A new fast hybrid adaptive grid generation technique for arbitrary two-dimensional domains

Mohamed Ebeida[1,†], Roger L. Davis[2,‡], and Roland W. Freund[3,§]

[1] *Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.*
[2] *Department of Mechanical and Aeronautical Engineering, University of California Davis, One Shields Avenue, Davis, CA 95616, U.S.A.*
[3] *Department of Mathematics, University of California Davis, One Shields Avenue, Davis, CA 95616, U.S.A.*

SUMMARY

This paper describes a new fast hybrid adaptive grid generation technique for arbitrary two-dimensional domains. This technique is based on a Cartesian background grid with square elements and quadtree decomposition. A new algorithm is introduced for the distribution of boundary points based on the curvature of the domain boundaries. The quadtree decomposition is governed either by the distribution of the boundary points or by a size function when a solution-based adaptive grid is desired. The resulting grid is quad-dominant and ready for the application of finite-element, multigrid, or line-relaxation methods. All of the internal angles in the final grid have a lower bound of 45° and an upper bound of 135°. Although our main interest is in grid generation for unsteady flow simulations, the technique presented in this paper can be employed in many other fields. Several application examples are provided to illustrate the main features of this new approach.

KEY WORDS: grid generation; quadtree; quad-dominant; adaptive; finite-element method; multigrid method; line-relaxation method; computational fluid dynamics

1. INTRODUCTION

Grid generation is an important first step in the solution of many computational problems, especially problems arising in scientific computing and computer graphics. For example, the accuracy of the numerical solution of partial differential equations depends on the quality of the grid used in the discretization of the problem, especially on the grid density, the distribution of the grid points, and the quality of the elements connecting these points. For simulations in computational fluid dynamics (CFD), an optimal grid is one that accurately captures all the

[†]E-mail: msebeida@andrew.cmu.edu
[‡]E-mail: davisrl@ucdavis.edu
[§]E-mail: freund@math.ucdavis.edu

main features of the geometry and the fluid flow with the minimum number of grid points and elements.

Grids are either structured or unstructured. For two-dimensional domains, structured grids are composed of quadrilateral elements, while unstructured grids may be composed of elements with any number of sides, but, in most cases, consist of triangular elements. Each element type has its own advantages and disadvantages. Structured grids with quadrilateral elements have been shown to be superior in capturing viscous flows adjacent to solid surfaces due to their orthogonality and somewhat insensitivity to large aspect ratios in the dominant flow direction. However, because grid lines must eventually extend across the entire grid-block or domain to maintain certain block structures that are required for some solution methods, additional grid points are often placed in regions where they are not required to resolve the flow accurately. On the other hand, triangular elements can cover any regular or irregular domain efficiently, but stretching these elements affects the accuracy of the solution to a large extent. As a result, use of unstructured grids with triangular elements in high-gradient regions along a dominant flow direction, such as flow through a boundary layer or across a shock, is limited. In order to resolve this problem, one usually employs hybrid grids that use quadrilateral elements wherever stretching is required. In addition, unstructured grids put some limitations on the use of multigrid methods, which are well known to be efficient and relatively simple techniques for the acceleration of flow solutions. There are other serious problems with hybrid grids that are dominated by triangular elements. For example, a larger amount of work is required per time step due to the larger number of faces and edges for such grids. Moreover, the generation of such grids usually requires more computational work and the required data structures consume more memory. The highly-referenced handbook [1] on grid generation summarizes these challenges as follows: *There are thus very clear incentives to use structured grids whenever possible. For hybrid grids the implication is that the extent of unstructured grid employed should be as minimal as possible.*

During the last two decades many remeshing techniques have been developed. Recently, various papers have addressed the generation of surface grids [2, 3]. Surface remeshing is the most widely used technique for the generation of high-quality surface meshes [4, 5, 6].

Most of the computing time required to simulate a certain flow problem is consumed by the solution of a linear system of equations. Hence, the use of an efficient solver is critical. As has been demonstrated in the CFD community, multigrid methods are among the best algorithms for the solution of these linear systems. In multigrid approaches, convergence acceleration is achieved by using successively coarser grids, where the errors associated with the high frequencies are damped by a carefully chosen smoother on the fine grid levels, while the errors associated with the lower frequencies are damped on the coarser grid levels and quickly propagated out of the domain. In the case of structured grids, geometric multigrid methods [7] are easily implemented where coarser grids are derived from a given fine grid by dropping every other grid line in each coordinate direction. Of course, such a straightforward implementation is not possible in the case of hybrid unstructured grids.

Algebraic multigrid (AMG) methods have been proposed as an efficient and more flexible alternative to geometric multigrid methods. When using unstructured grids, there are three different approaches that can be adopted for AMG. The first approach resembles the classical grid-sequencing technique, where a coarse level grid is first generated and the refined grid levels are obtained by repeated refinements [8, 9]. The second approach employs non-nested unstructured grids either with a subset of fine grid points to construct the coarse grids or

with completely independent coarse and fine grids. This has been shown to be successful for both inviscid and viscous flow computations [10, 11]. The third approach uses coarse grids that are obtained from a coarsening of the fine grid through agglomeration or fusion, resulting in polyhedral coarse grid control volumes. In 1992, Lallemand et al [12] devised a technique to generate coarser grid levels using topological neighboring relations for cell-vertex schemes. In 2001, the AMGE (AMG for finite elements) method based on element agglomeration was proposed by Jones and Vassilevski [13]. This approach was further refined to handle inviscid and viscous flows past complex configurations in both two and three dimensions [14, 15].

Although the first approach can be utilized in an adaptive procedure and has simple inter-grid operators, it suffers from the dependence of the fine grid distribution on the coarse levels. This is avoided in the second approach, but with the added complication of the inter-grid operators and the requirement to generate multiple grids that preserve the geometry. Although the third approach is the most popular one, it produces coarse grids of higher complexity. This increases the cost of multigrid V-cycles and makes the use of W-cycles impractical [16].

Over the last two decades, there has been a great deal of interest in adaptive methods in the engineering community. For some flow applications, such methods are crucial because of the pressing need for accurate computation of flows with variable density, for example in cavitating flows, or shock waves that might occur in compressible flows; see, e.g., [17, 18, 19]. Adaptive methods offer a means of tackling complex flow problems at a reasonable cost and of controlling the accuracy of numerical simulations.

Adaptive grids based on quadtree decomposition methods can be employed to efficiently refine the generated grid in regions where more accuracy is required. In some grid-embedding schemes, refinement can introduce hanging nodes [20]. Hanging nodes limit the use of finite-element discretizations, and for finite-volume discretizations, the values of the variables at these points are usually obtained via some form of interpolation, which is not always conservative and can introduce additional discretization errors. The application of spatial decomposition methods solve these problems by nesting these hanging nodes to the grid using triangular elements, but an optimization step is then required to enhance the quality of the triangular elements, especially those defining any internal boundaries. This usually leads to a grid that is dominated by triangular elements [21]. The problem with local refinements of such triangular element-dominant grids is that an optimization step is required after each local refinement to enhance the minimum angle condition for the triangular elements on the edges of the refined region. The optimization step usually involves moving grid nodes with retriangulation of the grid elements [22]. Some optimization approaches involve the creation of new nodes and nesting them to the original grid without moving any node. This approach suffers from placing nodes where they might not be needed for increasing the accuracy of the solution [23]. Also, once a retriangulation is performed, then all the parent-child relations between grid levels required for multigrid methods are no more valid. Thus the generation of grid levels has to be performed from scratch with each adaptive refinement followed by a retriangulation step.

In this paper, we describe a new fast adaptive procedure for generating grids for arbitrary two-dimensional domains, together with data structures that are suitable for the application of multigrid solvers. The grids produced by this approach are dominated by quadrilateral elements. Refinement of the elements is constrained to ensure the high quality of the grids, without the need to move any node and/or perform retriangulation. The procedure is applied in a way that makes the dependence of the fine-grid node distribution on the coarse levels an advantage, because it allows a restriction of the refinement to certain subregions within the

given domain. At each refinement level, we store the grid elements and update the parent-child tree for the newly added nodes. As a result, upon completion of the refinement steps, all the grid-level information required for multigrid solvers is readily available.

Spatial decomposition is employed in such a way that only a very simple optimization step is needed to represent the boundaries of the domain and to create a nearly orthogonal grid around them. Moreover, without optimization, most of the grid lines are either horizontal or vertical. Hence, the optimization step is required only when these lines fail to define the boundaries of the domain adequately.

The development of the technique proposed in this paper was mainly motivated by the need to generate grids that are suitable for viscous flow simulations. At high Reynolds number, the boundary layer requirements are achieved by successive one-dimensional refinement, i.e., by splitting only the edges normal to those solid surfaces modeled with the no-slip condition. This refinement is utilized to rapidly reduce the length of these edges. For example, the resulting grids allow the accurate capturing of turbulent flow features inside the boundary layer, using only a small number of grid points.

Further adaptive refinements can be performed based on flow features identified via solution gradients or based on variables that quantify the local error in the solution. The quality of the elements during this type of refinement are guaranteed by constraining the possible resulting grid element geometry.

The motivation for the grid generation technique proposed in this paper was to develop a procedure that meets the following requirements:

- The procedure is fast and adaptive.
- The generated grids are suitable for the application of multigrid solvers, finite-element methods, and line-relaxation techniques.
- Most of the domain is covered by square elements for optimal quality.
- All the internal angles in the final grid have a lower bound of 45° and an upper bound of 135°.

The main steps of our grid generation technique are as follows:

1. linear representation of the boundaries of the domain and distribution of boundary points,
2. refinement of the background Cartesian grid based on quadtree decomposition and the distribution of boundary points,
3. optimization of near-boundary elements, which includes
   (a) buffer zone creation around each geometry and removal of elements that lie inside the buffer zone or outside the desired domain,
   (b) covering the gap between the terminal elements and the boundaries of the domain,
4. grid adaptation based on some error function during numerical simulation.

The remainder of this paper is organized as follows. In Section 2, we describe an algorithm for representing the boundaries of any two-dimensional domain. In Section 3, we present a procedure that generates a base grid of the domain, given the linear representation of the domain boundaries. In Section 4, we describe how to modify the base grid so that it respects the boundaries of the domain without deteriorating the quality of the elements in the final

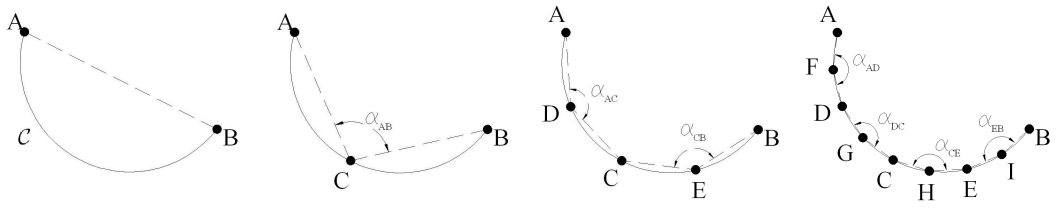
grid. In Section 5, we present an algorithm for solution-based adaptation of the grid. Section 6 discusses the use of the proposed grid generation technique within multigrid solvers. Finally, we make some concluding remarks in Section 7.

2. REPRESENTATION OF THE DOMAIN BOUNDARIES

In this section, we describe an algorithm for representing the boundaries of any given two-dimensional domain linearly. The distribution of the boundary points in the representation produced by this algorithm depends on the curvature of the boundaries and on the desired upper bound for the maximum edge length in the final linear representation.

