AN ADAPTIVE HYBRID MESH GENERATION METHOD FOR COMPLEX GEOMETRIES

By

Cameron Thomas Druyor JR

Approved:

Steve Karman Jr.
Professor of Computational Engineering
(Director of Thesis)

Daniel Hyams
Associate Professor of Computational Engineering
(Committee Member)

William Sutton
Dean of College of Engineering and Computer Science

Vincent C. Betro
Research Assistant Professor of Computational Engineering
(Committee Member)

Jerald Ainsworth
Dean of the Graduate School

AN ADAPTIVE HYBRID MESH GENERATION METHOD FOR COMPLEX GEOMETRIES

By

Cameron Thomas Druyor JR

A Thesis
Submitted to the faculty of
The University of Tennessee, Chattanooga
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computational Engineering

The University of Tennessee, Chattanooga
Chattanooga, Tennessee

August 2011

ABSTRACT

An adaptive hybrid mesh generation method is described to automatically provide spatial discretizations suitable for computational fluid dynamics or other 2D solver applications. This method employs a hierarchical grid generation technique to create a background mesh, an extrusion-type method for inserting boundary layers, and an unstructured triangulation to stitch between the boundary layers and background mesh. This method provides appropriate mesh resolution based on geometry segments from a file, and has the capability of adapting the background mesh based on a spacing field generated from solution data or some other arbitrary source. By combining multiple approaches to the grid generation process, this method seeks to benefit from the strengths of each, while avoiding the weaknesses of each. Possible future work to increase the robustness of the method is also discussed.

DEDICATION

I would like to dedicate this work to my family, without whose support, I would not be where I am today.

ACKNOWLEDGEMENTS

I would like to thank Dr. Karman for his guidance and unlimited patience thoughout the process, and Dr. Betro for paving the way with his work in 3D. I would also like to thank Matthew O'Connell for letting me bounce ideas off him for two years, and David Collao for creating the spacing library which will be an indispensable tool.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Motivation

Generating a grid for a complex geometry that is to be used in a computational fluid dynamics, structural dynamics, electromagnetic, or other type of analytic solver represents one of the most time consuming, man-power intensive obstacles in the way of getting a solution. While many different methods and technologies have been developed over the past few decades, an automatic mesh generation method that is universally applicable has yet to be developed. In the absence of a universal method, existing and new techniques that excel at different aspects of the mesh generation process must be combined to form more robust hybrid methods.

Mesh Classifications

Structured meshes consist of one or more blocks of cells, each of which contains elements with an implied connectivity. This means that cells are arranged in a regular fasion such that the index of a given cell is algabraically related to the indices of its neighbors, which allows for efficient storage in single or multidimensional arrays. The process for generating structured meshes is typically not fully automated, and requires the input from an experienced practitioner. While this makes the process time and work intensive for the user, it can result in high quality meshes that accurately capture the geometry and have qualities that are appropriate for the intended application. Commercial packages such as

Pointwise [1] are widely used to produce industrial multi-block structured grids (Note that they also have the ability to create unstructured grids).

Unstructured meshes have no implied connectivity, and as such, they require the creation of additional maps to describe the relationships between elements. These maps can increase the memory requirements greatly compared to a structured mesh with an equivalent element count, however the lack of implied structure permits the use of various element types including triangles, quadrilaterals, and polygons in 2D, and tetrahedra, pyramids, prisms, hexahedra, and higher order polyhedra in 3D.

Mesh Generation Techniques

Cartesian Methods

Cartesian grid methods have been a popular choice for unstructured mesh generation over the past two decades because they are efficient and produce high quality, Cartesian aligned elements. The work of Karman [2], Pember [3], Landsberg [4], and others demonstrated the versatility of automated Cartesian methods which used a cut-cell approach to create geometry conforming meshes in both two and three dimensions. Cutting cells that violate boundaries, however, introduced accuracy and stability concerns in cases where arbitrarily small cut-cells appear [5]. This situation is avoided in hybrid methods by simply removing any cell that would normally be cut, and using some alternate method to fill the resulting gap. The inherent directionality of Cartesian methods tends to make them inflexible in terms of adapting to non-Cartesian aligned spacings. This drawback caused Coirier to introduce a method that used anisotropic refinement that allowed cells to be cut only along one axis [6]. Allowing this refinement to be arbitrary, however, yielded meshes unfit for use with a solver. Later Aftmosis et al [5] placed limitations on the anisotropic nature of the elements in the mesh, with improved results. Refining in this anisotropic fashion can result in dramatic

reductions in the number of cells that are required to accurately resolve complex geometries as shown by Domel and Karman [7]. Their method, SPLITFLOW, used an Omni-tree that allowed cells to be split in any combination of the three Cartesian directions with a limited aspect ratio.

Since Cartesian methods are capable of easily controlling the spacing in any particular region of the mesh, they are perfectly suited for creating adapted meshes that have refinement based on solution data. Domel [7], Karman [8], Wang [9], and others have demonstrated that Cartesian methods can indeed produce highly accurate solution adapted meshes.

Extrusion Methods

Extrusion methods begin at the surface of the geometry, and march outward, generating points along surface normal vectors and creating quadrilateral elements one layer at a time. This kind of technique allows for the generation of high aspect ratio quadrilaterals near the surface, which is desireable in computational fluid dynamics when viscous terms are included in the flow solver. The proposed method uses a rudimentary extrusion algorithm, the implementation of which will be covered in later chapters. It is important to note that others including Thompson [10] have developed strategies for improving the quality of extruded meshes, but this work has not yet been incorporated into the proposed method. Wang [9], Dawes [11], and others have also shown that level sets can be used as input when creating viscous layers with extrusion type methods, but this is also left unexplored in the course of this research.

Triangulation Methods

Unstructured triangular meshes can be generated either by generating a point cloud and triangulating between those points or by generating the triangular elements and points

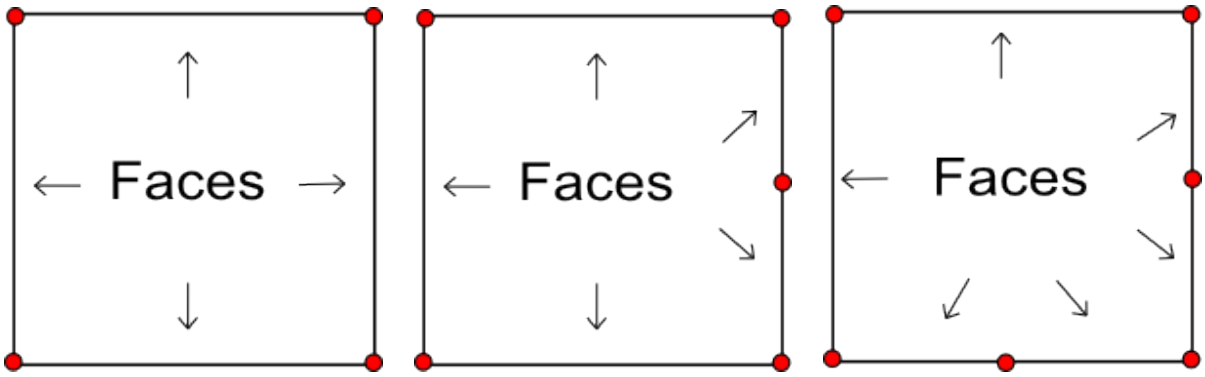
simultaneously with a method such as the Advancing Front Method [12]. Methods of this type are mature and robust technologies that are easily capable of handling complex geometries. They do not afford much control over spacing in the inviscid regions, however, and adaptation must typically be done after the generation process, even if the desired spacing is known. There is an exception to this drawback for point creation methods that organize the points in a quad-tree structure during the generation as done by Liu [13], in which case, the method has some knowledge of point locality and can create more points in the regions where the spacing is known to be smaller.

The proposed method simply uses a Delaunay triangulation method to stitch boundaries together, without inserting any new points into the mesh, so triangulation methods will not be discussed in great detail in the course of this research.

Hybrid Mesh Generation

Hybrid grid methods seek to retain the efficiency and robust nature of Cartesian methods while avoiding their inherent difficulties when dealing with boundaries. This is done by allowing a more flexible method to handle the regions close to the boundaries, but using a Cartesian method to control the spacing elsewhere in the mesh. When viscous layers are desired, a type of extrusion method can be used to create the mesh in the viscous region and the Cartesian method can generate a mesh for the remaining inviscid regions. This solves the boundary layer problem, but the Cartesian method still has to capture the outer layer of the viscous mesh. In order to achieve this without utilizing a cut-cell method, a third meshing strategy is introduced: triangulization, or tetrahedralization in 3D. This concept can be seen in the work of Lohner et al [14] in 2D and Betro [15] in 3D. The proposed method is primarily based on the results of Lohner [14], but with the intention of adding adaptive capabilities to the background Cartesian method.

It is important to note that the background mesh will consist of quadrilateral, pentagon, and hexagon elements, so either the solver must be prepared to accept the higher order elements, or they must be decomposed into triangles and quadrilaterals. Figure 1.1 shows the different scenarios that cause higher order elements in the background mesh. Note that the "pentagons and hexagons" are actually quadrilateral elements, but they have mid-edge nodes.



