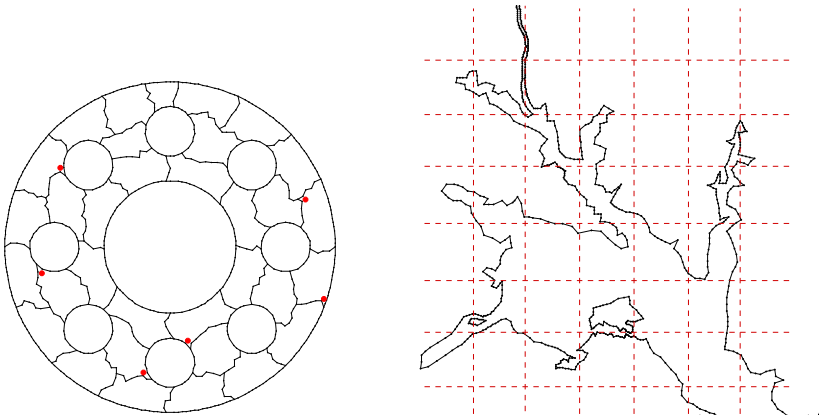

Fig. 2. **Left:** The Pipe geometry. The angles produced by graph-based partitioners, like Metis, depend on the background mesh, and can be as small as the smaller angle of the mesh. The angles marked with dots are "small" (less than 60°). **Right:** Part of the Chesapeake bay geometry. When the geometry 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.

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 [Chrisochoides 1996]. Over-decomposition allows both static and dynamic load balancing methods to distribute equally the work-load among the processors more effectively [Barker et al. 2004; Linardakis and Chrisochoides 2006]. 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 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

[2]Small work loads do not create load-balancing problems, on the contrary, the resulting granularity can be used to improve the load balance, especially on heterogenous 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 Delaunay mesh generators, the work-load is analogous to the area of the subdomain [Linardakis and Chrisochoides 2006]

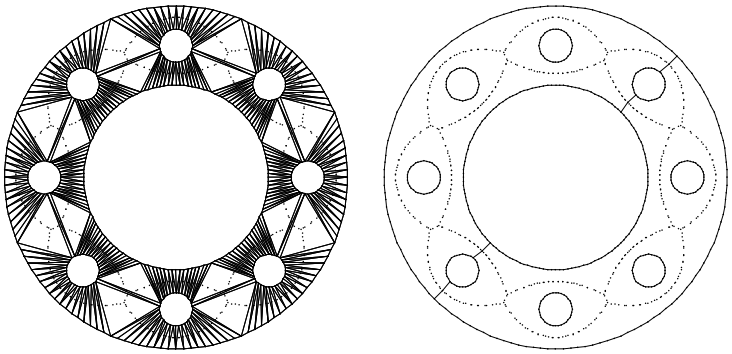

Fig. 3. **Left:** The Delaunay triangulation of the Pipe domain. The circumcenters of the triangles approximate the medial axis. **Right:** The circumcenters are the Voronoi nodes. The separator is formed by selecting a subset of the Voronoi nodes and connecting them with the boundary.

not address the problem of the formed angles. For example, graph based partitioning algorithms, 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 Left, and 19). 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 Right), so they are unsuitable for parallel mesh generation procedures. The geometric domain decomposition approach we present addresses all of the three above criteria, and is suitable for stable and efficient parallel mesh generation procedures.

3. MEDIAL AXIS DOMAIN DECOMPOSITION METHOD

The Medial Axis Domain Decomposition (MADD) method was first introduced in [Linardakis and Chrisochoides 2006] in the context of the Delaunay Decoupling method and it is based on an approximation of the medial axis (MA) of the domain. The MA was introduced in [Blum 1967], and has been studied and utilized extensively (cf. [Attali et al.]. The approximation of the MA in the MADD method is used as an auxiliary structure to determine separators that form good angles. In this paper we present an expanded and improved version of the MADD method which includes a graded N-way domain decomposition procedure, and a smoothing procedure for improving the quality of the separators.

A circle $C \subseteq \Omega$ is said to be maximal in a domain 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 and will be denoted by $\mathrm{MA}(\Omega)$. The intersection of the boundary of Ω and a maximal circle C is not empty. The points $C \cap \partial\Omega$, where a maximal circle C intersects the boundary, are called *contact points* of c, where c is the center of C. If b is a contact point of c, then the angles formed by the segment cb and the boundary are at least $\pi/2$ [Linardakis and Chrisochoides 2006].

The medial axis of the domain can be approximated by Voronoi nodes of a dis-

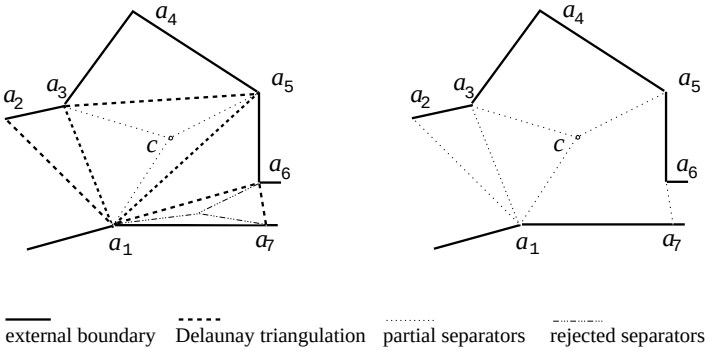

Fig. 4. **Left** is a part of the Delaunay triangulation and **right** are the partial separators. Triangle $a_1a_3a_5$ is a junction triangle, while the other triangles are not.

cretization of the domain [Brandt and Algazi 1992]. The Voronoi nodes are the circumcenters of the Delaunay triangulation of the discretized domain (see Fig. 3 right). As the Voronoi nodes approximate the MA, the segments that connect them to the boundary tend to create angles close to $\pi/2$. The approximation of $MA(\Omega)$ 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 nodes of the boundary points. The separators will be formed by either connecting these circumcenters to two of the vertices of the Delaunay triangles, giving two segments, or from edges of the Delaunay triangles, giving one segment. In both cases these segments are chosen so that they form good angles, with each other and the external boundary, and they are called *partial separators*. A complete separator will be formed by a set of one or more partial separators. Fig. 3 depicts the boundary conforming mesh of the cross section of a rocket and the medial axis approximation (left), and a 2-way separator for the same geometry (right).