To illustrate the main idea of this algorithm, consider the smooth curve $\mathcal{C}$ in Figure 1. This curve $\mathcal{C}$ is given by its end points, **A** and **B**, and the smooth parametrization $(x(t), y(t))$, $0 \leq t \leq 1$, of the points $(\mathbf{x}, \mathbf{y}) \in \mathcal{C}$. Here, $t = t_A = 0$ and $t = t_B = 1$ correspond to the end points **A** and **B**, respectively. As a first linear approximation ($n = 0$) to that curve we simply use the line segment **AB** connecting **A** and **B**; see Figure 1(a). If $\mathcal{C}$ has zero curvature, then this first approximation is exact. Otherwise, we may need a better, piecewise linear approximation. To this end, we refine the line segment **AB** by replacing it with the two-segment polyline, $\mathcal{P}^1$, that consists of the two line segments **AC** and **CB**, where $\mathbf{C} \in \mathcal{C}$ and $t_C = (t_A + t_B)/2$. Let $0 \leq \alpha \leq 180^\circ$ denote the the angle between the two new line segments (**AC** and **CB**). We refer to α as the *refinement angle* of the old line segment **AB**. If necessary, we continue with this refinement procedure to obtain an n-segment polyline $\mathcal{P}^n$, where n denotes the iteration index. Obviously, the larger n, the better $\mathcal{P}^n$ approximates the curve $\mathcal{C}$. Note that the refinement angle of any line segment in $\mathcal{P}^n$ approaches 180° as $n \to \infty$. This fact is used in the following point-distribution algorithm, where we use a parameter, ϵ, to control the termination of the iterative refinement procedure. More precisely, we consider a line segment to be a good representation of the curve passing through its end points if the refinement angle α corresponding to that line segment satisfies

$$180^\circ - \epsilon \leq \alpha \leq 180^\circ.$$



(a) $n = 0$ (b) $n = 1$ (c) $n = 2$ (d) $n = 3$

Figure 1. Linear representation of a curve $\mathcal{C}$

Algorithm 1. (Initial distribution of boundary points)
Input: *A set of smooth curves* $\{\mathcal{C}_i\}$ *representing the boundaries of the domain, where each curve has a parametric representation of the form* $(x(t), y(t))$*, a parameter* $\epsilon > 0$ *(in degrees) that controls the termination of the refinement iterative procedure, and a parameter* $\delta > 0$ *that controls the maximum length of the boundary edges.*

1. *For each curve* $\mathcal{C}_i$*, do:*

(a) (Creation of initial polyline $\mathcal{P}_i^0$)
Set a point **A** *at the beginning of the curve* ($t_A = 0$)*, a point* **B** *at its end* ($t_B = 1$)*, and initialize* $\mathcal{P}_i^0$ *to be the line segment* **AB**. *Note that for a closed curve, the points* **A** *and* **B** *are identical and can be placed anywhere on that curve.*

(b) *For* $n = 1, 2, 3, \ldots,$ *do:*

- *For each line segment in* $\mathcal{P}_i^{n-1}$*, do:*
  - *Calculate the refinement angle* $0 \le \alpha \le 180^\circ$ *corresponding to this edge.*
  - *If* $\alpha < 180^\circ - \epsilon$*, refine this line segment.*
  - *Otherwise, for all internal points of* $\mathcal{P}_i^{n-1}$ *do:*
    - *Calculate the angle* $0 \le \beta \le 180^\circ$ *between the two boundary line segments intersecting at this internal point.*
    - *If* $\beta < 180^\circ - \epsilon$*, refine both these boundary line segments.*

    (An internal point is a boundary point that is part of two boundary line segments.)
  - *If* $\mathcal{P}_i^n = \mathcal{P}_i^{n-1}$*, continue with step* (c).

(c) (Controlling the maximum length of the line segments)
For all line segments in $\mathcal{P}_i^n$*, do:*

- *Calculate the length* l_e *of that line segment.*
- *If* $l_e > \delta$*, refine that line segment.*
- *If no line segment was refined, proceed to step* 2*: The linear approximation* $\mathcal{P}_i^n$ *is acceptable.*

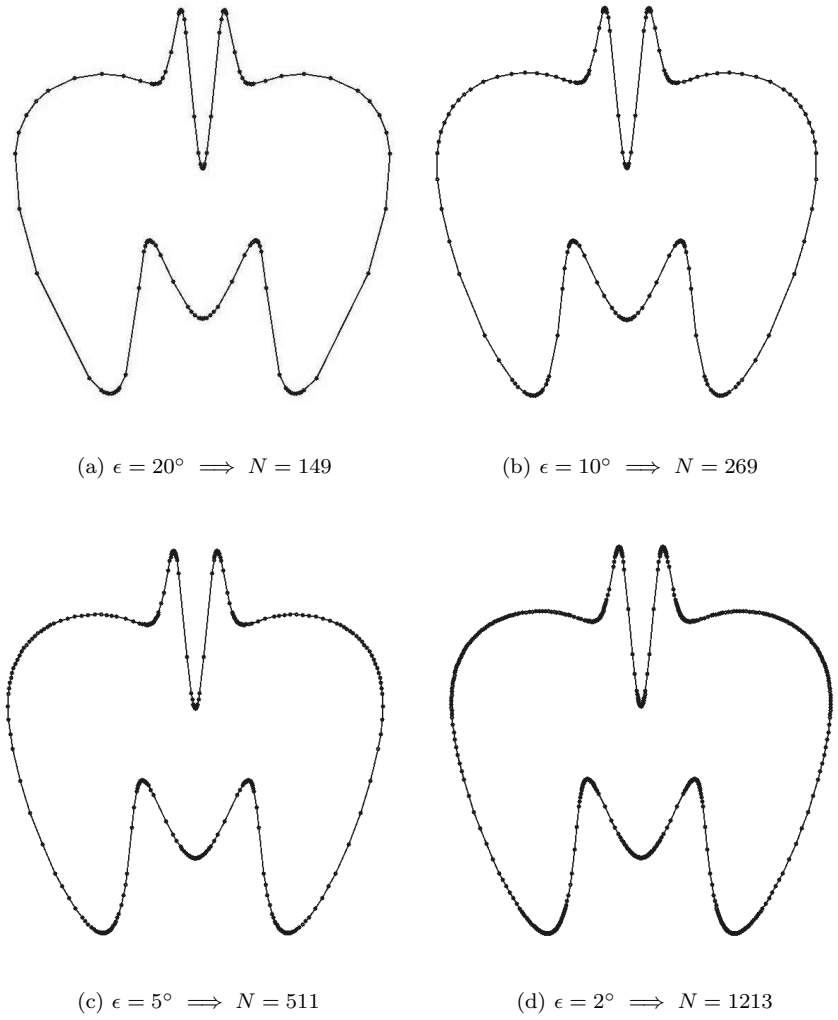
2. *Merge all the polylines with the same end points and relocate the two points surrounding any internal point associated with some sharp features such that the two line segments intersecting at that point will have the same length.*
(One way to do that is to shorten the longer edge without changing its slope.)

Output: *A set of polylines* $\{\mathcal{P}_i\}$*. Each polyline represents a closed part of the domain boundaries and is defined by a set of boundary points* $\{\mathcal{N}_i\}$ *and a set of line segments* $\{\mathcal{E}_i\}$*.*

In the following, we will often refer to the line segments in $\mathcal{E}_i$ as *edges*. We remark that step (c) in Algorithm 1 is optional. This step has no effect on the final output if we choose $\delta > l_c$, where l_c is the length of the curve to be approximated. We include this step when the final grid is to be used in turbulent flow simulations. In such cases, an upper bound for the edge length is enforced to ensure that the requirements of the turbulent model are satisfied when generating the grid in the boundary layer region. Sometimes we may enforce an additional constraint. For

instance, we may refine the edges around those parts of the boundaries associated with sharp features. These extra refinements are necessary to increase the accuracy of the simulation in these critical regions.

Next, we present some numerical examples in order to illustrate the behavior of Algorithm 1 and show the effects of the choice of the parameter ϵ on the number and the distribution of the boundary points. The first domain, $\mathcal{D}_1$, was chosen to test the adaptivity and the symmetry of the algorithm. The boundary of $\mathcal{D}_1$ consists of a single smooth curve with varying curvature. This curve is symmetrical with respect to the y-axis. As shown in Figure 2, the number of boundary points, N, increases as ϵ decreases and symmetry is always conserved.

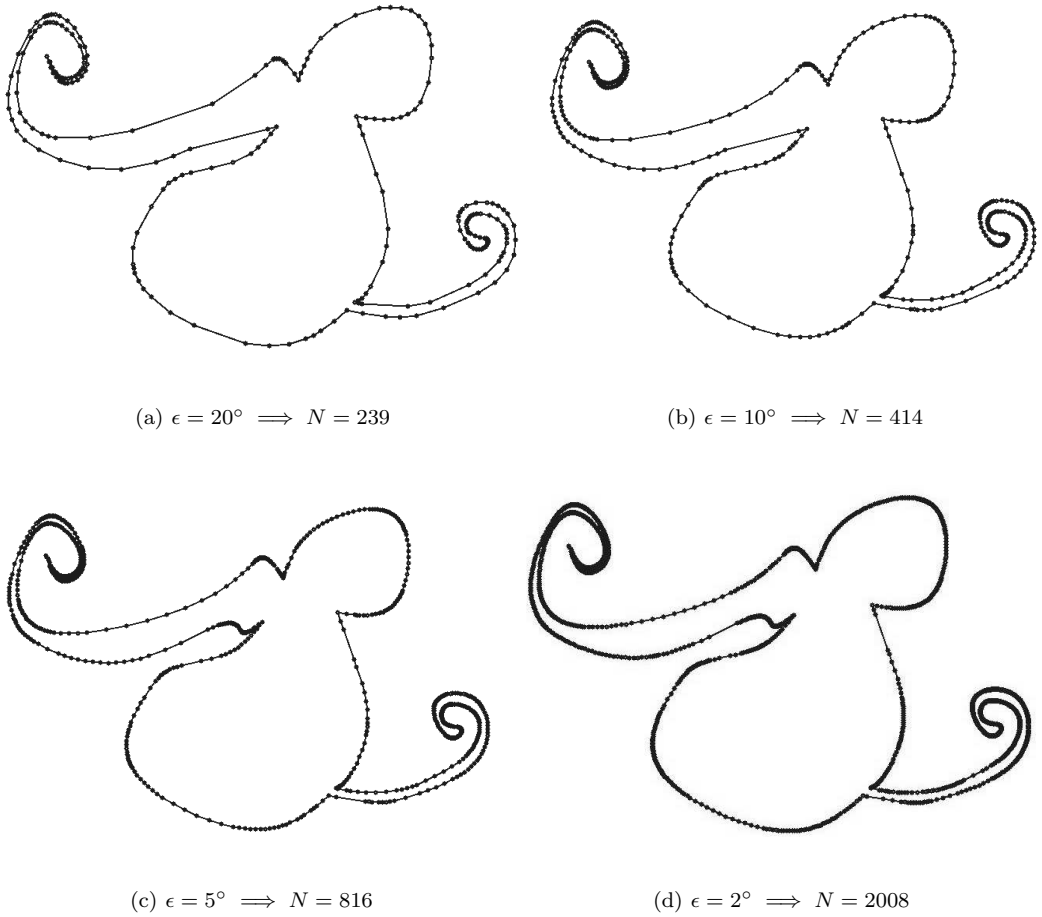


(a) $\epsilon = 20^\circ \implies N = 149$ (b) $\epsilon = 10^\circ \implies N = 269$

(c) $\epsilon = 5^\circ \implies N = 511$ (d) $\epsilon = 2^\circ \implies N = 1213$

Figure 2. Linear representation of $\mathcal{D}_1$ using different values for ϵ.

The second domain, $\mathcal{D}_2$, was selected to test the ability of the algorithm to deal with domain boundaries with singular points. The boundary of $\mathcal{D}_2$ consists of 6 smooth curves. Figure 3 shows the linear representations of $\mathcal{D}_2$ generated by Algorithm 1 when different values for ϵ are used. Clearly, as ϵ decreases, the ability of the algorithm to capture more details increases.