(a) Quadrilateral (b) Pentagon (c) Hexagon

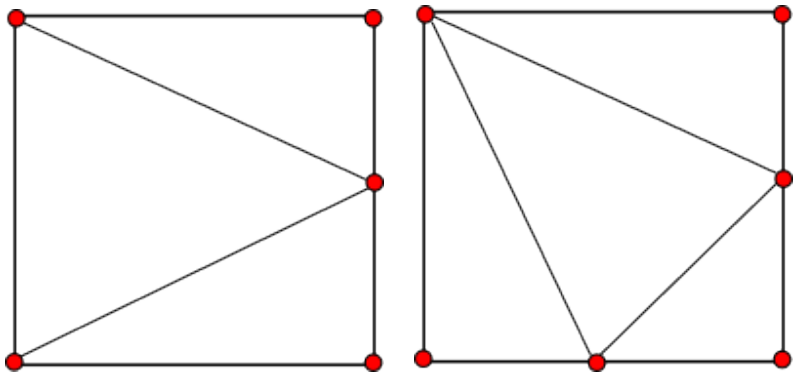
Figure 1.1 Right angle element types

Figure 1.2 shows how the higher order elements can be decomposed into triangles. Note that the triangular elements will always have this regular form, so the mesh quality remains high.

Chapter Summaries

Chapter 2

A brief description of the objectives of this research and the work that was done is presented.



(a) Pentagon (b) Hexagon

Figure 1.2 Polygon decomposition

Chapter 3

The basic input requirements of the method are discussed, and the results of different geometry spacings are compared.

Chapter 4

The reader is introduced to the data structures used in the background mesh generation process, and compares the split tree approach to other available approaches including the quad-tree and omni-tree. Then the mechanics of cell splitting are presented with visual examples. The idea of the Riemannian metric tensor is also briefly discussed, but the mathematics of its application can be found in the appendix.

Chapter 5

Viscous layer creation is discussed in detail. The mechanics of node prediction, front collision detection, node and cell generation, and removal of invalid elements are each presented alongside illustrations where appropriate.

Chapter 6

The critical aspect of joining the viscous mesh and the inviscid background mesh is addressed. The mechanics of cell removal and the triangulation method used are discussed.

Chapter 7

Experimental results are presented, demonstrating both the robustness of the method and its ability to produce meshes that are adapted based upon solution data.

Chapter 8

Conclusions concerning the advantages of the proposed method are discussed, along with future work that could improve the quality of the end result mesh and methods for avoiding the pitfalls of the proposed method are presented.

CHAPTER 2

OBJECTIVES

The goal of this research is to create a 2D hybrid mesh generation method that utilizes several techniques to create a single contiguous mesh for use in simulation, with a particular focus on Computational Fluid Dynamics applications. The proposed method differs from previous work such as that of Lohner [14] in two main aspects.

First, it has the capability to control spacing in the inviscid regions by adapting to a tensor field, which can be generated from solution data using a tool such as the spacing library developed by Collao [16]. The proposed method also provides utilities for creating tensor fields from other sources including existing meshes and spacing boxes, which are discussed and demonstrated in Chapter 3.

The proposed method also differs from the work of Lohner [14] in its use of a split-tree in place of a quad-tree, which allows for the creation of non-unit aspect ratio Cartesian cells. This is beneficial when the spacing field of a region is at least partially aligned with a Cardinal direction because the region can be filled with a smaller number of elements compared to using unit aspect ratio elements.

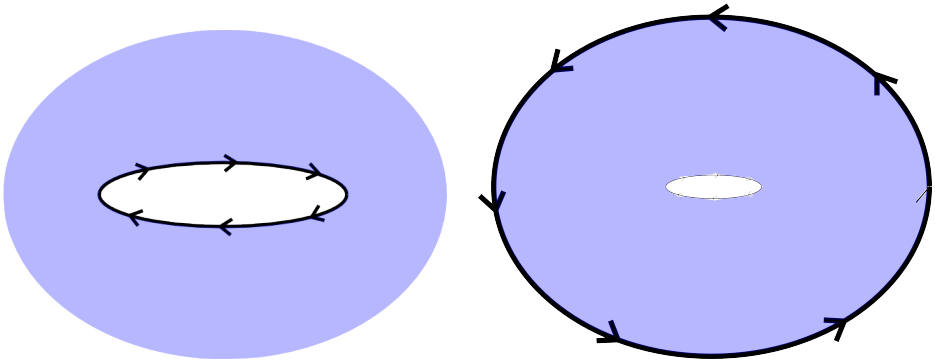
The following code was written for this research: a split-tree based Cartesian method for generating the background mesh, an extrusion method for generating meshes in viscous regions, a wrapper to incorporate a canned Delauney triangulation method, and a framework for creating and managing Reimannian metric tensors from a variety of sources.

CHAPTER 3

GEOMETRY PROCESSING

Requirements

The proposed method has two requirements regarding the geometry a mesh can be generated. The first requirement is that the defining geometry segments form water tight, closed loops. Secondly, the segments must be oriented such that the computational domain is on the left hand side of the segment, and the area outside the computational domain is on the right hand side as shown in Figures 3.1(a) and 3.1(b).



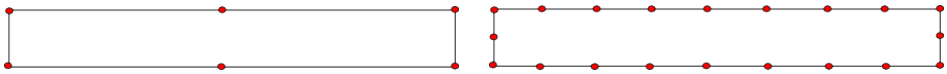
(a) The inner boundary has a clockwise winding

(b) The outer boundary has a counter-clockwise winding

Figure 3.1 The arrows show the direction of the geometry segments, and the blue tinted region is the computational domain.

The resolution of the mesh near the geometry is based upon the size of the boundary segments, so it is up to the user to provide geometry that has spacing that reflects the resolution desired. If the spacing on the boundaries is too coarse, the resulting mesh will

also be too coarse. While a small number of points may be enough to properly capture the features of the geometry as in Figure 3.2(a), it is unlikely that a mesh with edges of that length would be appropriate for obtaining a physical solution. Instead, the spacing on the geometry should be chosen such that it is representative of the desired spacing of the final mesh (Figure 3.2(b).

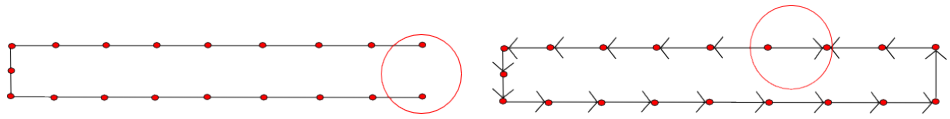


(a) Coarse spacing (b) Proper spacing

Figure 3.2 Boundary spacing on a flat plate

Geometry processing mechanics

Geometry is provided to the proposed method as a list of individual segments. The proposed method searches through the list for the minimum and maximum x and y values to determine the extent of the computational domain. This area, along with an additional buffer, will constitute the root cell for the background mesh generation process that is discussed in Chapter three. Once the extents of the computational domain have been calculated, the proposed method organizes the geometry segments into a set of closed bodies and tags the boundaries as either internal or external boundaries. If all of the boundaries are not closed, the method will abort because the input requirements were not met. A boundary is not closed if either a gap exists as shown in Figure 3.3(a), or if one or more edges is oriented in the wrong direction as shown in Figure 3.3(b).



(a) A gap in the geometry will cause it to be unclosed

(b) An edge that is oriented opposite the other edges in a given boundary will also cause that boundary to be unclosed

Figure 3.3 Unclosed boundaries

Riemannian Metric Tensors

From each geometry segment, a Riemannian metric tensor is created that defines the x and y spacing that is required in the vicinity of the segment. Equations 3.1, 3.2, and 3.3 are used to construct the Riemannian metric tensor M.

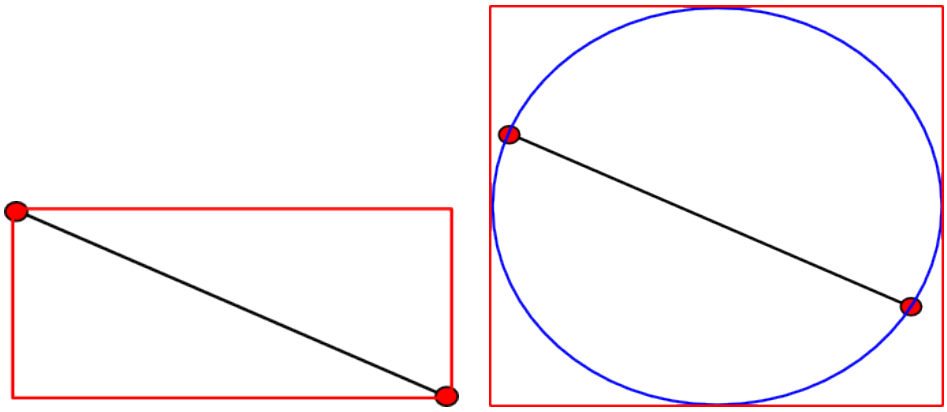
$$M = [R][\lambda][R]^{-1} \tag{3.1}$$

$$R = [\vec{e_1}\ \vec{e_2}] \tag{3.2}$$

$$\lambda = \begin{bmatrix} h_1^{-2} & 0 \\ 0 & h_2^{-2} \end{bmatrix} \tag{3.3}$$

The tensor is stored as a 2 x 2 matrix and it is given an extent over which it is effective. The proposed method has several options for calculating extent boxes. The first method simply defines an extent that is exactly sized to contain the segment (Figure 3.4(a)). The second method creates an extent box that circumscribes a circle whose center is at the centroid of the segment and whose radius is equal to half of the edge length as shown in Figure 3.4(b). This type of extent box is useful in cases where the geometry segment is

aligned or nearly aligned with either the x or y axis, which cause the extent box to have little or no area of effect. There are additional options for increasing the size of either type of extent box by a percentage or a scalar value. Once the tensor and its corresponding



(a) Pure extent box, defined only by the exact dimensions of the geometry segment

(b) Increased extent box, defined by a inscribing circle

Figure 3.4 Extent box creation

extent box are created, they are passed off to the split-tree to initiate the refinement process for the current geometry segment. It is important to note that only the tensor and the extent box, not the segment itself, are passed to the tree. This makes the refinement process flexible in that the tensors can be generated from any source including solution data, edges of another mesh, or even hand calculated. To determine whether a voxel needs to refine in either direction, a metric length is used. The metric length d for an edge v of a voxel in the mesh is calculated by Equation 3.4. If d is greater than 1.0, then the edge needs to be refined, and if d is less than 1.0, the edge is already an appropriate length.

$$d = \sqrt{\vec{v}^t M \vec{v}} \tag{3.4}$$

Since the spacing of the background mesh is based on metric tensors, the refinement process is decoupled from the geometry itself. This opens up a door for other sources to contribute spacing data, giving the user more control. First, a spacing box can be created that defines a scalar spacing for a region inside an arbitrary extent. Figure 3.5 demonstrates this capability.