Our goal is to create decompositions that form angles no less than a tolerance Φ_0. The partial separators we choose are of two types (see Fig. 4): (a) non-boundary edges of the Delaunay triangulation that form angles $\geq \Phi_0$ with the boundary, and (b) segments that connect a triangle circumcenter with the triangle vertices. The first type of partial separator is easy to identify. 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. In order to identify the second type of partial separator we define a special type of triangles. Let $\mathcal{D}$ be a Delaunay triangulation of a discretization D of the boundary $\partial\Omega$. We call a triangle $t \in \mathcal{D}$ a *junction triangle* if:

(1) it includes its circumcenter c,

(2) at least two of its edges are not in D,

(3) at least two of the segments defined by the circumcenter and the vertices of t form angles $\geq \Phi_0$, both with the boundary and each other.

The second type of partial separators are included in junction triangles. In Fig. 4,

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, where $\Phi_0 = 60°$), so it violates property (3). 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 $\mathcal{D}$ is a boundary conforming triangulation. The complete separator is formed by choosing a subset of partial separators that will guarantee the decomposition of the geometry into two connected subdomains.

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. It is a difficult problem to pre-determine the level of the refinement that would give an optimal decomposition. Increasing the boundary refinement results 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 depends on the number of input points. Furthermore, it will take more time to identify the partial separators and form a complete separator. Second, it could result into arithmetic rounding errors when calculating geometric entities, like circumcenters and angles.

There are three parameters that effect the level of required refinement of the boundary: (1) the number of subdomains we want to create, (2) the characteristics of the initial geometry, and (3) the angle lower bound Φ_0. In our implementation we compute a refining size based on the average length of the initial boundary edges and on the square root of the number of subdomains (see Section 6). The angle lower bound Φ_0 can be as large as 80°, depending on the geometry and the refining factor; for values larger than this the algorithm may not find a separator that satisfies the angle lower bound condition.

4. THE MADD ALGORITHM

The MADD algorithm uses as a starting point the approximation of the medial axis by the Delaunay triangulation $\mathcal{D}$, 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 [Shewchuk 1996], 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 $\mathcal{D}$ into a graph $G_{\mathcal{D}}$,

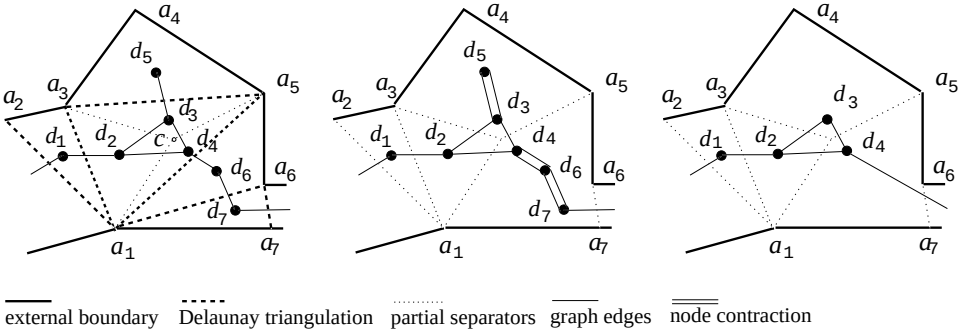

Fig. 5. An example of creating the MADD graph. **Left** is a part of the Delaunay triangulation and the creation of the corresponding initial graph $G_{\mathcal{D}}$. **Center**, the procedure of contracting the graph by combining the vertices of $G_{\mathcal{D}}$. The vertices connected by doubled lines are combined. **Right** is the final graph $G'_{\mathcal{D}}$ that corresponds to this part.

which is a modified element dual graph. The information encapsulated in this graph includes: (a) the topology of $\mathcal{D}$, (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 $G_{\mathcal{D}}$ is constructed, the graph is contracted, so that only the partial separators of $\mathcal{D}$ are represented as graph edges (see Section 4.2). Then the contracted graph is partitioned in a way that minimizes the cut cost and gives balanced subdomain weights. In our implementation we have used Metis [Karypis and Kumar 1995b], which is considered to be a state of art graph partitioner. Finally the graph partition is translated back into insertions of partial separators, which results a 2-way decomposition (see Section 4.3). The major steps of the algorithm are:

(1) Create a modified element dual graph $G_{\mathcal{D}}$ from the Delaunay triangulation $\mathcal{D}$.
(2) Contract $G_{\mathcal{D}}$ into the graph $G'_{\mathcal{D}}$, so that only the candidate partial separators are represented as edges of $G'_{\mathcal{D}}$.
(3) Partition the graph $G'_{\mathcal{D}}$, optimizing the cut-cost to subgraph weight ratio.
(4) Translate the cuts of the previous partition into the corresponding partial separators and insert them into the geometry.

4.1 Construction of the Graph $G_{\mathcal{D}}$

In this step the junction triangles of the Delaunay triangulation $\mathcal{D}$ are divided into three triangles, and the final triangulation is represented as a weighted dual graph. Each of the three triangles included into a junction triangle are represented by three graph vertices. Non-junction triangles are represented by a single graph vertex. Vertices that represent adjacent triangles are connected by a graph edge. The weight of each vertex is set equal to the area of the corresponding triangle, while the weight of a graph edge connecting two vertices is set equal to the length of the common triangle edge that is shared by the two corresponding triangles.

Fig. 5 (left) depicts the step for constructing the graph $G_{\mathcal{D}}$. 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 vertex, d_1, d_6, d_5, and d_7 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 vertices d_2, d_4, and d_3 respectively. The weight of each vertex is equal to the area of the corresponding triangle. For example, the vertex d_2 has weight equal to the area $|a_1ca_3|$. Vertices that represent adjacent triangles are connected by a graph edge, with weight equal to the length of their common triangle edge. For example, the vertices d_1 and d_2 are connected by a graph edge with weight equal to the length $|a_1a_3|$, while the vertices 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 1.

Algorithm 1.