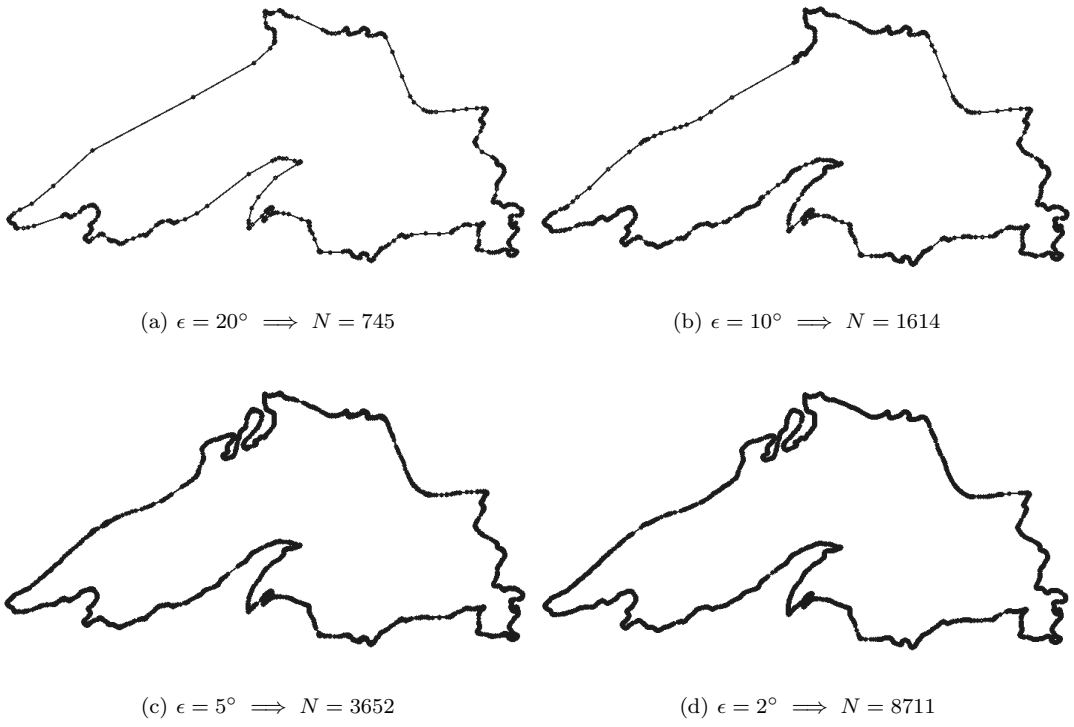
(a) $\epsilon = 20^\circ \implies N = 239$ (b) $\epsilon = 10^\circ \implies N = 414$

(c) $\epsilon = 5^\circ \implies N = 816$ (d) $\epsilon = 2^\circ \implies N = 2008$

Figure 3. Linear representation of $\mathcal{D}_2$ using different values for ϵ.

To illustrate the effect of the choice of ϵ on capturing the details of the boundaries, we used one of the great lakes as domain $\mathcal{D}_3$. This domain has a great deal of detail along its boundaries. The boundaries of $\mathcal{D}_3$ are treated here as a single smooth curve with a wide range of curvature values. Figure 4 shows the linear representations of $\mathcal{D}_3$ generated by Algorithm 1 when different values for ϵ are used. Again, using lower values for ϵ increases the ability of the algorithm to capture more details.

Finally, to test the ability of Algorithm 1 to deal with domains for computational fluid



(a) $\epsilon = 20^\circ \implies N = 745$ (b) $\epsilon = 10^\circ \implies N = 1614$

(c) $\epsilon = 5^\circ \implies N = 3652$ (d) $\epsilon = 2^\circ \implies N = 8711$

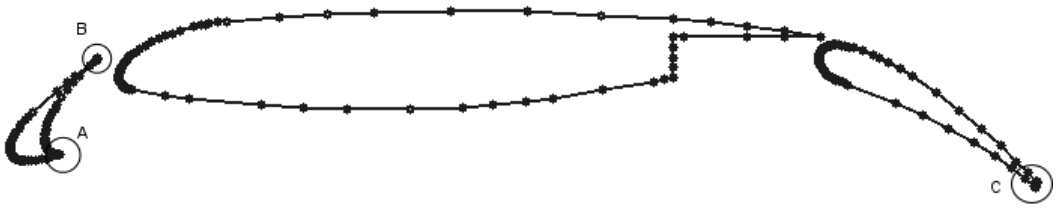
Figure 4. Linear representation of a $\mathcal{D}_3$ using different values for ϵ.

dynamics (CFD) simulations, we use the domain $\mathcal{D}_4$ of a multi-element airfoil. The overall airfoil length is almost one unit. The boundary of $\mathcal{D}_4$ consists of multiple closed curves, where each curve contains at least two singular points. Figure 5 shows the linear representation of the boundary of $\mathcal{D}_4$ that is produced by Algorithm 1 with parameter values $\epsilon = 4^\circ$ and $\delta = 0.1$. In this example, we forced each straight part of the boundaries to be represented by at least 5 boundary points.

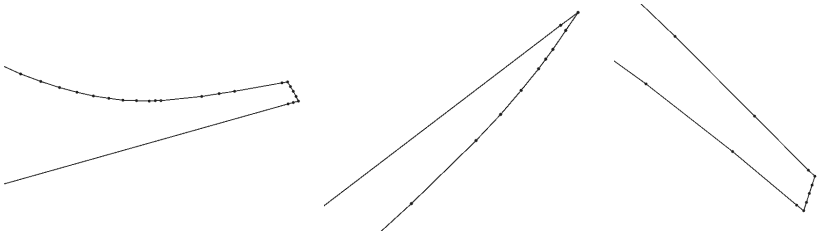
3. SPATIAL DECOMPOSITION

In this section, we present a procedure that generates a base grid of the domain, using the linear representation of the domain boundaries produced by Algorithm 1 as input. We refer to the boundary points of the linear representation as the refinement points in this section.

The spatial decomposition described in this section is based on the quadtree refinement algorithm. In the standard quadtree algorithm [24, 25], the iterative refinement procedure starts with a Cartesian grid of square elements and a certain distribution of refinement points. An element is then refined if it contains more than one refinement point. This algorithm is fast, but sensitive to the orientation of the refinement points. For example, Figure 6(a) shows a



(a) $N = 131 + 100 + 94$



(b) Zoom A (c) Zoom B (d) Zoom C

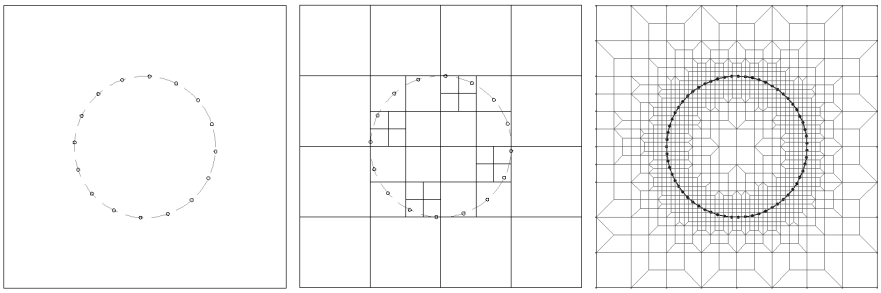
Figure 5. Linear representation of $\mathcal{D}_4$ using $\epsilon = 4^\circ$ and $\delta = 0.1$.

distribution of boundary points representing a circle. Here the curvature is fixed, so the length of all the boundary edges are the same. However, as Figure 6(b) illustrates, the output of the standard quadtree algorithm suffers from the following problems:

- The constant curvature of the circular boundary does not guarantee elements of the same size near that boundary.
- Large jumps in the element size may occur. In CFD simulations, we prefer to limit the size ratio of any two neighboring elements to an upper bound of 2.
- The standard quadtree algorithm produces a large number of hanging nodes. A hanging node is one that exists in one of the elements without being one of its corners. Such nodes are undesirable in finite-element and finite-volume methods.

We modified the standard quadtree algorithm to eliminate these problems. For the above example, the spatial decomposition produced by our algorithm for a circular boundary represented with more refinement points is shown in Figure 6(c).

Our algorithm starts with a background Cartesian grid with square elements. This grid has to contain all the refinement points. The goal is to refine the square elements iteratively so that in the final grid, $\mathcal{G}_b$, an element crossing the boundaries will have a size that is bounded from above by the distance between its nearest two refinement points and bounded from below by half that distance. The refinement iteration is done such that the size ratio between any two neighboring elements is guaranteed to be less than or equal to 2. The size of an element



(a) Refinement points (b) Standard quadtree (c) Our spatial decomposition

Figure 6. Spatial decomposition around a circular boundary.

here is measured using the maximum length of its edges. In order to eliminate the hanging nodes, we allow partial refinements of the square elements, as illustrated in Figure 7. This process results in the formation of *transition elements*, which are generated from refining one or two edges only of a square element. A transition element is either a trapezoid or a triangle. Our refinement algorithm ensures that no transition element will be refined during the entire refinement procedure. Also, for each refinement iteration, we refine any element that contains a singular node. In that way, we ensure that such a node is contained in the smallest elements of the final grid.

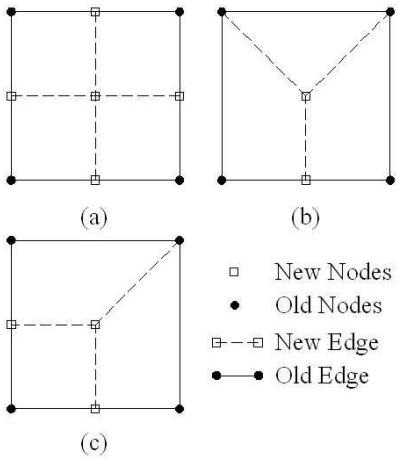


Figure 7. Refinement of square elements.

The following algorithm states our procedure for generating a base grid.

Algorithm 2. (Spatial decomposition)
Input: *A set of refinement points* $\{\mathcal{N}_i\}$*, a set of edges* $\{\mathcal{E}_i\}$*, an initial spacing* $s_0 > 0$*, and a minimum edge size* $s_m > 0$ *.*

1. *Construct a background grid* $\mathcal{G}_0$ *such that it contains all the edges in* $\{\mathcal{E}_i\}$ *and all its elements are squares with edge length of* s_0*.*

2. *For* $n = 1, 2, 3, \ldots,$ *do:*

(a) *Create an empty list* $\mathcal{L}$*.*

(b) *For the square elements that cross the boundaries in* $\mathcal{G}_{n-1}$*, do:*

- *Set the size,* s*, of that element to be the edge length in that element.*
- *Determine the distance,* d*, between the nearest two refinement points.*
- *If* $s > d$*, then add all the corner nodes of that element to* $\mathcal{L}$*.*
- *If an element is crossed by two boundary edges and the angle between these two edges is greater than* $90°$*, then add all the corner nodes of that element to* $\mathcal{L}$*.*
- *If an element has a first-level neighboring element that is crossed by a boundary edge and the angle between these two edges is greater than* $90°$*, then add all the corner nodes of both elements to* $\mathcal{L}$*.*
- *If an element has a second-level neighboring element that is crossed by a boundary edge and the angle between these two edges is greater than* $90°$*, then add all the corner nodes of both elements, as well as the element between them, to* $\mathcal{L}$*.*
- *If an element has a third-level neighboring element that is crossed by a boundary edge and the angle between these two edges is greater than* $90°$*, then add all the corner nodes of both elements, as well as the two elements between them, to* $\mathcal{L}$*.*

(c) *For the square elements in* $\mathcal{G}_{n-1}$ *that contain a singular point, add all the corner nodes of these elements to* $\mathcal{L}$*.*

(d) *For the square elements in* $\mathcal{G}_{n-1}$ *with at least one corner in* $\mathcal{L}$ *and with no more than three corners in* $\mathcal{L}$*, add all the corner nodes of these elements to* $\mathcal{L}$*.*

(e) *For the square elements with three corners in* $\mathcal{L}$*, add the fourth corner to* $\mathcal{L}$*, until there is no such element left.*

(f) *For all square elements with at least one corner in* $\mathcal{L}$*, do:*

- *Count the number of edges with both ends in* $\mathcal{L}$*.*
- *Refine that element based on the number of edges to be refined.*