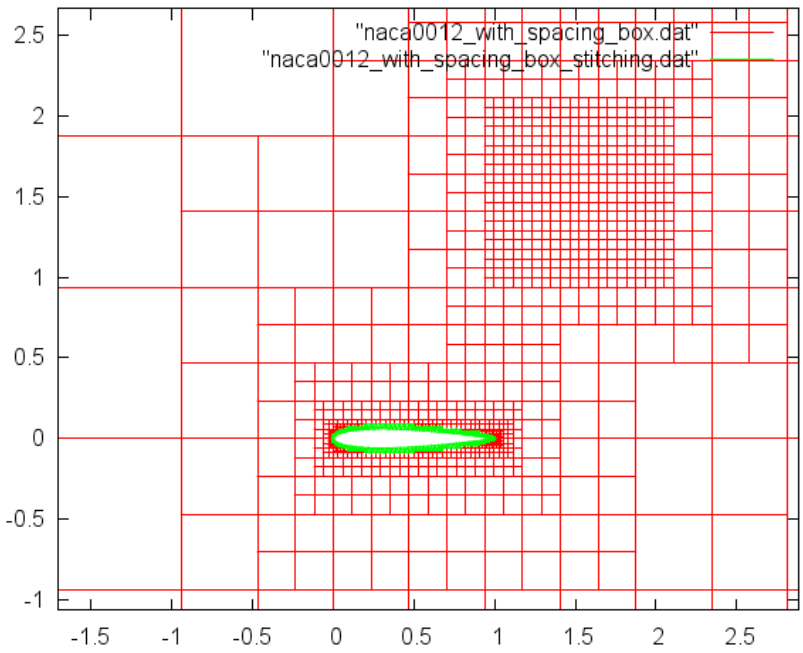


Figure 3.5 Adapting to a user defined spacing box

The proposed method has the ability to create a Cartesian background mesh whose spacing is based upon the spacing of another mesh as shown in Figure 3.6.

Note that the mesh from which the spacing data is obtained does not have to be a Cartesian mesh. In fact, the mesh used for this example, shown in Figure 3.7, consists entirely of triangles.

In Figures 3.8 and 3.9, a closer view is shown that demonstrates how the new Cartesian mesh mimics the gradation of the old triangular mesh.

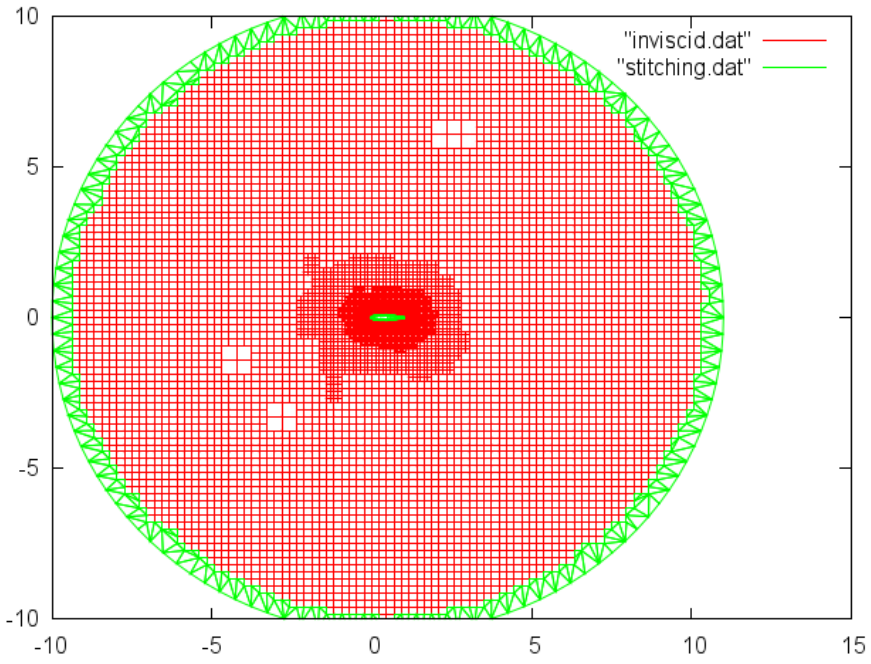


Figure 3.6 Adapting to the edges of an existing mesh

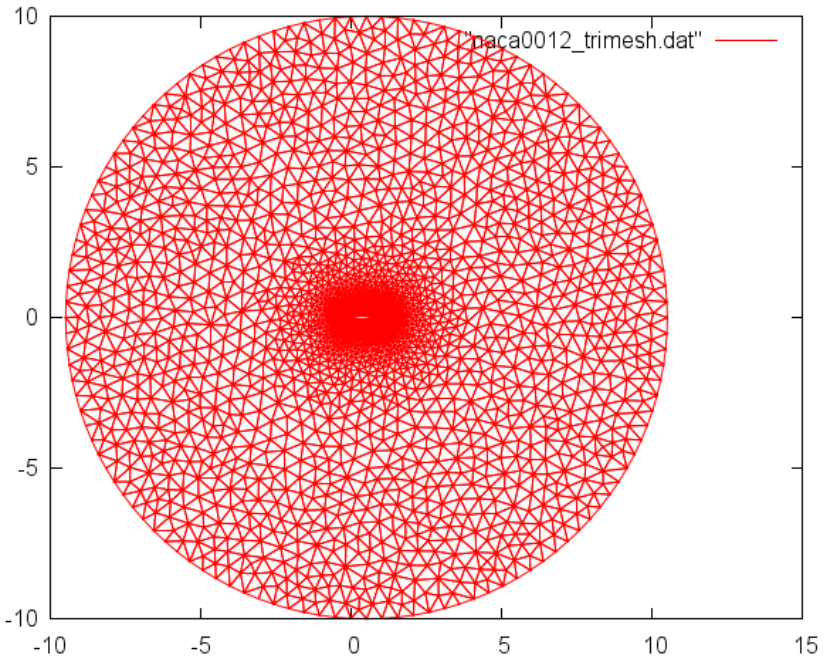


Figure 3.7 Original triangular mesh

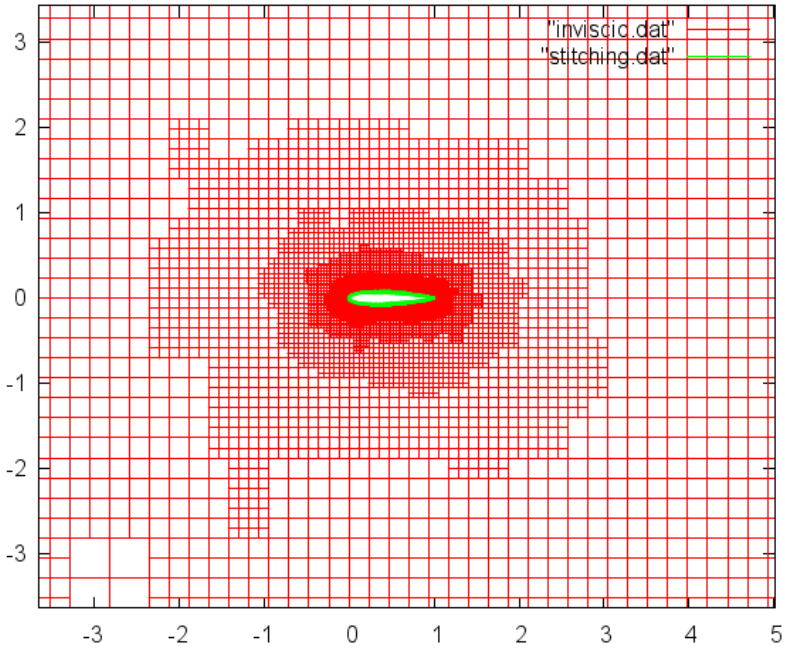


Figure 3.8 Adapting to the edges of an existing mesh zoomed

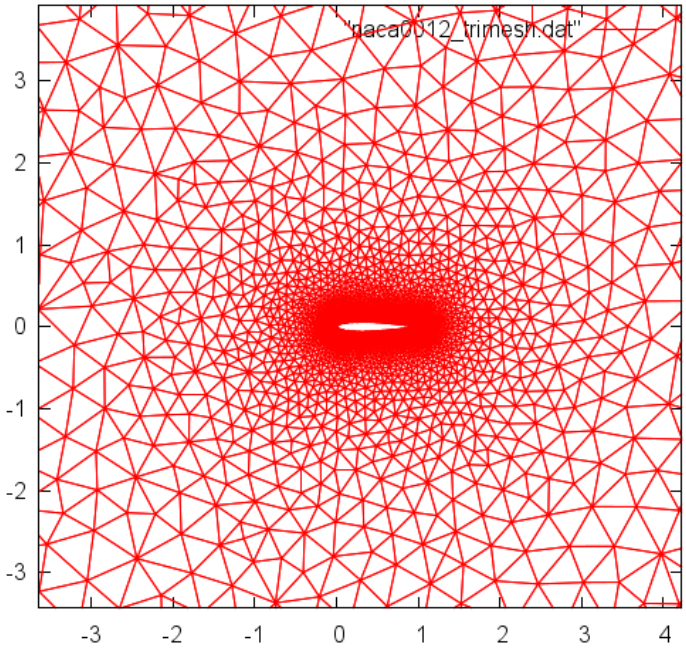


Figure 3.9 Original triangular mesh zoomed

If solution data is available, and a tensor field is created with a utility such as the work of Collao [16], the background mesh can be adapted to this as well. Figure 3.10 show an adapted mesh for a 30 degree supersonic ramp case.