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

4.2 Graph Contraction

In this step the graph $G_{\mathcal{D}}$ produced from the previous step is contracted into a new graph $G'_{\mathcal{D}}$, so that only the acceptable partial separators are represented as edges in $G'_{\mathcal{D}}$. In order to contract the graph $G_{\mathcal{D}}$ 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 vertices that were connected by the eliminated edge are combined into one vertex; the new vertex represents the total area of the triangles represented by the contracted vertices.

Fig. 5 (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 vertices d_3 and d_5 are combined into a new vertex. The new vertex 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 vertex also inherits all the external graph edges of the two previous vertices, which

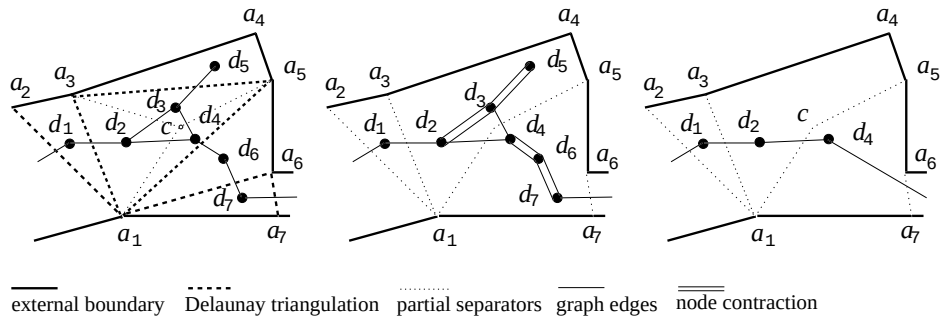

Fig. 6. An example of contraction of the vertices inside of a junction triangle. **Left** is a part of the Delaunay triangulation and the creation of the corresponding initial graph $G_{\mathcal{D}}$. **Center**, the procedure of contracting the graph, in this case the two vertices of the junction triangle $a_1a_3a_5$ are combined. **Right** is the final graph $G'_{\mathcal{D}}$ and the corresponding candidate partial separators.

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. 5 right the final graph $G'_{\mathcal{D}}$ is depicted with the corresponding areas and partial separators.

In Fig. 6 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 vertices d_2 and d_3 in the junction triangle $a_1a_3a_5$, which are separated by this edge, are combined into a new vertex. The new vertex inherits two graph edges connecting it to the same vertex 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 2.

Algorithm 2.

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

4.3 The Construction of the Separator

The result of the previous step is a graph $G'_{\mathcal{D}}$, 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.

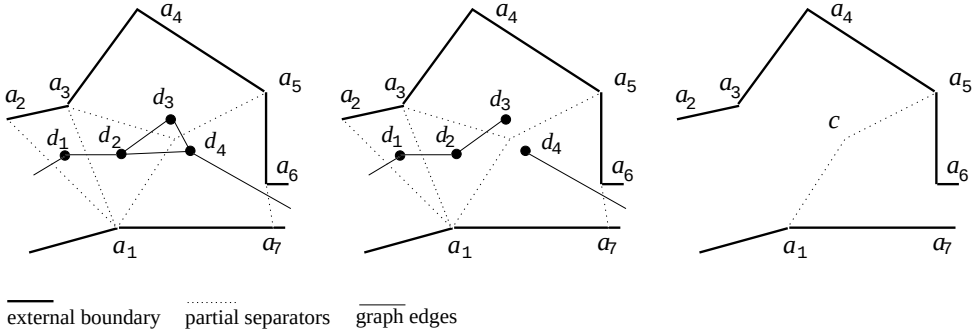

Fig. 7. **Left** is depicted the graph $G'_{\mathcal{D}}$ with the corresponding areas. **Center** The graph is partitioned by deleting two graph edges. **Right** The corresponding partial separator is inserted to the geometry.

The weights of the vertices of $G'_{\mathcal{D}}$ represent the size of the corresponding areas, while the 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 partitioning problem is challenging and has been the source of good algorithms and software [Kernighan and Lin 1970; Barnard and Simon 1994; Hendrickson and Leland 1995b; 1995c; Karypis and Kumar 1995a; Walshaw et al. 1997], The graph contraction step, described in the previous section, has the additional merit of reducing significantly the size of the graph, resulting a smaller partitioning problem. In our implementation we have used Metis library of graph partitioning algorithms [Karypis and Kumar 1995b].

After partitioning the graph $G'_{\mathcal{D}}$ 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. 7 (left) the graph $G'_{\mathcal{D}}$ 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.

Algorithm 3.

1. **for** all the edges $d_id_j \in G'_{\mathcal{D}}$ **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 $G'_{\mathcal{D}}$ has at least two vertices, then a 2-way partition exists and it will give a decomposition of the domain into two subdomains (for the proof see [Linardakis and Chrisochoides 2006]). Provided that the graph partitioner gives a small cut cost and balanced subgraph weights, the length of the separator will

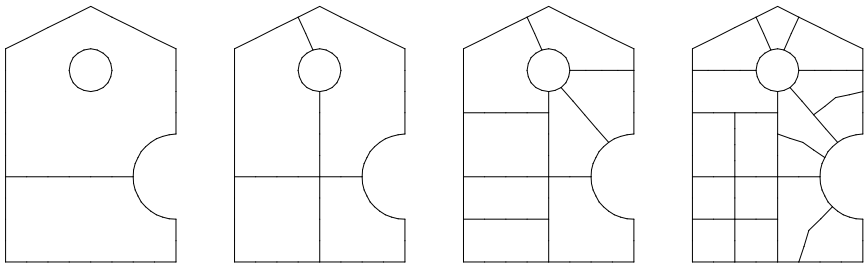

Fig. 8. N-way partitions, where $N = 2, 4, 8, 16$, by the MADD divide and conquer method.

be relatively small and the areas of the subdomains will be approximately equal. Moreover, since all the partial separators, by the construction of $G'_{\mathcal{D}}$, form good angles, the constructed separator will also form good angles. There are cases though 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 routine 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 occured.

In summary, the constructed separator meets the decomposition criteria C1 - C3 described in Section 2.