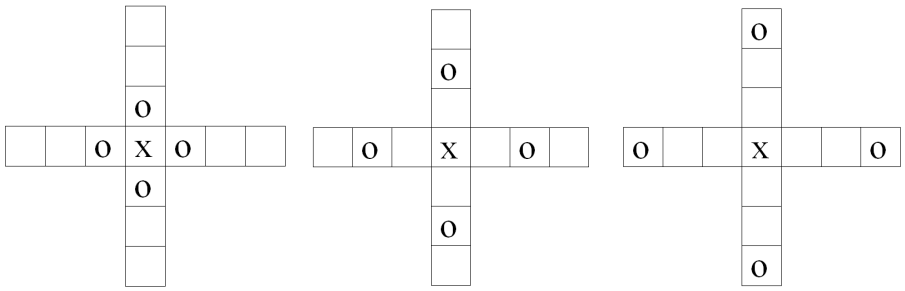
(g) *Set* $\mathcal{G}_n$ *to be the refined grid.*

(h) *If* $\mathcal{G}_n = \mathcal{G}_{n-1}$*, stop:* $\mathcal{G}_n$ *is the final grid.*

(i) *If the sizes* s *of all square elements in* $\mathcal{G}_n$ *satisfy the condition* $s < 2s_m$*, stop:* $\mathcal{G}_n$ *is the final grid.*

Output: *A base grid* $\mathcal{G}_b = \mathcal{G}_n$ *that is adapted to the curvature of the domain boundaries.*

We remark that the narrow regions of the boundaries are captured via the refinement of neighboring elements crossed by the boundaries. Figure 8 shows the first-level, second-level, and third-level neighbors of an element. This refinement procedure ensures the existence of at least three square elements inside a narrow region. The minimum edge size, s_m, is a parameter that controls the smallest size of the elements inside narrow regions. Note also that all the edges are oriented such that the end of an edge is the start of another one, so the angle, α, between two edges varies from zero (for two edges on the same straight line) and goes up to 180° at a very sharp corner. Finally, the purpose of step 2(d) in Algorithm 2 is to refine any extra layers of square elements in order to eliminate the possibility of refining any edge of a transition element through the whole process. Hence, this step ensures the minimum angle requirement of 45° and guarantees that the jump ratios in the size of first-level neighboring elements are bounded by 2.



(a) First Level (b) Second Level (c) Third Level

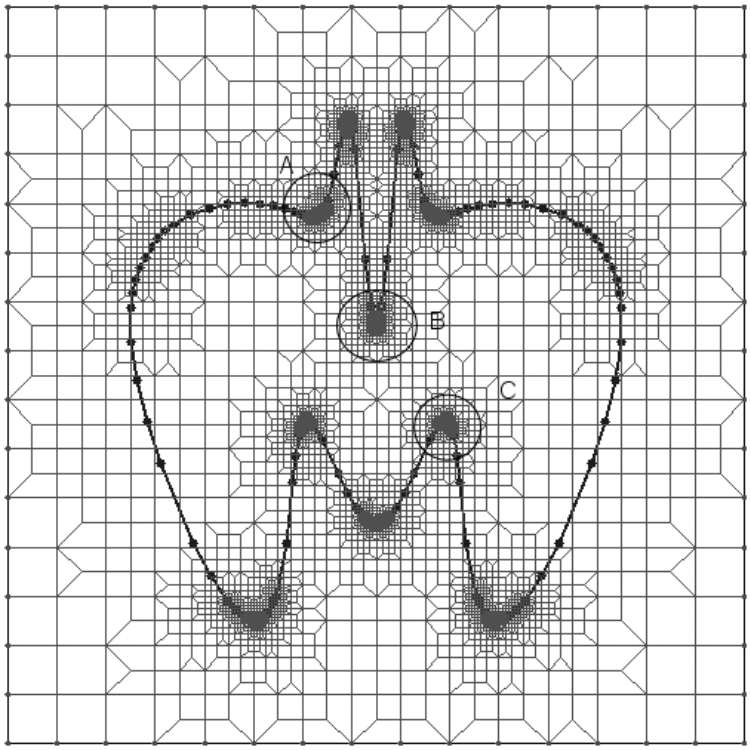
Figure 8. Neighbors (marked with "o") of an element (marked with "x").

In order to test Algorithm 2 and illustrate its feature to adapt to the curvature of the domain boundaries, we ran the algorithm with a point distribution for domain $\mathcal{D}_1$ as input. Figure 9 displays the spatial decomposition obtained for $\mathcal{D}_1$.

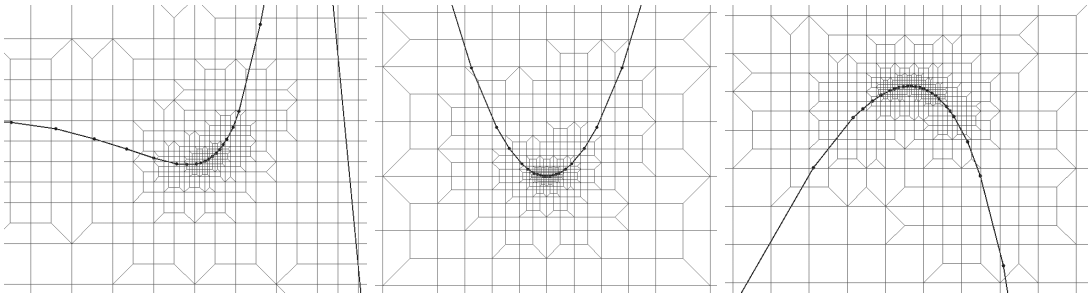
In order to test the ability of Algorithm 2 to capture the narrow regions of the domain, we ran the algorithm with a point distribution for domain $\mathcal{D}_2$ as input. Figure 10 displays the spatial decomposition obtained for $\mathcal{D}_2$.

4. GRID OPTIMIZATION

Up to this point, we have approximated the boundaries of the domain $\Gamma(\Omega)$ linearly. We will refer to the approximated boundaries as $\Gamma^*(\Omega)$. We also have a base grid, $\mathcal{G}_b$, that is adapted to the curvature of $\Gamma(\Omega)$. This base grid fills a bounding box around the domain, Ω, and has a large number of points outside of that domain. We need to finalize the grid through an optimization step that will eliminate all these exterior points. This is the most important step of the whole technique. In this step, we modify the base grid, $\mathcal{G}_b$, to represent the boundaries of the domain using the edges of the grid elements.

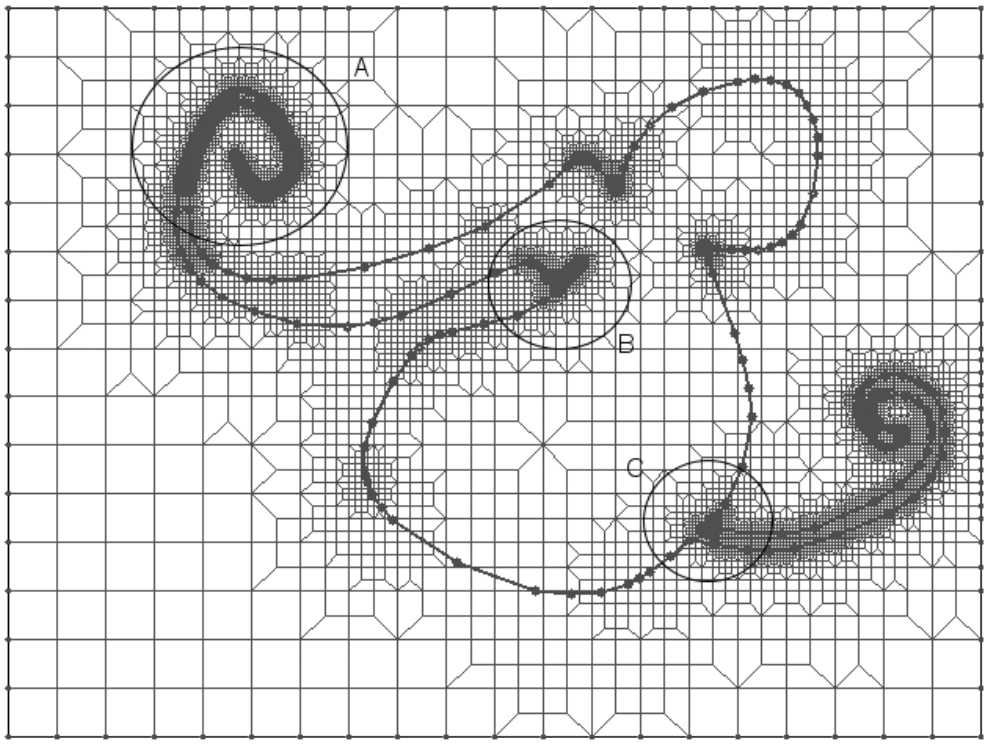


(a) The whole domain

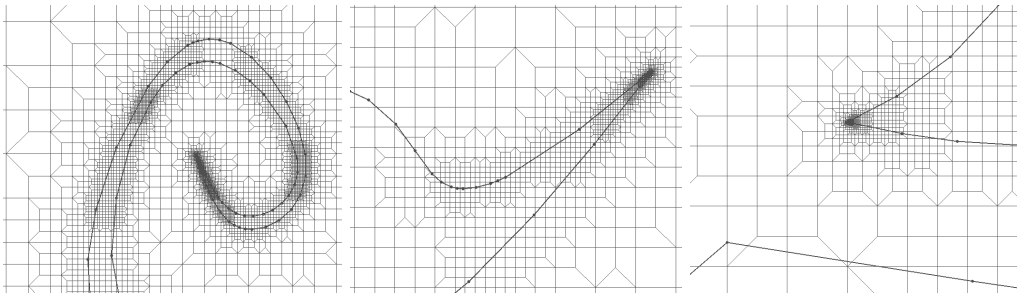


(b) Zoom A (c) Zoom B (d) Zoom C

Figure 9. Spatial decomposition of $\mathcal{D}_1$.



(a) The whole domain



(b) Zoom A (c) Zoom B (d) Zoom C

Figure 10. Spatial decomposition of $\mathcal{D}_2$.

4.1. Buffer zone creation

A buffer zone, $\mathcal{Z}$, that contains $\Gamma^*(\Omega)$ is created and any element with a node, $\mathbf{n}$, such that $\mathbf{n} \in \mathcal{Z}$ is to be deleted. The distance, d_z, between the buffer zone and $\Gamma^*(\Omega)$ at any point $\mathbf{x} \in \Gamma^*(\Omega)$ is prescribed using the size of the nearest element to that point. The size of any element is defined by its minimum edge length, s_e. We set $d_z(\mathbf{x}) = \frac{1}{2}s_e(\mathbf{x})$ to be sure that the new elements created to cover the gap between the terminal edges and $\Gamma^*(\Omega)$ will not be stretched.

Let $\mathcal{N}_\mathcal{Z}$ be the set of external nodes, i.e., all nodes that exist inside the buffer zone, and let $\mathcal{E}_\mathcal{Z}$ denote the set of elements that have at least one corner node in $\mathcal{N}_\mathcal{Z}$. We now define another set, $\mathcal{N}_\mathcal{T}$, for the terminal nodes as follows:

$$\mathcal{N}_\mathcal{T} := \left\{ \mathbf{n} \;\middle|\; \mathbf{n} \in \mathcal{E},\ \mathcal{E} \in \mathcal{E}_\mathcal{Z},\ \mathbf{n} \notin \mathcal{N}_\mathcal{Z} \right\}.$$

We denote any element having at least one terminal node as one of its corners as a *terminal element*. Moreover, any edge connecting two terminal nodes will be referred to as a t*erminal edge*.

This step will result in a first approximation for $\Gamma^*(\Omega)$ by the edges connecting the nodes in $\mathcal{N}_\mathcal{T}$. Note that any edge connecting two terminal nodes is either horizontal, vertical, or having slope magnitude of ± 1.

In some cases, we may choose to modify, rather than deleting, an element in $\mathcal{E}_\mathcal{Z}$ in order to ensure the minimum angle requirement for the new elements generated to cover the gap between $\Gamma^*(\Omega)$ and the terminal edges. Figure 11 shows these cases.