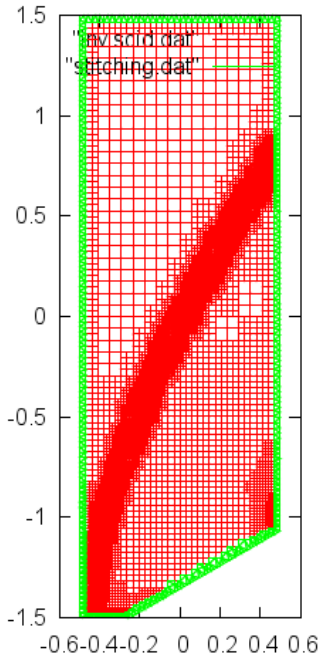


Figure 3.10 Spacing field adaptation, supersonic ramp

Figures 3.11, 3.12, and 3.13 show closer views of the adaption that has occured at the inflow and outlfow boundaries and part of the shock.

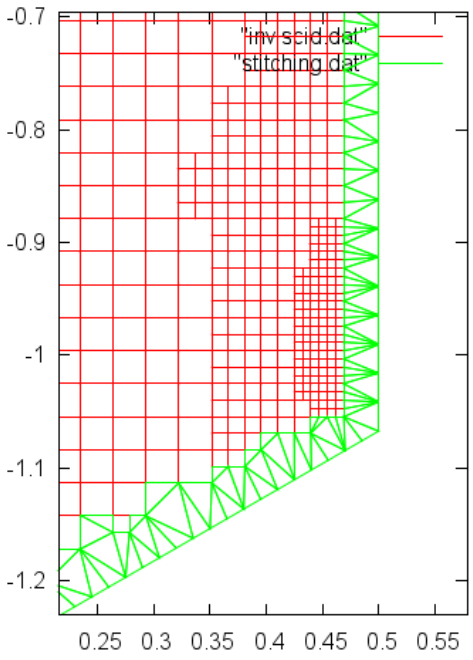


Figure 3.11 Spacing field adaptation, supersonic ramp 2

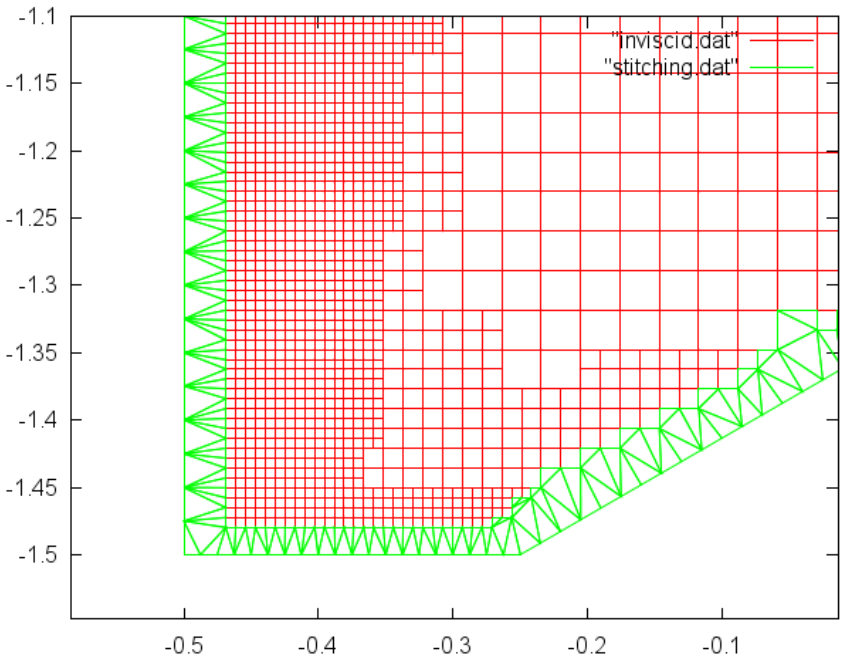


Figure 3.12 Spacing field adaptation, supersonic ramp 3

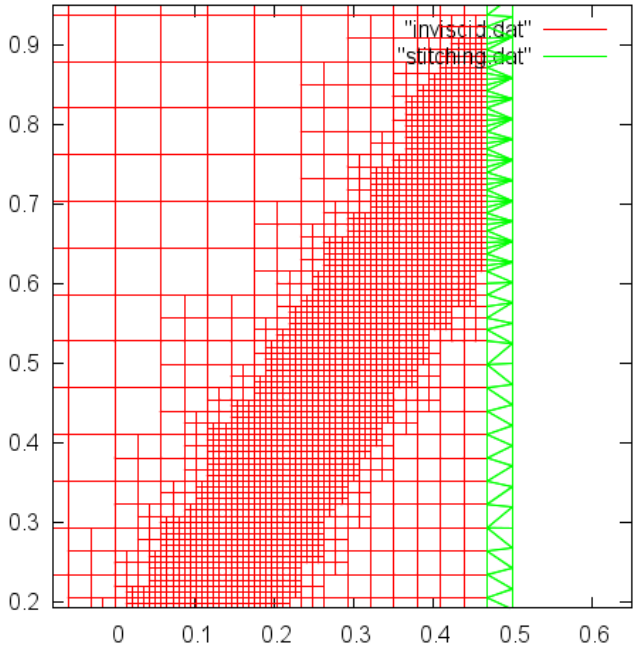


Figure 3.13 Spacing field adaptation, supersonic ramp 4

CHAPTER 4

GENERATING THE BACKGROUND MESH

Tree basics

Conceptually, trees represent a method for organizing data in a hierarchical fashion; in Computer Science, they provide infrastructure for efficiently managing and searching through large amounts of data. Information is stored at nodes, which are linked together. Each node has a parent and possible children. There is a single node that has no parent which is called the root. This node, shown in red in Figure 4.1, is the beginning of the tree and is the only node that is directly accessible. The other nodes must be accessed by traversing the tree (see Figure 4.2(a)), though once at a leaf, it is possible to traverse to another desired node without starting back at the root as shown in Figure 4.2(b). Nodes that have no children are called leaves and are shown in blue in Figure 4.1.

In C++, in which the proposed method was developed, this type of data structure can be implemented using class objects that have pointers to their parent's children. The traversals shown in Figures 4.2(a) and 4.2(b) are accomplished by recursively dereferencing either the pointer to the parent or the pointer to the child depending on the direction of the traversal. The additional data and functions specifically related to the split-tree will be discussed in greater detail in the Data Structures section of this chapter.

Spatial discretization via tree structures

Tree structures provide the framework for hierarchical mesh generation. First, a super-cell that contains the entire region of interest must be created. This super-cell is a volumetric

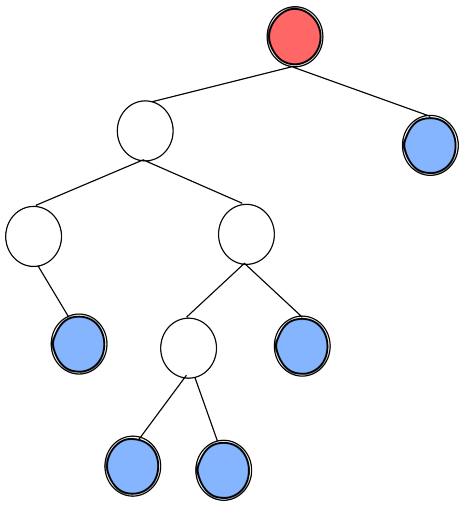
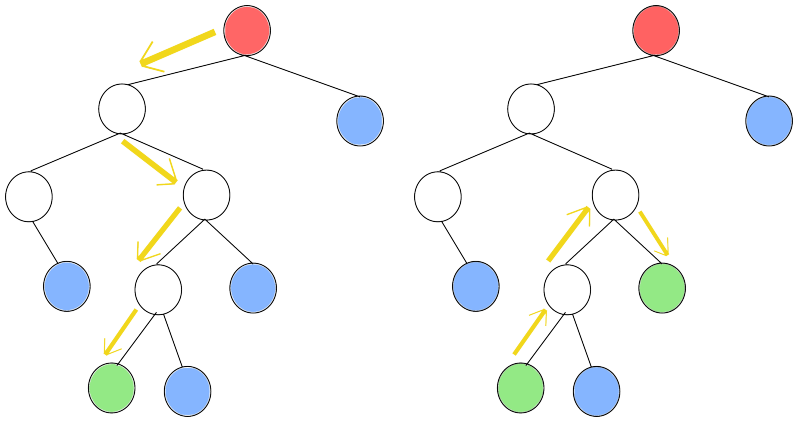


Figure 4.1 In this basic tree, the nodes are represented by circles and the links by straight lines. Note that the links are bidirectional, so it is possible to travel up or down the tree if needed.



(a) The path from the root to the node of interest is shown by yellow arrows.

(b) The path from the current node to the new node of interest is shown by yellow arrows

Figure 4.2 Tree traversals

pixel, or voxel, and it is the root of the tree. As shown in Figure 4.3, the geometry and outer boundaries must be entirely contained in the root voxel. Here, the red ellipse is the geometry, the green circle is the outer boundary, and the black box is the root voxel.