5. N-WAY DECOMPOSITION

The procedure described in the previous section decomposes the domain into two subdomains, our goal though is to obtain much larger number of subdomains. N-way decompositions can be obtained by applying the MADD procedure recursively, in a divide and conquer way (see Fig. 8). In the first step the domain is decomposed into two subdomains. Next, the largest subdomain is chosen and is decomposed again into two subdomains. The procedure is repeated until we have created the required number of subdomains. The N-way decomposition is described by Algorithm 4.

Algorithm 4.

1. Read the definition of the domain Ω;
2. Initialize and maintain a list of the subdomains;
3. **while** the current number of subdomains is less than N **do**
4. decompose the largest subdomain in two subdomains using the MADD algorithm;
5. update the list of the subdomains;
6. **endwhile**

The recursive approach has both advantages and disadvantages. The MADD algorithm is applied from scratch for every subdomain that is decomposed, so the

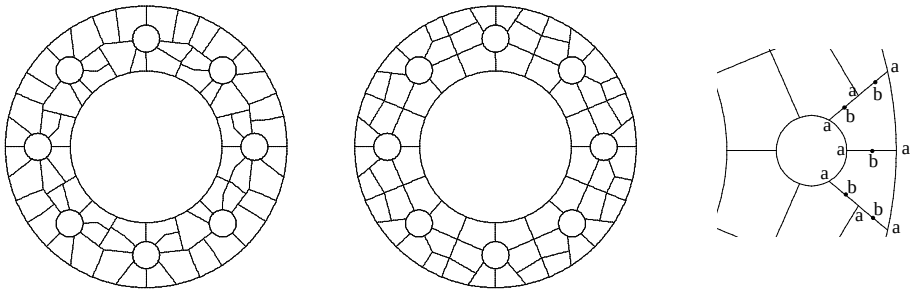

Fig. 9. The Pipe domain decomposed in 64 subdomains using the MADD algorithm. On the **left** no smoothing is used. Most of the separators don't meet at their end-points, and they create small segments on their common boundaries. On the **middle** the smoothing procedure is used, giving conforming separators. **Right**, the points of the first type (a) and second type (b) are depicted.

whole procedure, from the creation of the Delaunay triangulation to the graph partition and the insertion of separators, is repeated, discarding any previous information. Another disadvantage rises from the fact that each subdomain is decomposed independently from the neighboring subdomain. This might cause the insertion of separators that create unnecessary small segments on their common boundary of two neighboring subdomains (see Fig. 9, left). In order to solve this problem we introduce a smoothing procedure, which is described in Section 5.1.

The major advantage of the divide and conquer approach is that it adapts to the current geometry. The Delaunay triangulation of the subdomains are re-calculated, after we insert the separators, incorporating the information of the so far decomposition. The new separators are formed taking into account the current shape of the geometry. Fig. 8 demonstrates this fact. In contrast, if we used an N-way graph partitioning approach, this information would not be avaliable, resulting poor decompositions.

5.1 Smoothing the Separators

One of the the disadvantages of independently decomposing each subdomain is the possible creation of small features. The independent computation of the separators might create small segments along their common boundary (see Fig. 9, left). The size of these segments depends on the level of the boundary refinement. As we increase the number of segments, 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 (that is near $\pi/2$). In order to deal with these two issues we introduce a smoothing procedure that improves the quality of the decomposition.

The smoothing procedure takes place in two steps. The first step takes place during the construction of the graph $G_{\mathcal{D}}$. 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. 9, right). Points of the first type are either initial points part of 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 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 computation of this function is similar as in the previous smoothing step.

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.

5.2 N-way Graded Decomposition

The procedure that we have described so far for N-way decompositions 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. Certain parts of the domain, where the model indicates higher activity, require smaller size of mesh elements and thus a denser mesh. These parts could be determined in advance, based on the properties of the geometry and the model, or as a result of an error estimation

Fig. 10. 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.

function from a previous FEM procedure. The local mesh refinement procedure will result disproportional mesh sizes for the subdomains that include these critical areas. In order to maintain balanced memory requirements, and consequently workload, during the mesh generation procedure, we have to follow a graded approach in creating the domain decomposition. The areas of the created subdomains should be proportional to the expected mesh size, and the subdomains that require higher refinement should be decomposed into smaller subdomains.

The problem of determining the element size, and thus the gradation of a mesh, has been studied extensively in the mesh generation and refinement literature (cf. [Borouchaki et al. 1997; Löhner 1997; Owen and Saigal 1997; Deister et al. 2004; Zhu et al. 2002]). Usually the size of the elements is computed as a function of: (a) the geometry of the domain (curvature), (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. In most cases a background mesh and an interpolation procedure is employed to define the desired element size in each position of the domain.

In this section we describe a procedure that produces graded domain decompositions. using the MADD method. There are two ways to define the gradation of the subdomains. The first is to define the required area for each subdomain. The second is to assign a relative density weight for each subdomain, and use it as a gradation criterion. While the first approach is a natural extension of the existing approaches for defining the gradation of the mesh, it does not allow the user to predefine the number of subdomains she 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.

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 line 4 of Algorithm 4. In this step the subdomain with the greater area to required area ratio, or with the greater density weight, is chosen to be decomposed, instead of the subdomain with the larger area. In the case of using the area ratio, 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 parts of the geometry that have been assigned greater density weights will be decomposed more intensively, and the number of the created subdomains is predefined.

In our implementation the gradation of the decomposition 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. 10.