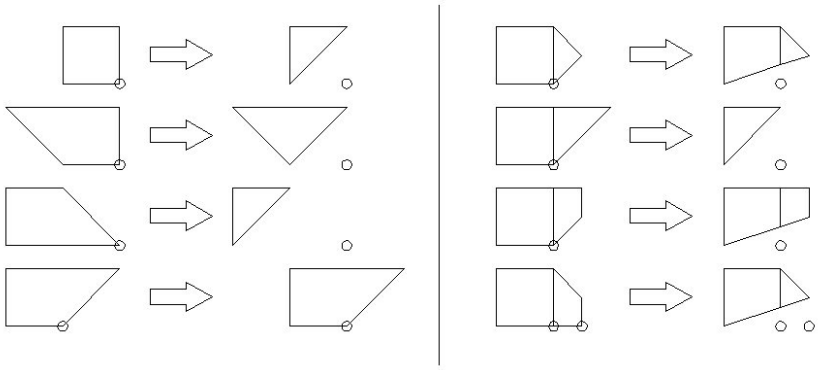
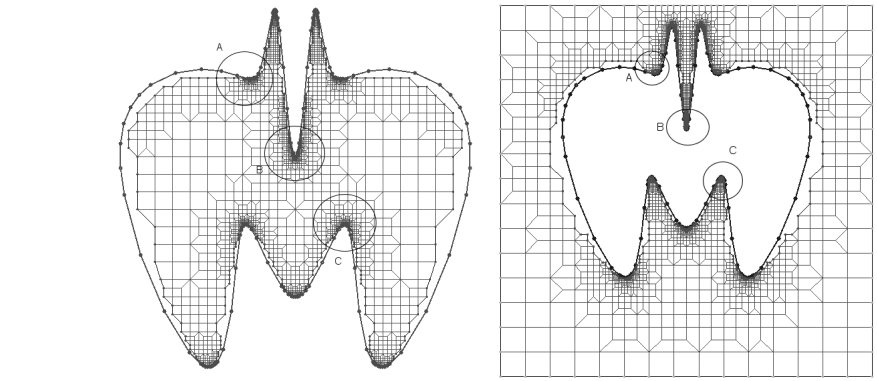


Figure 11. Modification of terminal elements, external nodes are marked with "o".

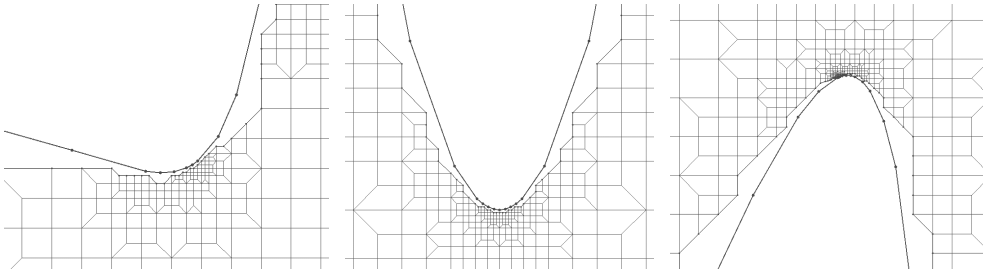
These modifications are to guarantee the following property almost everywhere. Let e_1 and e_2 be two terminal edges having a common terminal node. Then the external angle, α, between these two edges satisfies the following relation

$$135^\circ \leq \alpha \leq 225^\circ.$$

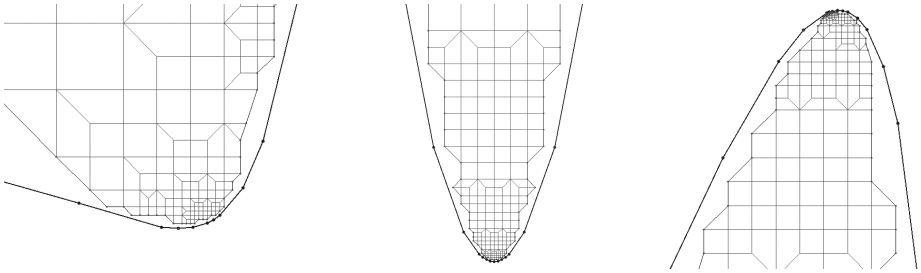
This gives us more freedom in controlling the quality of the elements to be generated within the gap between $\Gamma^*(\Omega)$ and the terminal edges. Figures 12 and 13 shows the output of this step for our two test domains $\mathcal{D}_1$ and $\mathcal{D}_2$.



(a) Internal domain (b) External domain

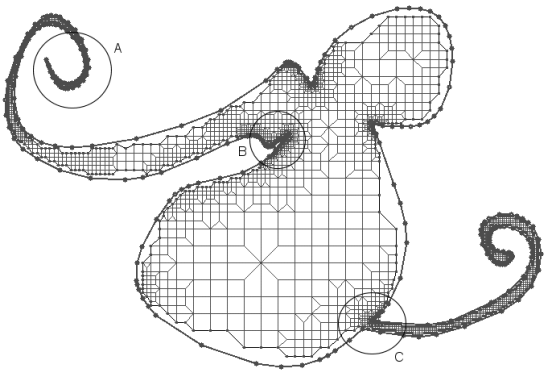


(c) Zoom A (d) Zoom B (e) Zoom C

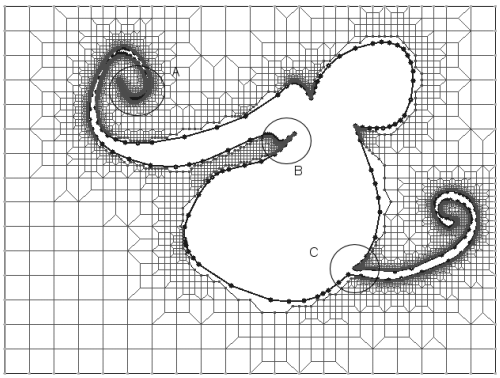


(f) Zoom A (g) Zoom B (h) Zoom C

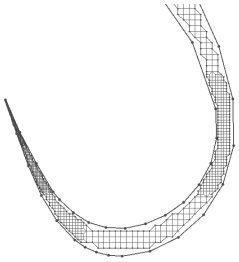
Figure 12. Creation of a buffer zone for internal and external domains for $\mathcal{D}_1$.



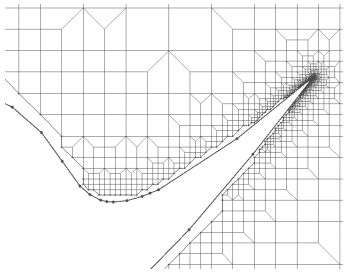
(a) Internal domain



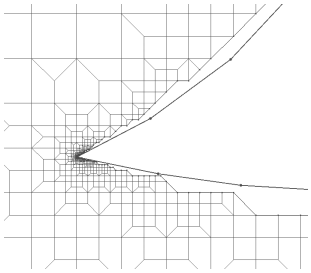
(b) External domain



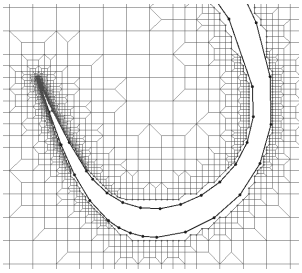
(c) Zoom A



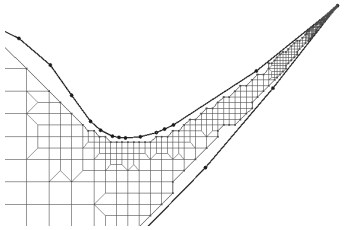
(d) Zoom B



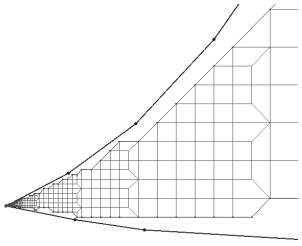
(e) Zoom C



(f) Zoom A



(g) Zoom B

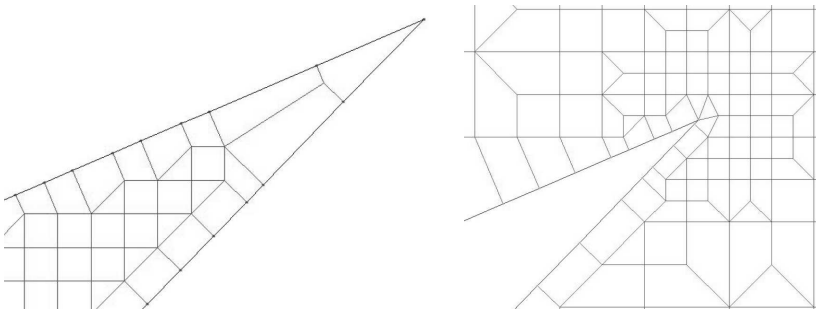


(h) Zoom C

Figure 13. Creation of a buffer zone for internal and external domains for $\mathcal{D}_2$.

4.2. Filling the buffer zone

The goal here is to modify the base grid again so that its boundaries represent Ω more accurately. This is accomplished by iterating over the terminal edges and constructing an element per edge by the orthogonal projection of the terminal points of that edge. If the orthogonal projection fails, we project the terminal node to the nearest singular point. The elements generated during this step are quadrilaterals with aspect ratio less than 2 or triangles if both ends of the terminal edge projects to a singular boundary point. In the case of sharp corners, a quadrilateral is constructed by projecting the nearest node to that corner twice on the two line segments forming that corner. This quadrilateral can be subdivided as much as necessary, depending on the desired edge length near that corner. Figure 14 shows an element constructed and refined near a sharp corner and a regular one. Note that a sharp corner is one with internal angle α such that $\alpha \leq 90°$.

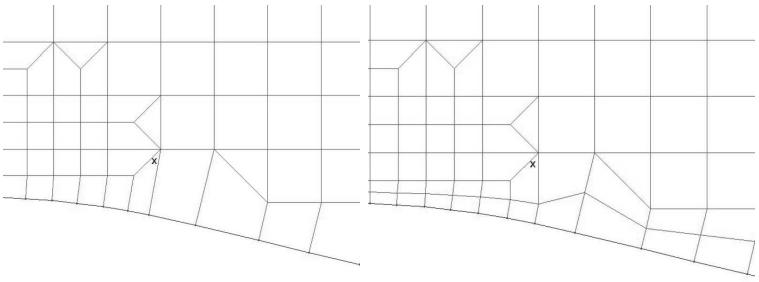


(a) Sharp corner (b) Regular corner

Figure 14. Elements near corners.

Up to this point, we have conserved the quality of most elements except some elements where angles are less than 45°. In order to fix that, all the edges of the new elements perpendicular to the boundaries of Ω are split and then some of these edges are allowed to rotate to ensure that the minimum angle is 45° for all elements. This step is illustrated in Figure 15. For simulation of turbulent flows, successive edge splitting can be done until we reach the required number of points in the viscous sublayer. Note that, as we perform more edge splitting, the boundary elements become more rectangular with all angles reaching 90°. Figures 16 and 17 show the final grids generated using the test domains we have used in the previous sections.