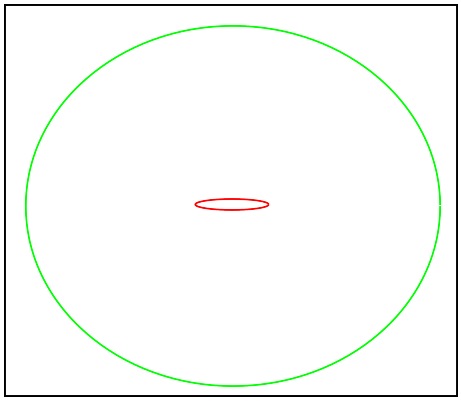


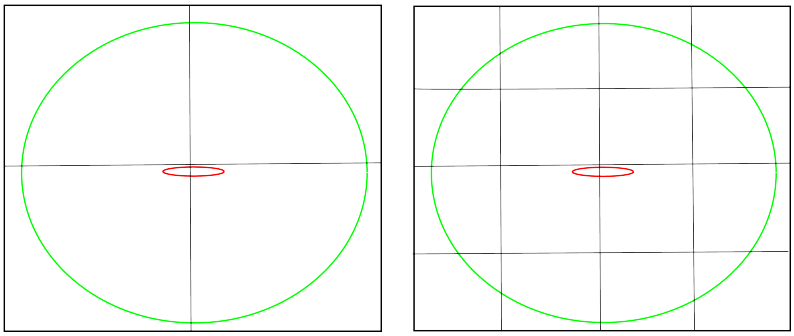
Figure 4.3 The root voxel contains all regions of space that will be discretized.

The root voxel must then spawn children who represent sub-regions of the space contained in the root voxel. The children then spawn their own children, subdividing the space even further, and this process continues recursively until the leaves of the tree have the desired size.

Split-trees, Quad-trees, and Omni-trees

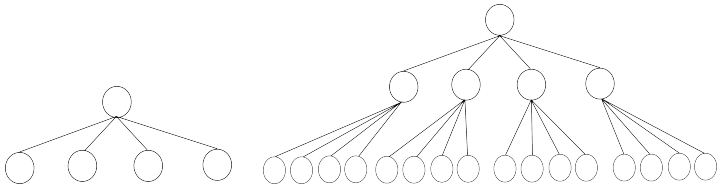
There are several different methods for subdividing each voxel, and each method has its own advantages and disadvantages. The quad-tree is the most popular due to its straightforward implementation. In a quad-tree, each voxel that is refined is divided into four equal sub-regions as shown in Figure 4.4(a) and 4.4(b). The tree structure at both refinement levels is shown in Figures 4.5(a) and 4.5(b).

In a split-tree, each voxel is split into two sub-regions instead of four. It is capable of generating a discretization that matches that of a quad-tree, but the path to get there is different; this process is shown in Figures 4.6(a), 4.6(b), 4.6(c), and 4.6(d). The split-tree has the additional capability of generating non unit aspect ratio elements as shown in the intermediate steps 4.6(a) and 4.6(c). The structure of the tree for each of the four steps is



(a) One level of refinement (b) Two levels of refinement

Figure 4.4 Quad-tree refinement

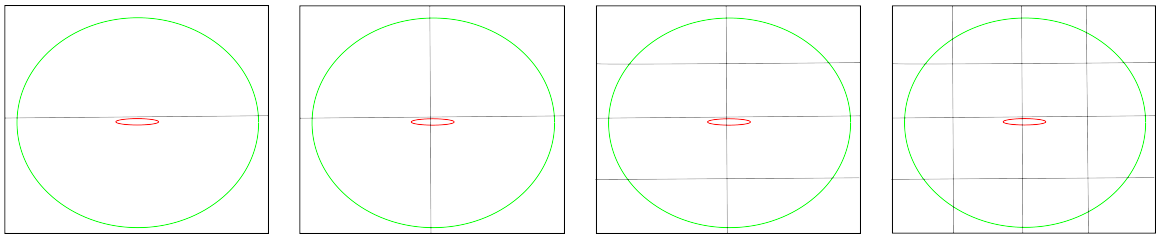


(a) One level of refinement (b) Two levels of refinement

Figure 4.5 Quad-tree structure

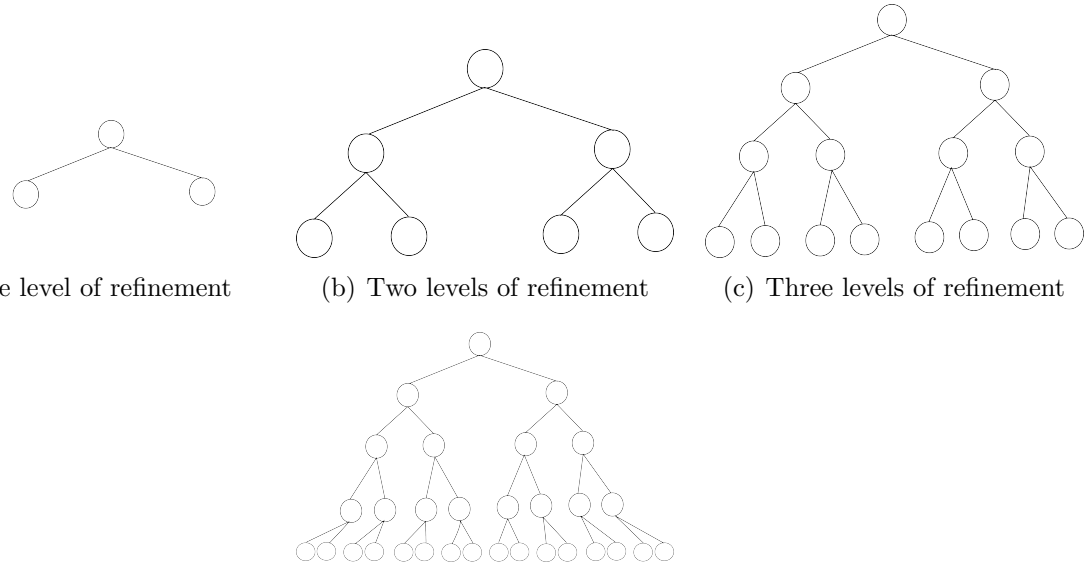
shown in Figures 4.7(a), 4.7(b), 4.7(c), and 4.7(d). The split-tree requires more nodes in the tree to achieve the same discretization as the quad-tree, which means that its memory overhead will be higher, but the increase in cost allows the method to be more flexible and robust by generating a non uniform discretization.

The omni-tree is the most logically complex of the three structures in that each voxel can divide into either two or four children. This means that it can generate the same discretization generated by the quad tree with the same number of refinement levels, but it also has the ability to generate the non unit aspect ratio elements that can be produced by the split-tree. The final structure that the omni-tree has after generating the same discretization is identical to that of the quad-tree as well, but managing the tree along



(a) One level of refinement (b) Two levels of refinement (c) Three levels of refinement (d) Four levels of refinement

Figure 4.6 Split-tree refinement



(a) One level of refinement (b) Two levels of refinement (c) Three levels of refinement

(d) Four levels of refinement

Figure 4.7 Split-tree structure

the way is more complex. If the root voxel is first split horizontally (Figure 4.7(a)), then its children determine that they must be split vertically, the discretization will match that of Figures 4.5(a) and 4.7(b), but its tree will have the form of the split-tree in Figure 4.7(b). Since its structure should match that of the quad-tree, the omni-tree root will have to "adopt" its grand children to correct its structure. While this allows the omni-tree to

have the low storage requirements of the quad-tree and the flexibility and robustness offered by the split-tree, implementing the adoption process is cumbersome.

The split-tree was chosen for the proposed method because it offers superior robustness when compared to the quad-tree, while maintaining relative simplicity that is not offered by the omni-tree. Generating non unit aspect ratio elements can have the advantage of lower overall element counts for certain mesh cases, particularly when adapting to spacing information generated from solution data.

Data Structures

Voxel

The proposed method uses a split-tree during the generation of the background mesh. Each element of the tree, called a volumetric pixel, or voxel [17], stores the data in Table 4.1.

Table 4.1 Voxel datastructure

Datatype	Count	Description
Voxel*	3	pointers to children and parents
double	4	extents of the voxel
int	5	integer codes
int*	1	pointer to array of global node numbers
double*	1	pointer to array of points

The Voxel* pointers point to the left child, right child, and parent of the voxel, and they allow traversal up and down the split-tree. As discussed earlier, the root is the only voxel with no parent, thus its parent pointer is null. Similarly, the leaf voxels have no children and their left and right child pointers are also null. The extents of the voxel, which are the minimum and maximum x and y coordinates, which are stored as two ordered pairs: the

coordinates of the bottom left corner and the top right corner of the voxel. From these, all of the corners and mid-edge nodes can be calculated. There are five integer codes or flags that are used in different phases of the generation process. The first is the split code, which determine how the voxel is split and the status of its children as depicted in Table 4.2.

Table 4.2 Split code values

Split value	Meaning
0	no split
1	split on the x-axis
2	split on the y-axis
3	left child is only child
4	right child is only child
5	childless

During the geometry processing phase, the values are constrained to 0, 1, and 2, since all voxels that are split will have exactly two children. During the voxel removal phase however, voxels will be deleted from the tree, and the other values are needed. The second integer code is a status flag used for marking cells during the flood-fill algorithm of the voxel removal section. The flag is initialized to 0, then set to 1, 2, or 3 if the voxel is found to be in violation of a boundary, internal to the computational domain, or external to the computational domain respectively. The third integer code is used for determining cell type during the unique node and element creation phase. Cell types are quadrilateral, pentagon, and hexagon, depending on the number of faces that the voxel has. The two remaining integers are used to store the global element number of the voxel and the number of nodes in the voxel's node array during the node and element creation phase. The double* and integer* pointers are both null until the node and cell creation phase, when they are initialized to store the ordered pairs and global node numbers of the voxel's vertices. This translates to large savings in memory overhead compared to having fixed static arrays for

two reasons. First, half of the voxels are not leaves and have no use for the arrays. Second, static arrays would have to have space for six nodes for each voxel, but only a fraction of the voxels actually have six nodes.