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 12, 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. 11, which is a model used to study the incompressible turbulent flow past a circular cylinder [Dong and Karniadakis 2005], and in Fig. 12. The 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

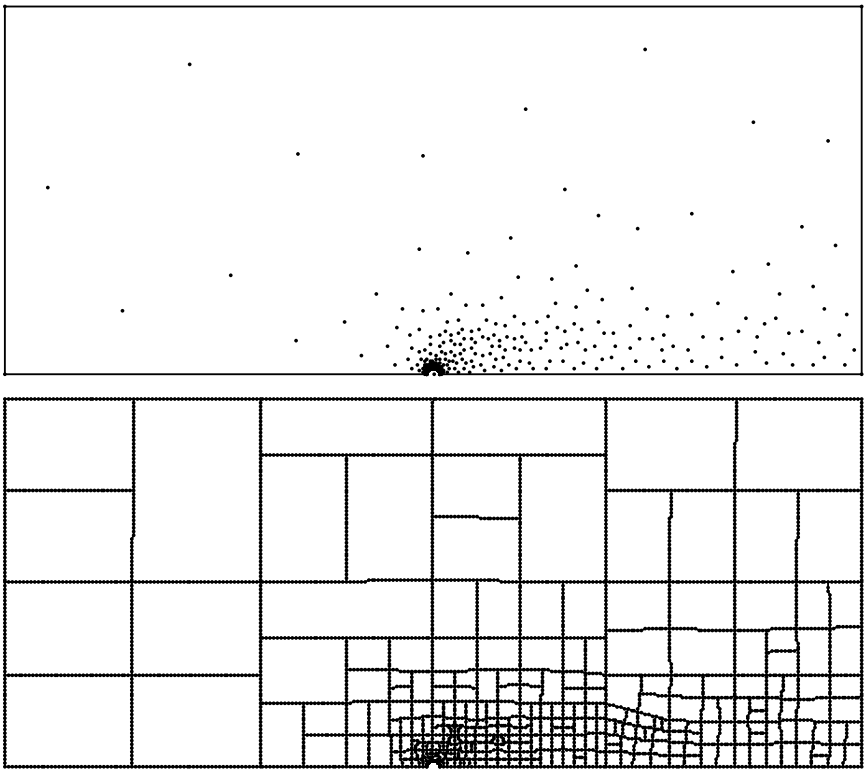

Fig. 11. Graded MADD based on weighted background mesh. **Top** is the weight background mesh vertices of the Cylinder domain, and **bottom** is the corresponding decomposition in 280 subdomains.

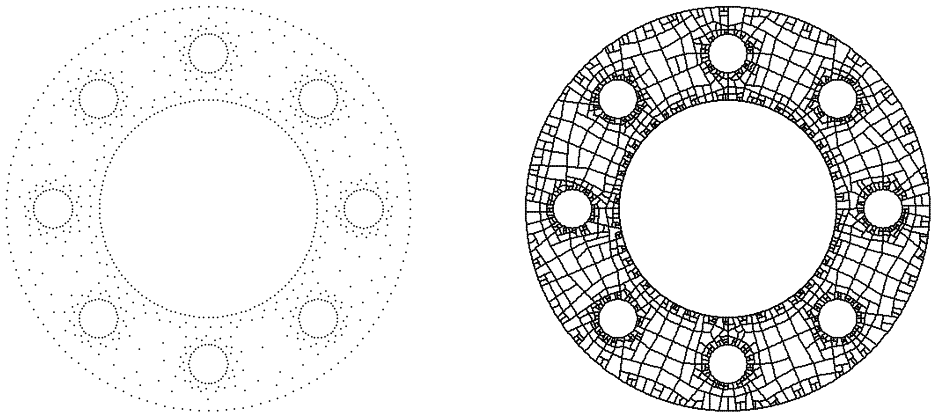

Fig. 12. Graded MADD based on weighted background mesh. **Left** is the weight background mesh vertices of the Pipe domain, and **right** is the corresponding decomposition into 1250 subdomains.

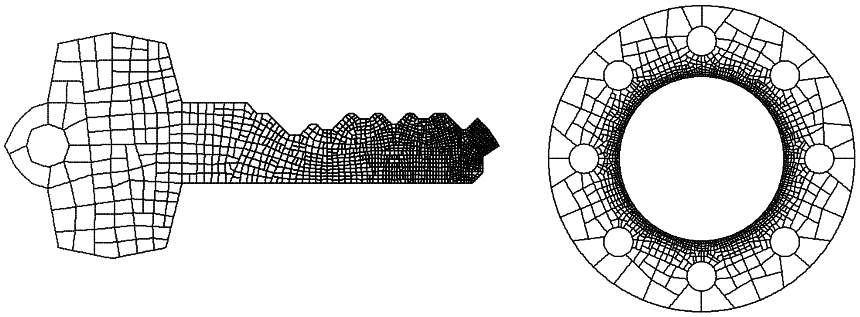

Fig. 13. **Left**, the Key is decomposed in 1250 subdomains using a linear weight function, proportional to x coordinate. **Right**, the Pipe decomposed in 1315 subdomains using an area function proportional to ρ^{12}, where ρ is the distance from the center of the inner circle.

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. 13 (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. 13 (right).

6. IMPLEMENTATION

The programming language for our implementation is ANSI C++, and the whole library is encapsulated into the `madd` class. The [Triangle] library ([Shewchuk 1996]) was used for the creation of the Delaunay triangulation during the MADD procedure. Also, the [Metis] library ([Karypis and Kumar 1995b]) was used for the graph partitioning step in the MADD procedure. The MADD method requires a graph partition into two connected subgraphs; we implemented a routine that restores the connectivity in the cases where Metis returns non-connected components. All the libraries where used without modifications.

The basic methods are described in Table I, a detailed description is provided in the source program. The libraries for the area and weight function are loaded dynamically through the `ld` dynamic linker. The file format (.poly) of the domains is the same as in [Triangle], and describes the domain in three sections: (a) a set of boundary points in x y coordinates, (b) a set of segments, in $\langle endpoint1\rangle$ $\langle endpoint2\rangle$ form, and (c) a set of holes in x y coordinates. The file format of the decompositions is similar. It describes the decomposition in four sections: (1) a set of points, (2) a set of segments, (3) a set of subdomains defined by segments of section (2), and (4) a set of holes for each subdomain. Examples of files are provided with the software.

```
readPolyFile(char* name)           Reads a poly file (the initial domain).
decompose(int subDomains)          The main decomposition routine.
readBackgroundWeights(char* name)  Reads weighted background mesh nodes from a file.
readBackgroundAreas(char* name)    Reads area background mesh nodes from a file.
writePolyFileAll(char* fileName)   Writes all the subdomains to different poly files.
writeSubdomainFile(char* name)     Writes the decomposition in points, segments, and
                                   subdomains - segments.
setWeightFunction(char *libraryName,     Sets the weight function.
                  char *functionName)
setAreaFunction(char *libraryName,       Sets the area function.
                char *functionName)
setPhi(double phiValue)  Sets minimum acceptable angle formed during the decomposition.
setDecAreaRatio(double ratioValue) Sets the min area ratio for decomposing a subdomain.
setUniformRefineLevel(double uniformValue)  Sets the uniform refining and
                                            decomposition factor.
setAdaptiveRefineLevel(double adaptiveValue) Sets the graded refining and
                                             decomposition factor.
setMaxImbalanceLevel(double imbalanceValue)  Sets the maximum acceptable imbalance
                                             during the madd partition.
setMaxSmoothLevel(int smoothLevel)  Defines the number of the smoothing iterations.
setInterpolationWeightLevel(double weightValue) Sets the interpolation coefficient
                                                for the  weights of new points.
```

Table I. The basic methods of the madd class.