Next we use the shape of one of the Great Lakes to demonstrate the efficiency of our procedure. The boundaries of this domain consists of seven closed curves with no singular points. Since the boundaries have much detail and widely-varying curvature, this problem is often used in the grid-generation literature to test new algorithms. Also, some islands are very close to the boundaries of the lake that provide another opportunity to test the ability of our algorithm to capture and represent domains of narrow regions. Figure 18 shows the initial distribution of the boundary points for Lake Superior, the output of the spatial decomposition, and the final grid generated by our technique. Some further details of the grid generated using



(a) Angle less than 45° (b) Quality Enhanced

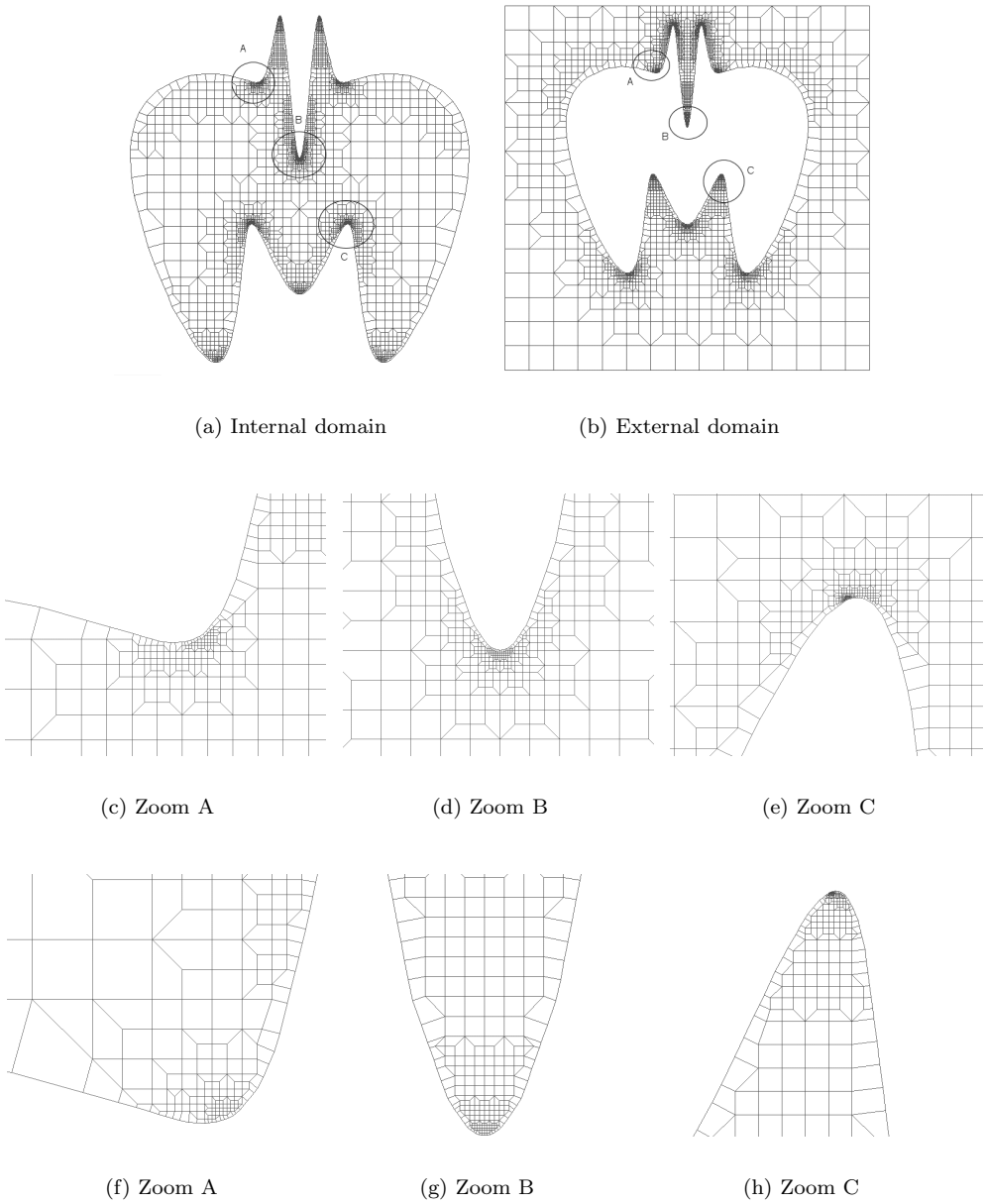
Figure 15. Optimizing the minimum element angle by edge splitting and rotation.

our technique for this domain are presented in Figure 19. For comparison, Figure 20 displays the grids produced by two other procedures: Ruppert's algorithm [26] and the quadtree-based algorithm by Bern, Eppstein, and Gilbert [27]. These results show that our technique has better adaptivity and produces grids with a larger minimum angle.

Finally, we represent a domain around a multi-element airfoil. The boundaries consist of three closed curves with curvature varying from zero to infinity. We will use the same domain in showing our treatment of the boundary layer region in the following section. The results for a domain to be used in laminar flow simulations are presented in Figure 21.

4.3. The boundary layer region

In simulation of viscous flows, the velocity flow variables have a high gradient near the boundaries when representing the solid surfaces with a no-slip condition. In order to capture the flow variables efficiently in these regions, successive one-dimensional refinement, i.e., the splitting of only the edges normal to these boundaries, is employed to rapidly reduce the length of these edges. Note that this one-dimensional refinement also acts as an element smoother. Applying such refinement to the elements close to the boundaries will rapidly increase the minimum angle of the boundary elements. This fact is demonstrated in Figure 23 where we started with the worst case, namely an element with a minimum angle of 45°. After only three levels of this one dimensional refinement, the minimum angle increased to 83° which indicates that the element is becoming more rectangular. The number of these one-dimensional refinements depends on the viscous flow length scales described by the Reynolds number and on the requirements of the employed turbulence model. For example, the Wilcox k–ω model [28, 29] requires that a few points are located within the viscous sublayer. We can achieve this requirement usually with 6 to 8 levels of one-dimensional refinements. Note that the aspect ratio of the boundary elements increases with more levels, however. We ran six levels of refinements for the grid presented in Figure 21. The details of the boundary layer in some regions are shown in Figure 24.

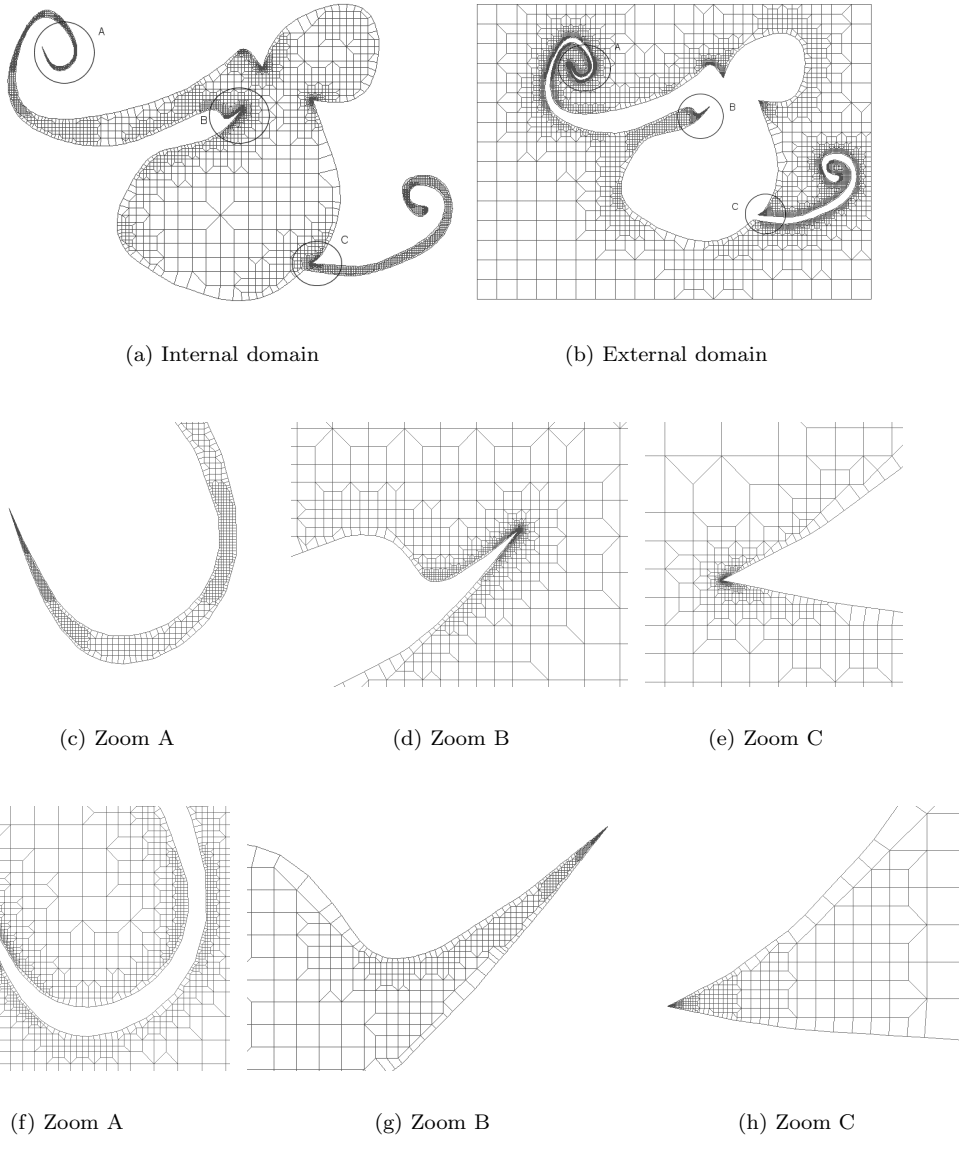


(a) Internal domain (b) External domain

(c) Zoom A (d) Zoom B (e) Zoom C

(f) Zoom A (g) Zoom B (h) Zoom C

Figure 16. Internal and external domains for $\mathcal{D}_1$.

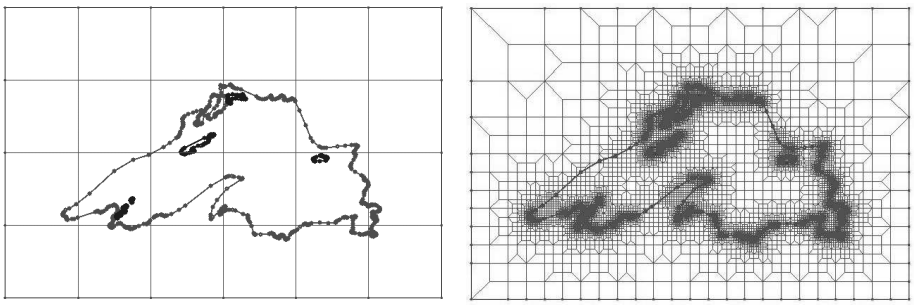


(a) Internal domain (b) External domain

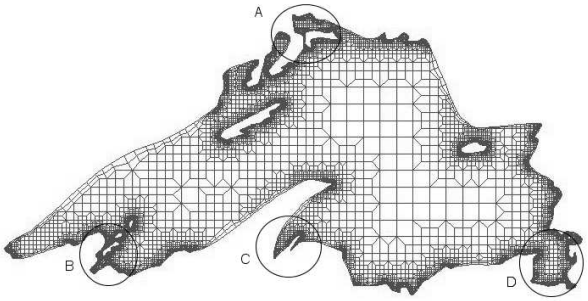
(c) Zoom A (d) Zoom B (e) Zoom C

(f) Zoom A (g) Zoom B (h) Zoom C

Figure 17. Internal and external domains for $\mathcal{D}_2$.