Important functions

Neighbor search

Generating and maintaining a neighbor connectivity map can be cumbersome and wasteful as the element count of the mesh rises. For this reason, the proposed method utilizes an efficient novel neighbor searching method similar to that used in FASTAR . A voxel can have a maximum of twelve neighbors as shown in Figure 4.8.

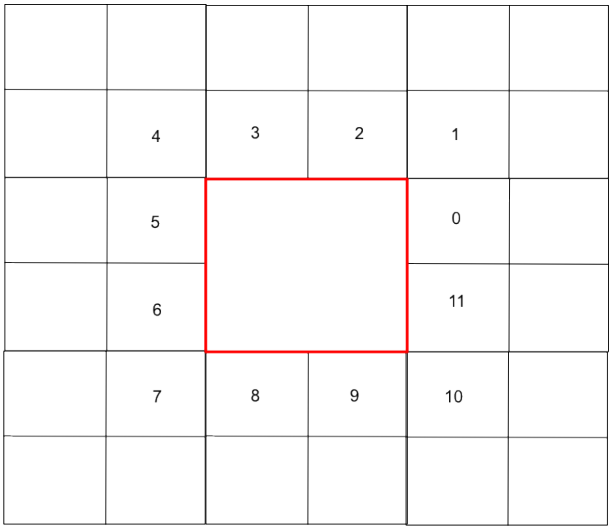


Figure 4.8 Neighbor ordering convention

To find a neighbor voxel, an offset point is created that lies inside the neighbor. The offset point is then passed to the root of the tree to initialize the search. The algorithm then recursively determines which child contains the point and passes it to that child. When a

leaf of the tree is reached, the neighbor has been found, and a pointer to that element is returned. This process is visualized in Figure 4.9

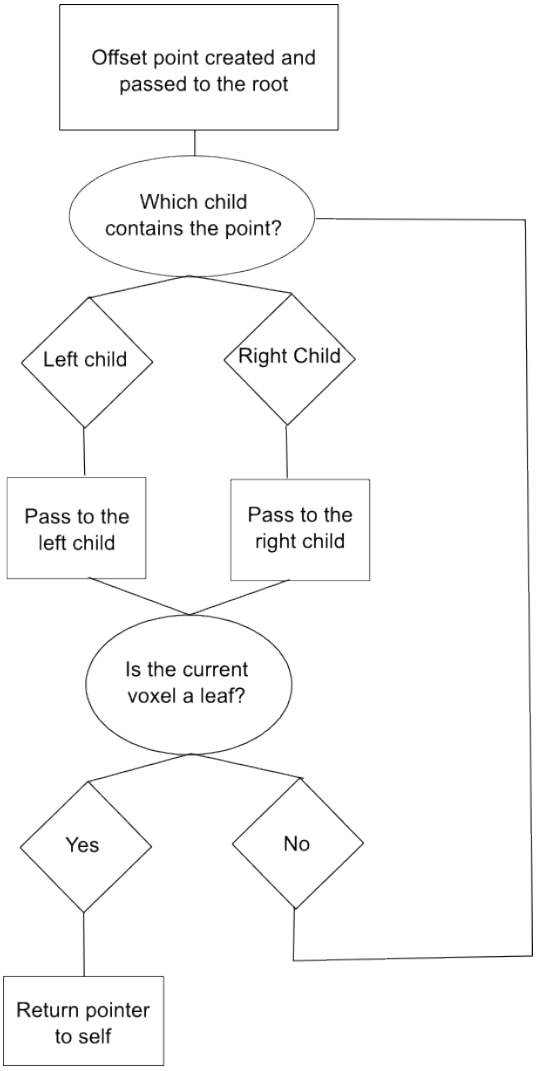
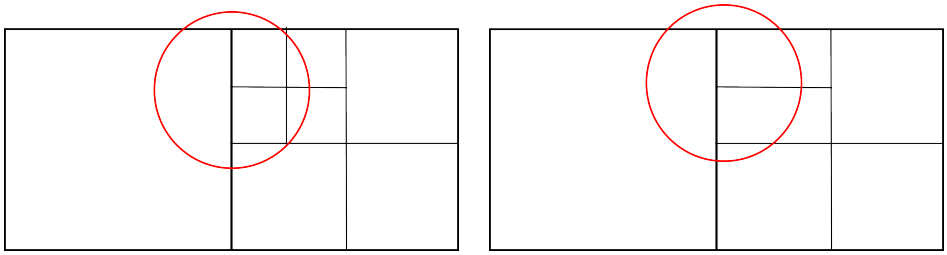


Figure 4.9 Neighbor search

Required Quality Constraints

As the mesh is refined, each voxel must maintain a set of prioritized quality metrics, without which, a valid mesh would not be possible. The highest priority metric is the aspect ratio limit. Any voxel that exceeds the aspect ratio limit is immediatly refined in the appropriate cardinal direction to bring it within tolerance. Secondly, each voxel is only allowed two neighbors per face, so it must check its size against that of its across-face neighbors. In Figures 4.10(a) and 4.10(b), for example, the voxel on the left requests the height of neighbor 0 (see Figure 4.8). The neighbor's height is one fourth of the left voxel, thus the voxel must be refined. In both of these examples, the voxel on the left would

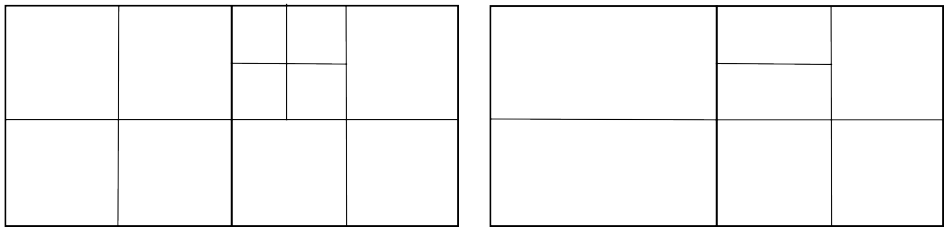


(a) Isotropic case (b) Anisotropic case

Figure 4.10 4 to 1 Violations

require a horizontal split to pass the four-to-one neighbor ratio check. In the isotropic case, the voxel's children would also be split vertically to maintain unit aspect ratio. Figures 4.11(a) and 4.11(b) show the results of refining to correct the violations in Figures 4.10(a) and 4.10(b).

As each geometry segment is processed, the voxels in the mesh will continue to refine to meet the spacing dictated by the segment's Riemannian metric tensor. In the event that the



(a) Isotropic case (b) Anisotropic case

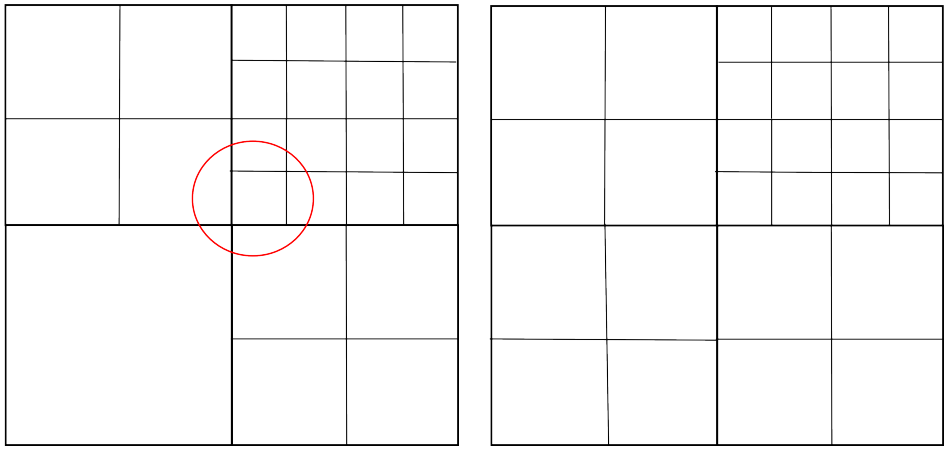
Figure 4.11 After refinement

global minimum spacing is reached, however, refinement will cease. Only if the voxel has surpassed the aspect ratio limit can it be refined once more.

Desired Quality Constraints

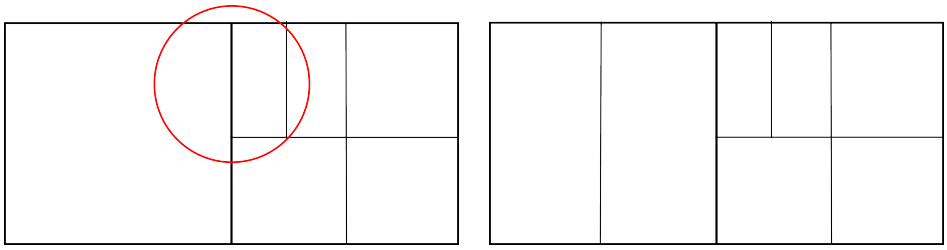
The required quality contraints represent the absolute bare minumum requirements for generating a valid mesh. There are other desirable attributes, however, that make a mesh suitable for computational fluid dynamics and other applications. Cell to cell gradation should be smooth, which means that a given voxel should not be much greater or smaller than any of its neighbors. This issue is already partially solved by the required four-to-one constraint, but there are several cases that must still be addressed. The first case involves corner neighbors (Figures 4.12(a) and 4.12(b)). In this example, the bottom left voxel requests the dimensions of neighbor 1 (see Figure 4.8). The neighbor's edge lengths are one fourth of the edge lengths of the voxel, so it must be refined.