A line command user interface is provided with the library. This interface program (`maddi`) can receive and execute a number of simple commands; the basic commands are are described in Table II. An example of a set of commands is:

```
read pipe.poly
set f=0.35
dec 400
write pipe400.poly
writeSubdomains pipe400.dat
exit
```

The user can set a number of optional parameters, through the `set` command, that controls various functional aspects, like the level of refinement, the level of smoothing, balancing, etc. The basic parameters are described in Table III.

The level of refinement is based on a user-defined uniform refinement factor r, and the $\sqrt{N}$, where N are the number of subdomains. The square root function 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 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. The refinement factor r defines a uniform refinement level,

```
read <filename>                     Reads a poly file.
dec [<noOfSubdomains>]              Decomposes the doamin.
write <filename> [all]              Writes the decomposed poly file.
read_weights <filename>             Reads weighted background mesh nodes from a file.
read_areas <filename>               Reads area background mesh nodes from a file.
writeSubdomains <filename>          Writes the decomposition into a file.
setWeightFunction <libraryName> <functionName>    Sets the weight function.
setAreaFunction <libraryName> <functionName>      Sets the area function.
set <parameter>=<value>             Sets the <value> to the <parameter>.
exit                                Exits.
```

Table II. The basic commands for the maddi interface.

for example, if a boundary segment is to be divided to four subsegments, a factor $r = 2$ will cause it to be divided into 8 subsegments. In all our experiments the values for r were between 2 and 4. The gradation factor a controls the level of gradation, in association with the uniform refinement factor r. The total density weight of the subdomain is computed as

$$r \times \text{subdomain\_area} + a \times \text{subdomain\_weight}.$$

In this way the relation between a and b controls the intensity of the gradation.

7. 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 [Shewchuk 1996], and has 54 boundary segments and 1 hole. The *Chesapeake bay* (Cbay) model defined from 13,524 points and it has 26 islands.

We ran three sets of experiments. In the first set of experiments we produced uniform decompositions for the three test domains. In the second set of experiments we produced graded decompositions, using the three approaches described in Section 5.2: (a) we used weights on the boundaries to produce graded decompositions, (b) we experimented using weight and area background meshes, and (c) we used weight and area functions over structured grids. The above experiments were performed on a a Pentium IV 3GHz processor, and we used a lower angle bound of $f = 0.3333$ rads ($\approx 60°$). A third set of experiments was performed in order to asses the quality and efficiency of the MADD and compare it to Metis, which is a state of the art graph partitioner. For these experiments we decomposed the Key geometry, using a Dual Pentium 3.4GHz processor.

Our 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. 14 -15). 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,

f	Defines the minimum angle bound created by the separators (in rads).
r	Defines the uniform refinement level. The expected length of the segments, after refining, is multiplied by $1/r$. Equivalently, the number of the segments after refining is multiplied by r.
a	The gradation factor for density weights. The density weight of the subdomains is multiplied by a, while the area is multiplied by the uniform refinement level r. The total sum gives the density weight of the subdomain.
p	The interpolation factor for density weights. The weights of the new points of the separators are computed by linear interpolation of the weights of the separator end points, multiplied by p. A value $p = 0$ eliminates the weights on the separators.
i	The level of acceptable imbalance during the smoothing. The values should be between 0.6 and 0.9 (default is 0.75).

Table III. The basic parameters used in the set command.

which has few initial segments, and requires more intense refinement in order to get good decompositions.

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 a gradation factor a, and the weight interpolation, p, that will be applied on the interfaces. The parameters for the Pipe and the Key were $r = 3, a = 3, p = 0.5$, while for the Cbay they were $r = 2, a = 3, p = 0$. One of the difficulties for decomposing a geometry based on boundary weights (and also weight or area functions) is that the boundary refinement will have to be adaptive on the local weights. In our implementation, segments that have greater weight will be refined more. These difficulties can be solved by a computing on the fly, locally and independently, the level of refinement required for each subdomain.

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. 12). 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. 13), while the cost to create them is small (see Fig. 17). 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.

For the third set of experiments we partitioned the Key geometry up to 2,000 subdomains uniformly, and we compare the results obtained by MADD to those obtained by Metis. For the Metis decompositions we created background Delaunay meshes of size approximately 120 triangles per subdomain, The Delaunay mesh generation procedure is the only one that provides quality guarantees, creating

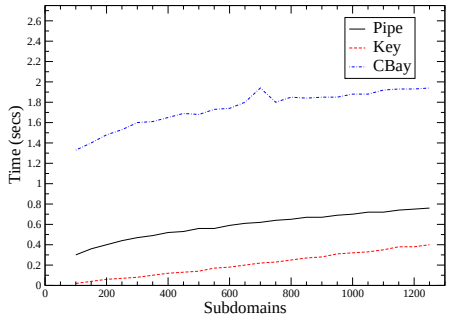

Fig. 14. Decomposition times for the uniform MADD.

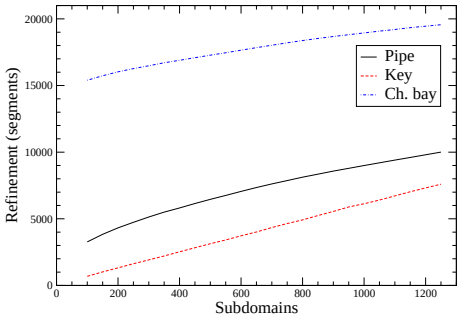

Fig. 15. The refinement (number of segments) for the uniform decompositions

angles no less than 30°. The background mesh was translated into a weighted graph, with weights reflecting the edge lengths and the triangle areas. For the MADD we used an adaptive local refinement approach to produce the Medial Axis approximation. The lower angle bound was set to 70°.