(a) Distribution of boundary points (b) Spatial decomposition

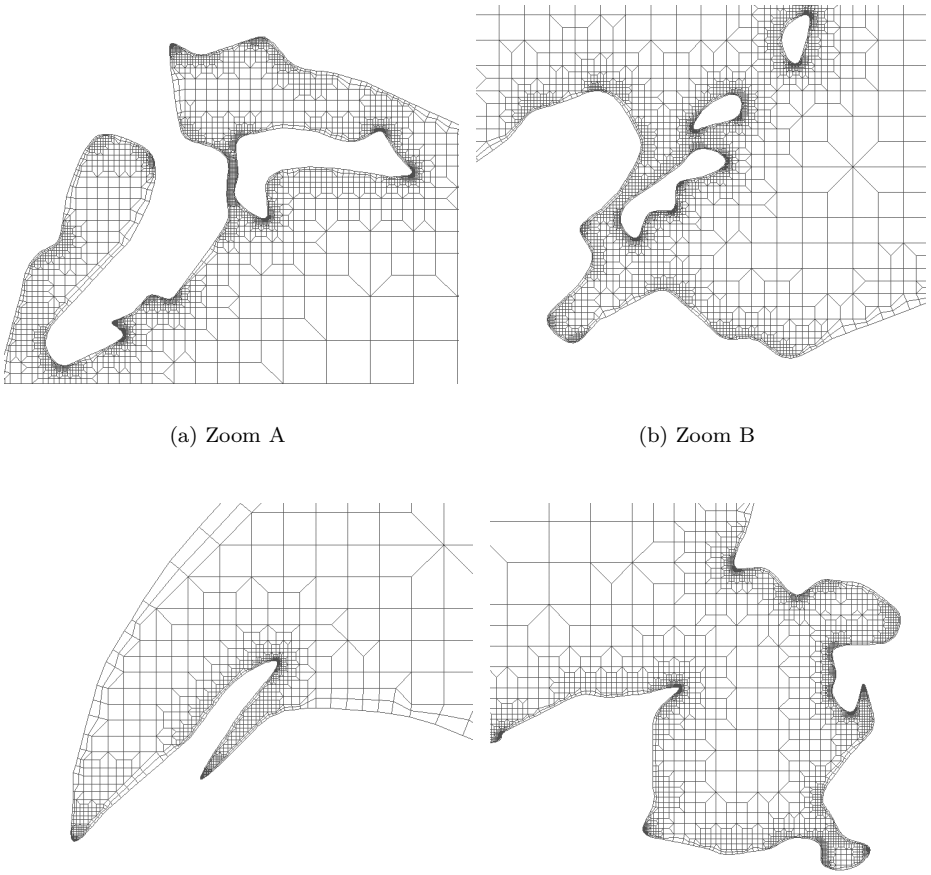


(c) Our algorithm: minimum angle 45°

Figure 18. Grid generated for Lake Superior by our technique.

5. SOLUTION-BASED ADAPTATION

In this section, we discuss a procedure for solution-based spatial adaptation. We start this step with the base grid, $\mathcal{G}_b$, and the grid, $\mathcal{G}_c$, obtained from the last adaptive regridding. If this is the first adaptive regridding, we simply set $\mathcal{G}_c = \mathcal{G}_b$. Using the simulation output, we generate some function, f, that controls the grid size. For example, f could be an error estimation of a finite-element solver, or it could be the gradient of one of the solution variables since error is typically proportional to the gradient. Note that the base grid $\mathcal{G}_b$ is coarse everywhere except near the boundaries. Refining this $\mathcal{G}_b$ during different solution times in the simulation will have the effect that the grid elements are sometimes refined and other times agglomerated, although we are always refining $\mathcal{G}_b$. The problem with depending on the base grid only is that we may lose solution accuracy due to neglecting solution information at the fine levels. To overcome this problem, we use two grids in this algorithm. The base grid is used for regridding and the one obtained from the last adaptive regridding is used for passing the variables to the new grid. The details of the algorithm are as follows.



(a) Zoom A (b) Zoom B

(c) Zoom C (d) Zoom D

Figure 19. Details of the grid generated for Lake Superior by our technique.

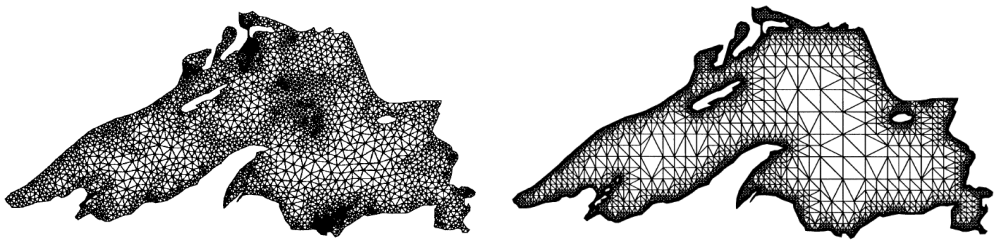
Algorithm 3. (Unsteady grid adaptation)
Input: *A base grid $\mathcal{G}_b$, a grid $\mathcal{G}_c$, solution variables f defined at every node of $\mathcal{G}_c$ from the last time step, a user-specified integer $n_R > 0$, and a vector $r_L \in \mathcal{R}^{n_R}$ of user-specified thresholds for the refinement levels.*

1. *Interpolate the solution values f from grid $\mathcal{G}_c$ to $\mathcal{G}_b$.*

2. *For $i = 1, 2, \ldots, n_R$, do:*

- *Merge transition elements for that refinement level with its corresponding fine region.*
- *Refine any elements with a corner node j if the condition*

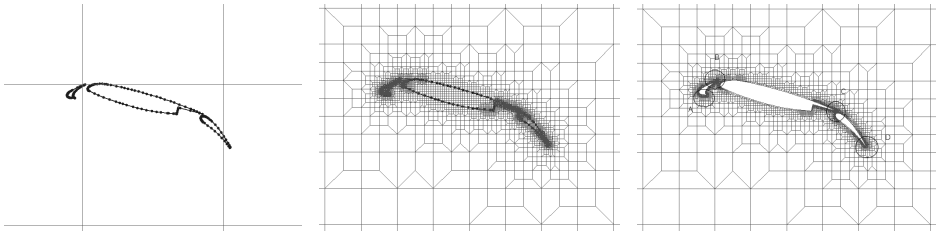
$$|f_j| > r_L(i)$$



(a) Ruppert algorithm: minimum angle 34.2°

(b) Algorithm by Bern et al.: minimum angle 18°

Figure 20. Grid generated for Lake Superior by other algorithms.

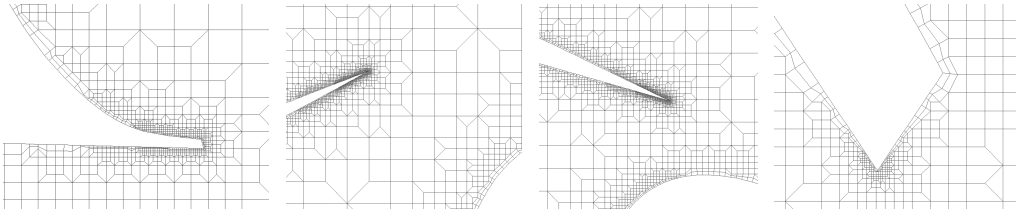


(a) Distribution of boundary points

(b) Spatial decomposition

(c) Final grid

Figure 21. Three-part airfoil grid produced by our technique.



(a) Zoom A

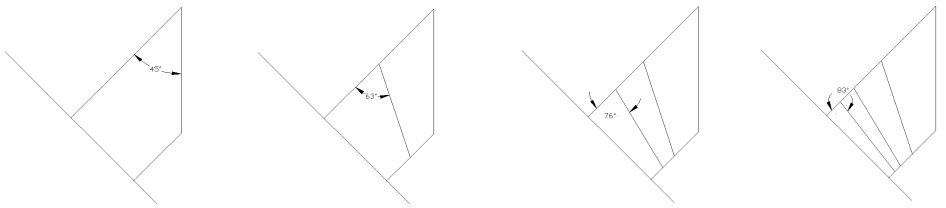
(b) Zoom B

(c) Zoom C

(d) Zoom D

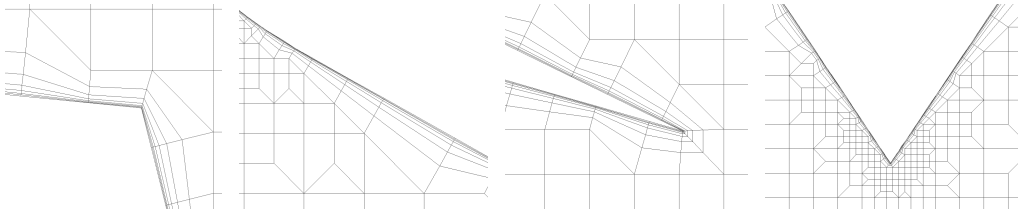
Figure 22. Details of three-part airfoil grid produced by our technique.

is satisfied.



(a) Level 0 (45°) (b) Level 1 (63°) (c) Level 2 (76°) (d) Level 3 (83°)

Figure 23. Effects of the one-dimensional refinement on the minimum angle of a boundary element.



(a) (b) (c) (d)

Figure 24. Details of the boundary-layer region around a multi-element airfoil.

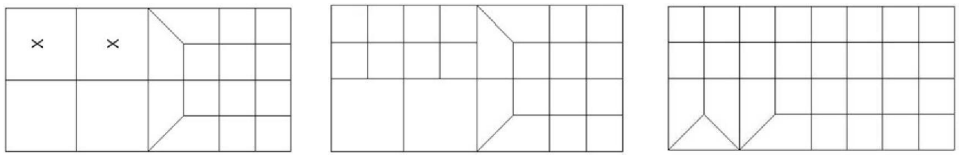
- *Interpolate the values of the solution variables for the new point from grid $\mathcal{G}_c$.*

3. *Set $\mathcal{G}_c$ to be the new refined grid.*

Output: *New refined grid $\mathcal{G}_c$.*

Merging transition elements near the boundaries is done to avoid the refinement of any edge in these elements. Figure 25 demonstrates this problem and how to fix it. Here, if we decide to refine a square element in the coarse region (marked with an "x"), a grid quality problem will result as shown in Figure 25(b). The technique used to fix this problem is to merge all the transition elements for that level. In other words, transform each block of transition elements into 4 square elements, as shown in Figure 25(c).

In order to show the efficiency of this algorithm, we ran an unsteady flow simulation over two vertical cylinders at Reynolds number $Re = 200$. Figure 26 shows the interaction of the vortex shedding around each cylinder and how the vortices impact each other at the centerline preserving the symmetry of the flow. Vorticity contours are overlaid onto the computational grid to show the efficiency of the adaptive regridding. In this case, we chose f to be $\nabla\omega$ where ω is the vorticity generated in the domain.



(a) (b) (c)

Figure 25. Merging transition elements.

6. MULTIGRID LEVELS

The application of multigrid solvers was our main motivation for the development of the proposed grid generation technique. For this particular application, we need a sequence of grid levels that can be implemented easily in multigrid solvers, and this is exactly what we have achieved here. Since we are already using a grid-embedding technique, we can easily store all the grid levels that are generated during the refinement procedure. However, we need to redefine the near-boundary elements at each level and delete all the elements with deleted nodes by the optimization step. This modification can be summarized in the following steps for each grid level. Note that all the near-boundary elements that need to be modified are squares.

- Delete any element with more than two deleted corners.
- Loop over those elements that have only two corners in the base grid $\mathcal{G}_b$: replace the other two deleted nodes with the closest two boundary nodes in $\mathcal{G}_b$.
- Loop over those elements that have only three corners in the base grid $\mathcal{G}_b$:
  - Identify the two closest corners p_1, p_2 of such an element to the boundaries and denote the third one to be p_3.
  - Identify the closest boundary nodes q_1 and q_2 in the base grid $\mathcal{G}_b$ to p_1 and p_2.
  - Create a triangular element with the nodes p_1, p_2, p_3.
  - Create a quadrilateral element with the nodes p_1, p_2, q_2, q_1.

Figure 27 displays the output of this algorithm using a grid that was generated during an unsteady simulation over two vertical cylinders ar $Re = 200$. The results show that we have all the levels required for the multigrid method. Note that many of the levels share the same elements. When applying multigrid techniques to such grids, we need only to smooth the error at the corners of the non-matching elements to save time.

7. CONCLUSION

We have introduced a technique suitable to create adaptive grids for arbitrarily-shaped two-dimensional domains. The minimum angle of 45° is achieved in all elements, except at

singular boundary points. Our approach provides more flexibility to approximate regions near sharp corners. Several application examples of two-dimensional grids have been provided to illustrate the main features and the efficiency of the proposed approach. Moreover, the proposed technique is capable of handling problems with complex geometry, such as free-surface problems with moderate distortion, as well as problems with multiple bodies. This work can be considered as a preliminary stage toward a comprehensive tree-based decomposition method, prior to the extension of the algorithm to three dimensions.