The second case, only applicable in anisotropic scenarios, involves a neighbor that is split in the direction parallel to the shared face (Figures 4.13(a) and 4.13(b)). In this scenario, the left voxel requests the width of neighbor 0 (see Figure 4.8). The neighbor's width is one fourth that of the voxel, so the voxel must be refined.



(a) Before refinement (b) After refinement

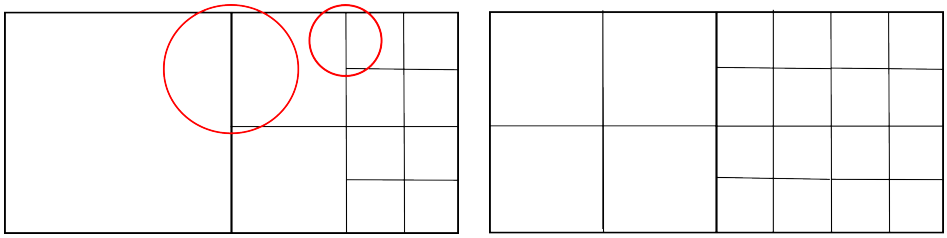
Figure 4.12 Corner gradation



(a) Before refinement (b) After refinement

Figure 4.13 Anisotropic gradation

Even with control over cell to cell gradation, meshes still tend to grow coarse too quickly. To address this issue, a cross-cell gradation check is performed. This insures that the cell size does not jump twice in a row, providing a small buffer between increases in cell size. A violation of the cross-cell gradation parameter is shown in Figures 4.14(a) and 4.14(b). In this case, the top middle voxel requests the dimensions of both neighbor 0 and neighbor 5 (see Figure 4.8). Neighbor 0's edges are one fourth that of neighbor 5, so the voxel must be refined.



(a) Before refinement (b) After refinement

Figure 4.14 Cross-cell gradation

Generation of Non Unit Aspect Ratio Elements

Since a split tree is used in place of a quad tree, it is possible to generate non unit aspect ratio elements, if desired. The user sets an aspect ratio limit that insures that elements will maintain reasonable quality. Elements that match those created by a quad tree can be obtained simply by setting the aspect ratio limit to one. Figure **??** demonstrates the

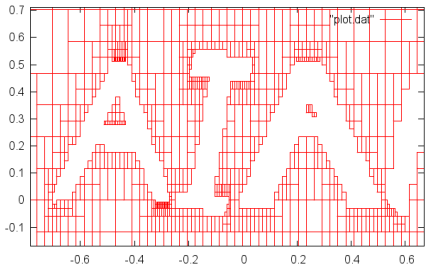


Figure 4.15 Anisotropic elements

capability to generate non unit aspect ratio elements (the aspect ratio limit is set to four instead of one in this case). This capability will prove more useful when adapting to a spacing field than shown here, generating a mesh purely based on the gemoetry.

CHAPTER 5

GENERATING VISCOUS LAYERS

The method utilized by this hybrid method to insert boundary layers is similar to a boundary layer extrusion method. For each boundary segment, a stack of high aspect ratio quadrilateral elements is created, extending normal to the surface. As the stack grows, the aspect ratio of its outermost cell is brought closer to a unit aspect ratio. A stack is terminated when either a negative area is created, or unit aspect ratio (within a tolerance) is achieved. Thus each stack may end up being different heights, both in terms of element count and physical distance from the boundary surface which can prove to be advantagous when compared to a pure extrusion method. Some care must be taken when creating stacks near sharp corners, but this issue will be discussed later.

Geometry Types

Convex Geometries

Geometries with convex or flat surfaces are excellent candidates for extrusion type meshing due to the fact that the surface normal vectors will never intersect (Figure 5.1). This allows an extrusion method to march away from the surface, creating as many viscous layers as desired by the practitioner (Figure 5.2).

Concave Geometries

Concave geometries have more limitations for extrusion type methods due to their intersecting surface normal vectors (Figure 5.3). Poor quality or negative cells will be

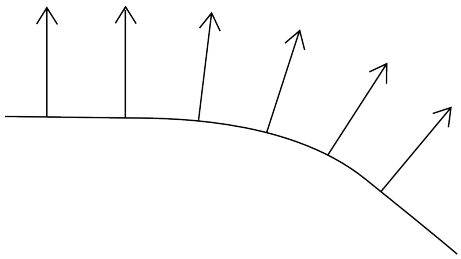


Figure 5.1 Surface normals on convex geometry

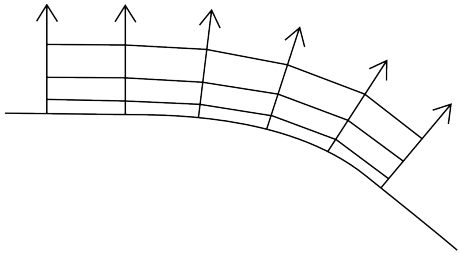


Figure 5.2 Viscous layers on convex geometry

created as the surface normal vectors approach or intersect each other (Figure 5.4). As shown in the Figure 5.4, not all of the cells generated are necessarily of poor quality, so extrusion may still work provided that the number of desired viscous layers is small where the geometry is highly concave.

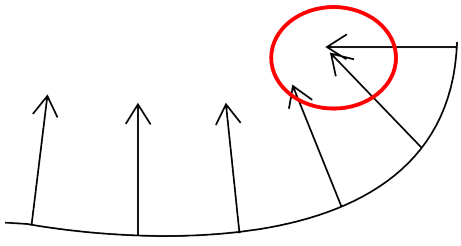


Figure 5.3 Surface normals on concave geometry

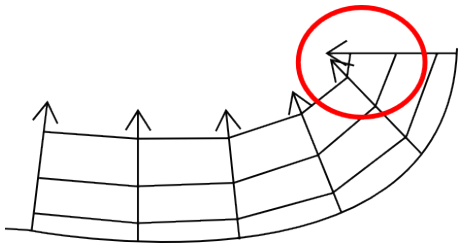


Figure 5.4 Viscous layers on concave geometry

Multi-Element Geometries

For geometries that contain multiple bodies, it is possible for the viscous layers to overlap if the bodies are in close proximity (Figure 5.5). This situation can be avoided by reducing the height of the stacks in that region.

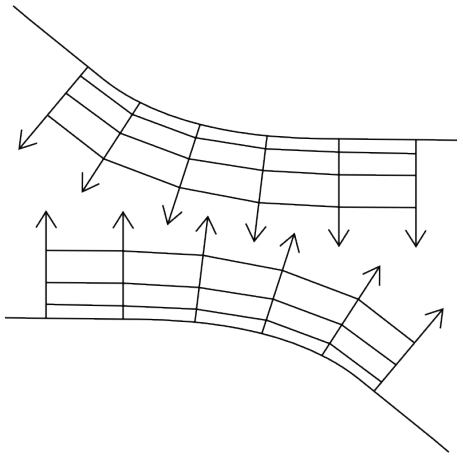


Figure 5.5 Viscous overlap of multiple bodies

Determining Stack Height

Achieving Near Unit Aspect Ratio

The height of each element in a stack is calculated using a geometric progression.

$$h = h_0(1.0 + r)^n \tag{5.1}$$

where h_0 is the spacing on the first layer, r is the growth rate (0.2 was used for the sample grids, but it is an assignable parameter), and h is the height of the nth element in the stack. Solving for n yields

$$n = log(\frac{h}{h_0})/log(r) \tag{5.2}$$

This equation is used to calculate the number of stacks required to meet the desired cell height.

Avoiding Collisions

In the event that the geometry has multiple bodies in close proximity or highly concave surfaces, it is possible for viscous fronts to overlap, which would result in an invalid mesh. To avoid this situation, a test vector is created from each surface normal that extends out three times the desired stack height, which allots a third of the space between the bodies to each opposing surface, and the remaining third for the stitching layer. The method then performs an intersection test with each geometry segment. If the test vector violates a geometry segment (Figure 5.6), then the desired stack height is reduced by a factor of three (this is also an assignable parameter). This cycle is continued until the test vector passes the intersection test. Figure 5.7 shows the new desired stack height and corresponding test vector, which no longer violates the other boundary.

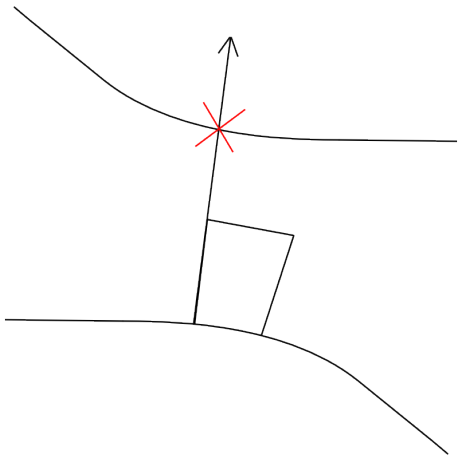


Figure 5.6 Test vector that violates a boundary

Creating Nodes

Once it is verified that there will be no colliding fronts, nodes are created along each normal vector. An orphan node occurs when a test vector is longer than both of its neighbors and no quadrilateral element can be created that includes the orphan. In Figure 5.8, the orphan node is shown in red. In the case that orphan nodes would be created, shown in Figure 5.8, they are skipped and the node count is updated.