Figure 18 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. 19). The efficiency of the MADD depends on the geometry (Fig. 14), while the efficiency of Metis depends on the size of the background mesh. For the Key geometry MADD performs better (see Fig. 20), 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. 21), 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. 22. The maximum subdomain area is close to the average subdomain area for the MADD method (Fig 23), while Metis results almost perfect maximum subdomain area due to the near perfect balancing that it

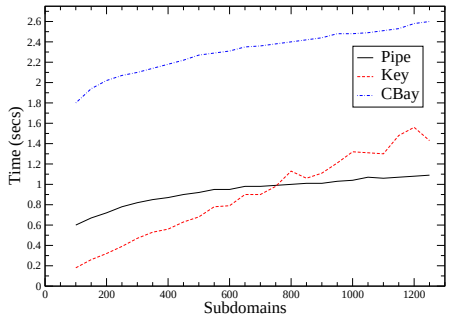

Fig. 16. Decomposition times for the weighted boundary MADD.

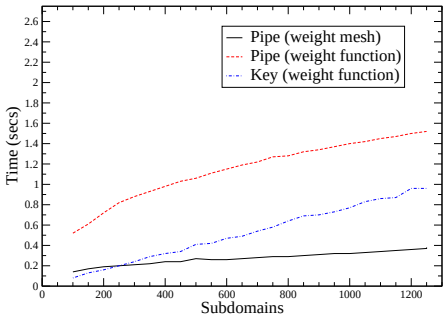

Fig. 17. Decomposition times for the graded MADD using a weighted background mesh and weight functions.

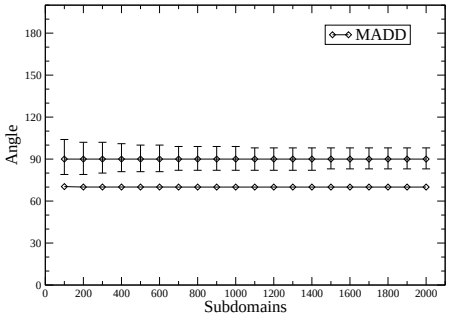

Fig. 18. The angles created by MADD. The minimum, median and 90% quantiles are depicted.

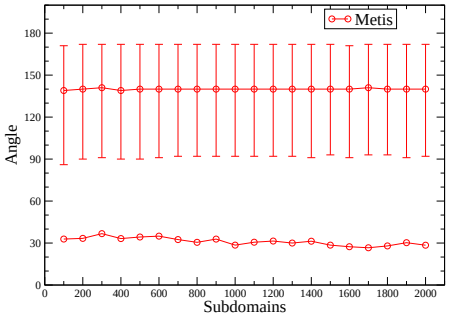

Fig. 19. The angles created by Metis. The minimum, median and 90% quantiles are depicted.

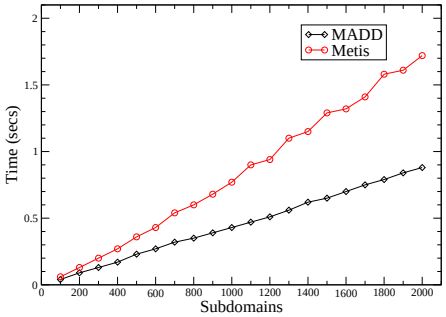

Fig. 20. The decomposition times for MADD and Metis. The mesh generation time is included.

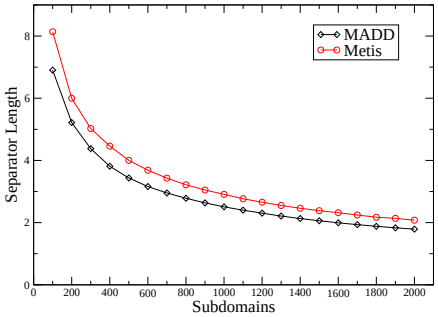

Fig. 21. The average separator length per subdomain.

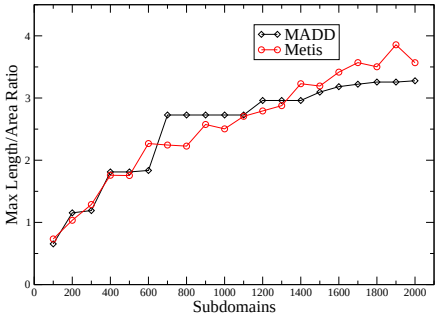

Fig. 22. The maximum ratio of subdomain separator length/subdomain area.

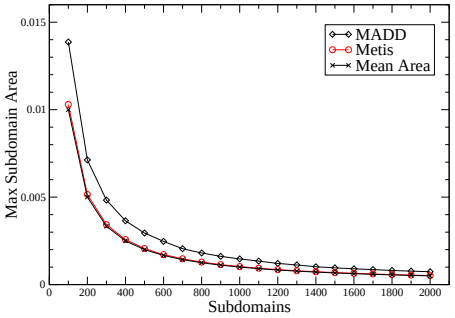

Fig. 23. The maximum area of the subdomains.

All figures refer to uniform decompositions of the Key geometry.

produces.

8. CONCLUSIONS AND FUTURE WORK

We propose a geometric domain decomposition method, based on the Medial Axis, which produces good domain decompositions in terms of three basic criteria: (1) The internal boundary of the subdomains forms good angles, i.e., angles no smaller than a given tolerance Φ_0. (2) The size of the separator is relatively small compared to the area of the subdomains. (3) The maximum area of the subdomains is close to the average subdomain area. The resulting decompositions do not create new artifacts in the geometry and are suitable for stable and efficient parallel mesh generation procedures. The MADD can also create graded decompositions, based on density grids, or density functions; these decompositions are useful for graded parallel mesh generation. The experimental data demonstrate that the method is efficient, generating more than a thousand subdomains in less than two seconds, and effective, resulting angles close to 90° and small length of separators.