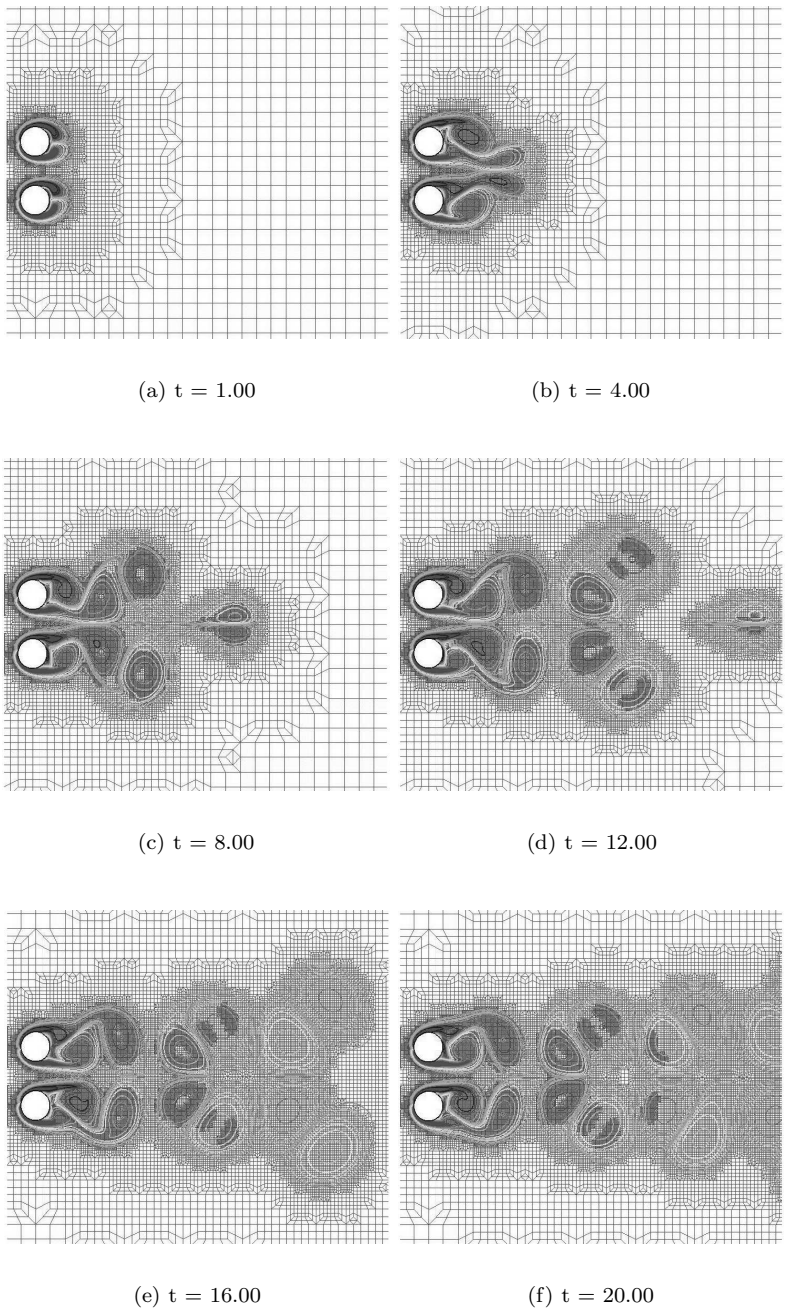
ACKNOWLEDGEMENTS

The authors would like to thank Professor Dimitri Mavriplis, University of Wyoming, for providing the three-part airfoil geometry.

REFERENCES

1. Thompson JF, Soni BK, Weatherill, NP (eds). *Handbook of Grid Generation*. CRC Press, 1999.
2. Ito Y, Nakahashi K, Surface triangulation for polygonal models based on CAD data. *International Journal for Numerical Methods in Fluids* 2002; **39**:75–96.
3. Frey PJ. Generation and adaptation of computational surface meshes from discrete anatomical data. *International Journal for Numerical Methods in Engineering* 2004; **60**:1047–1049.
4. Löhner R. Regridding surface triangulations. *Journal of Computational Physics* 1996; **126**:1–10.
5. De Cougny HL. Refinement and coarsening of surface meshes. *Engineering with Computers* 1998; **14**:214–222.
6. Wang D, Hassan O, Morgan K, Weatherill N. Enhanced remeshing from STL files with applications to surface grid generation. *Communications in Numerical Methods in Engineering* 2007; **23**:227–239.
7. Briggs WL, Henson VE, McCormick SF. *A Multigrid Tutorial* (2nd edn). SIAM, 2000.
8. Connell SD, Braaten DG. A 3D unstructured adaptive multigrid scheme for the Euler equations. *AIAA Journal* 1994; **32**(8):1626–1632.
9. Perez E. Finite element and multigrid solution of the two dimensional Euler equations on a non structured mesh. *INRIA Research Report* No. 442; INRIA, Paris, 1985.
10. Mavriplis DJ. Three dimensional unstructured multigrid for the Euler equations. In *Proceedings of the AIAA 10th Computational Fluid Dynamics Conference*. Paper **91-1549**, 1991.
11. Mavriplis DJ. Three-dimensional multigrid Reynolds-averaged Navier-Stokes solver for unstructured meshes. *AIAA Journal* 1995; **33**(3):445–453.
12. Lallemand MH, Steve H, Dervieux A. Unstructured multigridding by volume agglomeration: current status. *Computers & Fluids* 1992; **21**:397–433.
13. Jones JE, Vassilevski PS. AMGE based on element agglomeration. *SIAM Journal on Scientific Computing* 2001; **23**(1):109–133.
14. Koobus B, Lallemand MH, Dervieux A. Unstructured volume-agglomeration MG: solution of the Poisson equation. *International Journal for Numerical Methods in Fluids* 1994; **18**(1):27–42.
15. Mavriplis DJ, Venkatakrishnan V. Agglomeration multigrid for two-dimensional viscous flows. *Computers & Fluids* 1995; **24**5:553–570.
16. Mavriplis DJ. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. *Journal of Computational Physics* 1998; **145**(1):141–165.
17. Flaherty JE, Paslow PJ, Sheppard MS, Vasilakis JD (eds). *Adaptive Methods for Partial Differential Equations*. SIAM, 1989.
18. Babuska I, Zienkiewicz OC, Gago J, de A. Oliveira ER (eds). *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*. Wiley, 1986.
19. Zienkiewicz OC, Gago JP, Kelly DW. The hierarchical concepts in finite element analysis. *Computers & Structures* 1983; **16**:53–65.
20. Davis RL, Dannenhoffer III JF. Decomposition and parallelization strategies for adaptive grid-embedding techniques. *International Journal of Computational Fluid Dynamics* 1993; **1**:79–93.
21. Frey PJ, Marechal L. Fast adaptive quadtree mesh generation. In *Proceedings of the 7th International Meshing Roundtable* 1998; 211–224.

22. Persson PO, Strang G. A simple mesh generator in MATLAB. *SIAM Review* 2004; **46**(2):329–345.
23. Shewchuk JR. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications* 2002; **22**(1):21–74.
24. Finkel RA, Bentley JL. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica* 1974; **4**(1):1–9.
25. de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. *Computational Geometry* (2nd edn). Springer-Verlag, 2000.
26. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms* 1995; **18**(3):548–585.
27. Bern M, Eppstein D, Gilbert J. Provably good mesh generation. *Journal of Computer and System Sciences* 1994; **48**(3):384–409.
28. Wilcox DC. *Turbulence Modeling for CFD* (3rd edn). DCW Industries, 2006.
29. Wilcox DC. Formulation of the k–ω turbulence model revisited. In *Proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit*. Paper **2007-1408**, 2007.

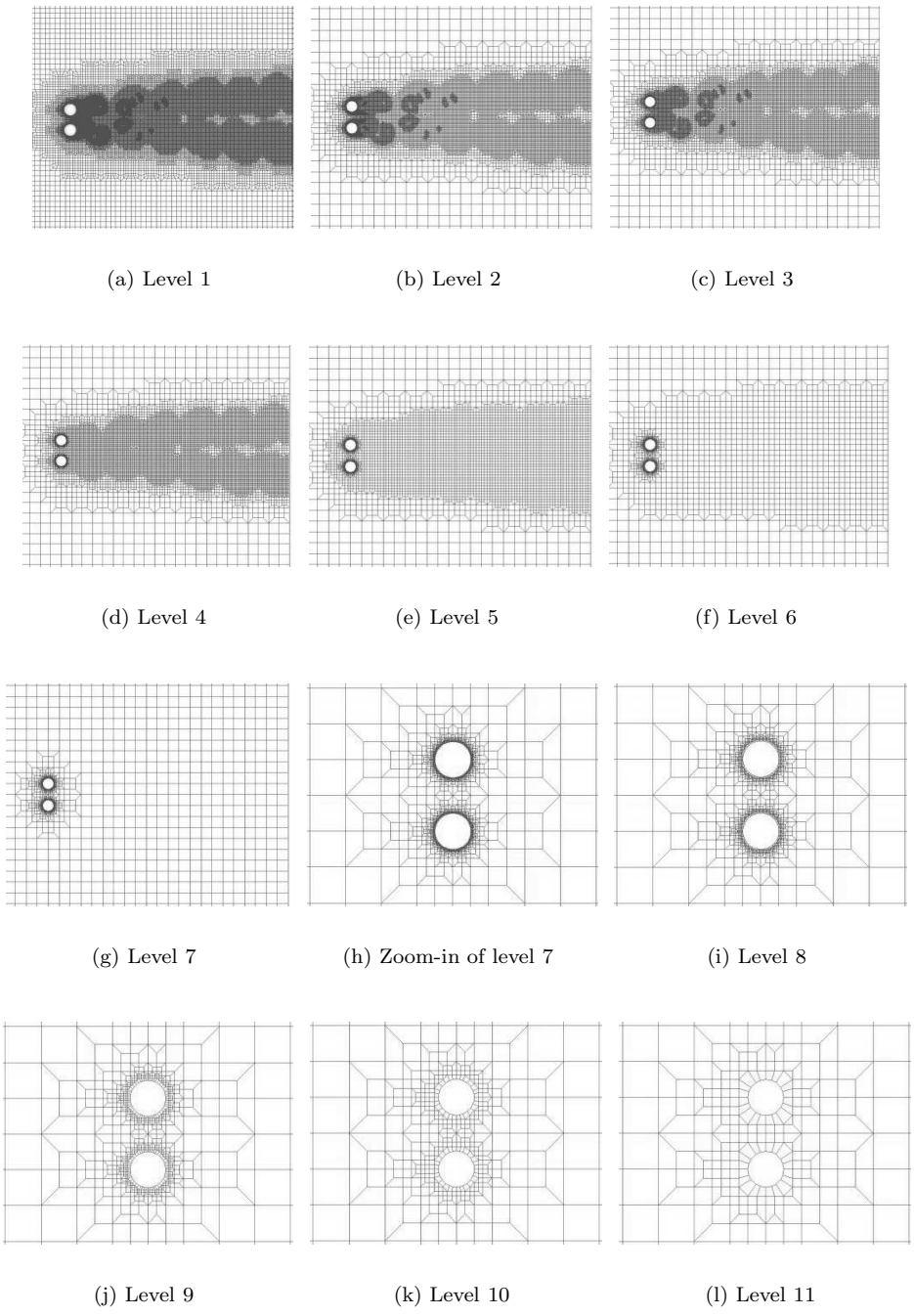


(a) t = 1.00 (b) t = 4.00

(c) t = 8.00 (d) t = 12.00

(e) t = 16.00 (f) t = 20.00

Figure 26. Evolution of grid and vorticity contours for flow over two vertical cylinders at $Re = 200$.



(a) Level 1 (b) Level 2 (c) Level 3

(d) Level 4 (e) Level 5 (f) Level 6

(g) Level 7 (h) Zoom-in of level 7 (i) Level 8

(j) Level 9 (k) Level 10 (l) Level 11

Figure 27. Grid levels for a domain with two vertical cylinders.