Although it is straight forward to extend the MADD algorithm to three dimensions, using a tetrahedral Delaunay mesh, there are no guarantees of the quality of the resulting decomposition. The Voronoi points of the refinement do not converge to the Medial Axis in three dimensions, and the formed angles are not guaranteed to be close to 90°. However, we believe that the Medial Axis can be a useful tool for geometric domain decompositions in three dimensions, and this is a subject of future work.

ACKNOWLEDGMENTS

Two basic software libraries were used by our software: [Triangle], by Jonathan Shewchuk, for the Delaunay triangulation, and [Metis], by George Karypis et al., as the graph partitioner. We thank them for kindly providing these libraries. The copyright and rights of use, for [Triangle] and [Metis] are as stated in the respective software; the user interested in using MADD should download these two libraries directly form their web sites, observing the conditions of use stated there and in their source code. [Showme] and [mview] were used to produce the figures of the domains in this paper.

We would like to thank Professor Harry Wang and Dr. Mac Sisson from Virginia Institute of Marine Science for providing the data for the shoreline of the Chesapeake bay. Finally, we would like to thank the anonymous referees for their very useful suggestions, which improved this paper.

REFERENCES

Attali, D., Boissonnat, J.-D., and Edelsbrunner, H. Stability and computation of the medial axis — a state-of-the-art report. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration.*

Barker, K., Chrisochoides, N., Chernikov, A., and Pingali, K. 2004. A load balancing framework for adaptive and asynchronous applications. *IEEE Trans. Parallel and Distributed Systems 15,* 2 (Ferbruary), 183–192.

Barnard, S. T. and Simon, H. D. 1994. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience 6*, 101–107.

Berger, M. and Bokhari, S. 1985. A partitioning strategy for pdes across multiprocessors. In *Proceedings of the 1985 International Conference on Parallel Processing*.

Blum, H. 1967. A transformation for extracting new descriptors of shape. In *Models for the Perception of speech and Visual Form*. MIT Press, 362–380.

Borouchaki, H., George, P. L., Hecht, F., Laug, P., and Saltel, E. 1997. Delaunay mesh generation governed by metric specifications, Part I. Algorithms and Part II. Applications. *Finite Elements in Analysis and Design 25*, 61–83 and 85–109.

Brandt, J. W. and Algazi, V. R. 1992. Continuous skeleton computation by Voronoi diagram. *Comput. Vision, Graphics, Image Process. 55*, 329–338.

Chernikov, A. and Chrisochoides, N. 2004. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. In *Proceedings of the 18th annual international conference on Supercomputing*. Malo, France, 48–57.

Chew, L. P. 1989. Guaranteed quality triangular meshes. Tech. Rep. TR-89-983, Department of Computer Science, Cornell University.

Chew, L. P. 1993. Guaranteed-quality mesh generation for curved surfaces. In *9th Annual Symposium on Computational Geometry*. ACM, San Diego, California, 274–280.

Chew, L. P., Chrisochoides, N., and Sukup, F. 1997. Parallel constrained Delaunay triangulation. In *ASME/ASCE/SES Special Symposium on Trends in Unstructured Mesh Generation*. Evanston, IL, 89–96.

Chrisochoides, N. 1996. Multithreaded model for the dynamic load-balancing of parallel adaptive pde computations. *Applied Numerical Mathematics 20*, 349–365.

Chrisochoides, N. and Nave, D. 2003. Parallel Delaunay mesh generation kernel. *International Journal for Numerical Methods in Engineering 58,* 2, 161–176.

Deister, F., Tremel, U., Hasan, O., and Weatherill, N. P. 2004. Fully automatic and fast mesh size specification for unstructured mesh generation. *Engineering with Computers 20*, 237–248.

Dong, S. and Karniadakis, G. 2005. DNS of flow past a stationary and oscillating rigid cylinder at $re = 10{,}000$. *Journal of Fluids and Structures 20(4)*, 519–531.

Galtier, J. and George, P.-L. 1996. Prepartitioning as a way to mesh subdomains in parallel. In *5th International Meshing Roundtable*. Pittsburgh, Pennsylvania, 107–122.

Hendrickson, B. and Leland, R. W. 1995a. The Chaco user's guide version 2.0. Tech. rep., Sandia National Laboratories.

Hendrickson, B. and Leland, R. W. 1995b. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing 16,* 2, 452–469.

Hendrickson, B. and Leland, R. W. 1995c. A multi-level algorithm for partitioning graphs. In *Supercomputing*. San Diego, CA.

Kadow, C. and Walkington, N. 2003. Design of a projection-based parallel Delaunay mesh generation and refinement algorithm. In *4th Symposium on Trends in Unstructured Mesh Generation*.

Karypis, G. and Kumar, V. 1995a. A fast and high quality multilevel scheme for partitioning irregular graphs. Tech. Rep. TR 95-035, Department of Computer Science, University of Minnesota, Minneapolis.

Karypis, G. and Kumar, V. 1995b. *MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*.

Kernighan, B. W. and Lin, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal 49(2)*, 291–307.

Linardakis, L. and Chrisochoides, N. 2006. Delaunay Decoupling Method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing 27,* 4, 1394–1423.

Löhner, R. 1997. Automatic unstructued grid generators. *Finite Elements in Analysis and Design 25*, 111–135.

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

mview. `http://mview.sourceforge.net/`.

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

Owen, S. J. and Saigal, S. 1997. Neighborhood-based element sizing control for finite element surface meshing. *6th International Meshing Roundtable*.

Ruppert, J. 1995. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms 18,* 3, 548–585.

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

Shewchuk, J. R. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds. Lecture Notes in Computer Science, vol. 1148. Springer-Verlag, 203–222. From the First ACM Workshop on Applied Computational Geometry.

Showme. `http://www.cs.cmu.edu/~quake/showme.html`.

Smith, B., Bjrstad, P., and Gropp, W. 1996. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 1996., New York.

Triangle. `http://www.cs.cmu.edu/~quake/triangle.html`.

Walshaw, C., Cross, M., and Everett, M. G. 1997. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing 47,* 2, 102–108.

Zhu, J., Blacker, T., and Smith, R. 2002. Background overlay grid size functions. In *11th International Meshing Roundtable*. Sandia National Laboratories, 65–74.