Creating Quadrilateral Elements

After creating a list of unique nodes for the entire viscous region, a list of quadrilateral elements that span the region utilizing those nodes must be created. Since orphan nodes have already been eliminated, creating the cells for each stack is straightforward. The method simply marches along each pair of normal vectors, stopping when the last node of either is reached.

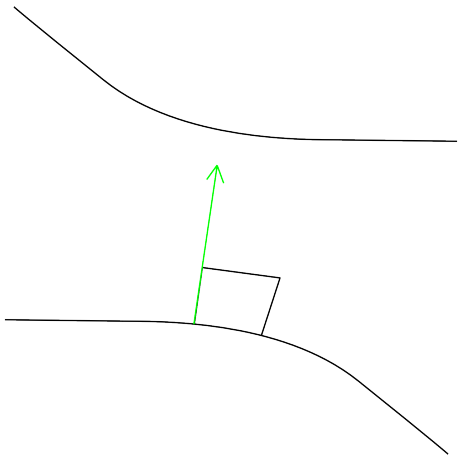


Figure 5.7 Reduced stack height eliminates the violation

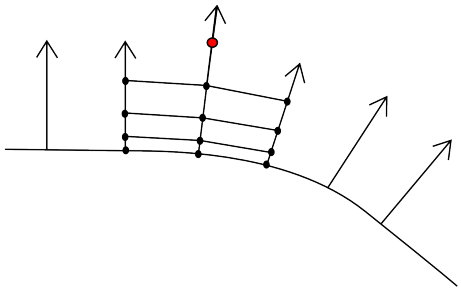


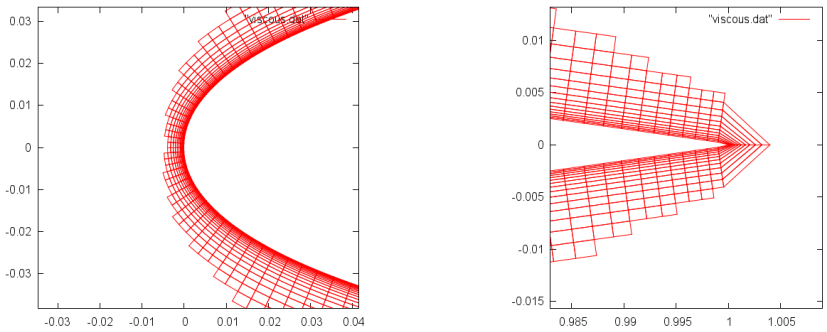
Figure 5.8 Orphan node

Removing Negative Elements

The final step of meshing the viscous region is to check for and remove negative elements. Any nodes or cells that are no longer needed are deleted, then the lists are compressed.

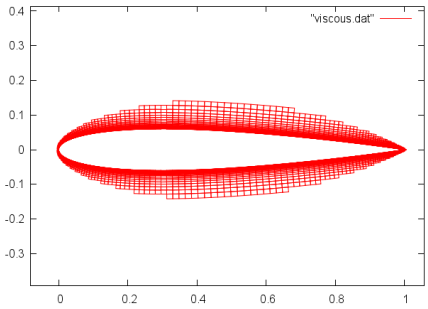
Sample Viscous Mesh

Figures 5.9(a), 5.9(c), and 5.9(b) showcase the viscous region generated for a NACA0012 symmetric airfoil.



(a) Leading edge

(b) Trailing edge



(c) Entire Airfoil

Figure 5.9 NACA0012 Symmetric Airfoil

Note that the sharp point at the trailing edge causes the local elements to be highly skewed. While there are multiple avenues available for avoiding this situation, only the one chosen for this method will be discussed in detail. In Thompson's work, the skewed elements are subdivided if beyond a given tolerance [10]. It is also possible to define separate normals at the trailing edge, which allows the stacks to grow outward in the same fashion as their neighbors. The proposed method however, simply removes the highly skewed elements entirely, leaving geometry segments at the trailing edge exposed. Doing this requires some additional footwork to ensure that a closed boundary around the viscous region is closed before starting the background meshing process, but is overall relatively simple to implement. Figure 5.10 shows the trailing edge of a NACA0012 with the skewed elements removed.

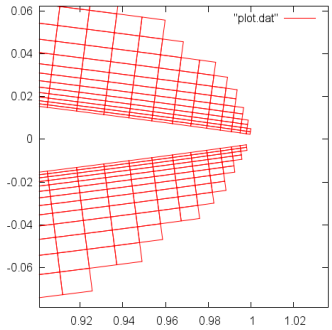


Figure 5.10 Skewed elements removed

CHAPTER 6

MESH STITCHING

The hybrid method generates several different meshes which must be assembled into a single mesh before exporting to a mesh file that can actually be used by a solver. There are two main elements to accomplishing this task, removing elements from the background mesh that are not to be part of the final mesh, and bridging the gap between the viscous and inviscid meshes.

Removal of Background Mesh Elements

Removal of the unnecessary elements of the inviscid mesh is done in two stages. First, all elements that violate a boundary segment, or come close to violating a segment within a tolerance, are removed from the mesh. This partitions the mesh into contiguous blocks that are wholly inside, or outside the computational domain. The remaining voxels are marked in or out with a recursive flood-fill algorithm detailed in Figure 6.1. It is important to note that flood-filling the contiguous blocks recursively can result in a stack overflow for large meshes because of the number of function calls that get pushed onto the call stack. There are two options for avoiding this situation: increase the maximum stack size (as done by Betro [15]), or develop a replacement routine that applies the recursive algorithm without making recursive function calls. The proposed method takes the second approach, utilizing a queue style structure. The unmarked cell checks to see if its neighbors have been marked. Each neighbor that is not yet marked is marked and then pushed onto the queue. Then, while the queue is not empty, the first element of the queue is popped off. This element checks

for neighbors that have not yet been marked, marks them, and pushes them onto the queue. When there are no voxels left in the queue, the algorithm searches for another unmarked voxel to start the process again, until there are no unmarked voxels left. The process is shown in Figure 6.1. Once all cells have been marked, the marked cells are removed from the mesh.

Triangulation

Once the voxel removal is complete, there is a gap between the viscous region and the background mesh (in the absence of a viscous mesh, there is a gap between the geometry and the background mesh). This region must be filled with cells before a valid mesh can be created. This is done in two steps. First a list of unique nodes and boundaries for the region to be triangulated are created. The boundaries consist of the exposed edges of the voxel front, the outermost edges of the viscous mesh, and any exposed geometry segments, and the unique node list contains each point that is part of any of those edges. Once these are packaged up properly, they are passed to a Delauney triangulation method written by Dr. Steve Karman, which returns a list of triangles that fill the gap region.

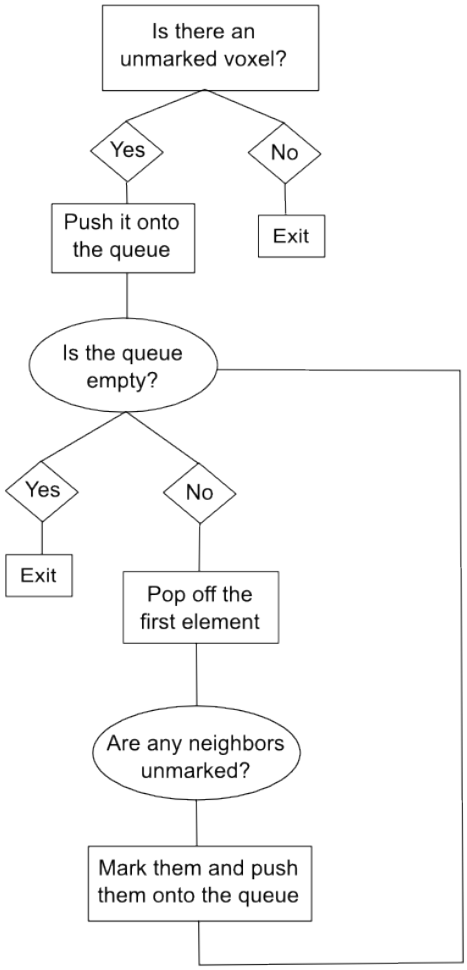


Figure 6.1 Flood-fill flow chart

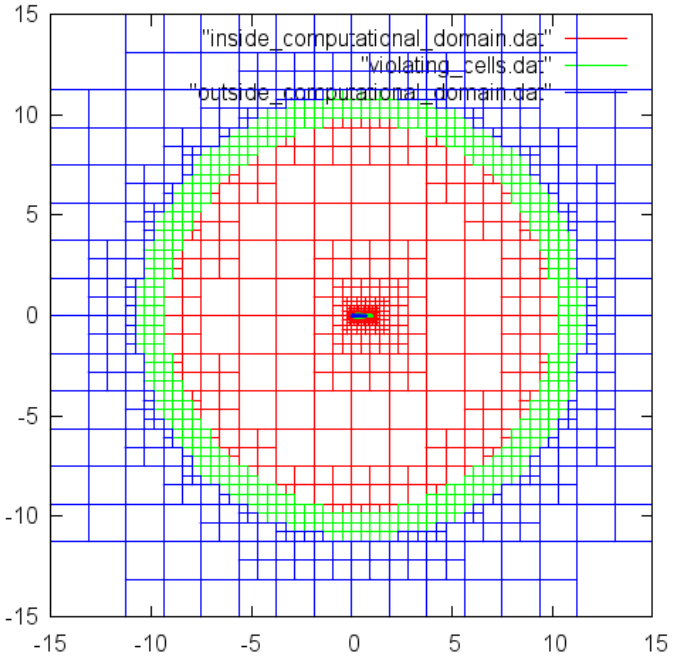


Figure 6.2 Flood fill farfield

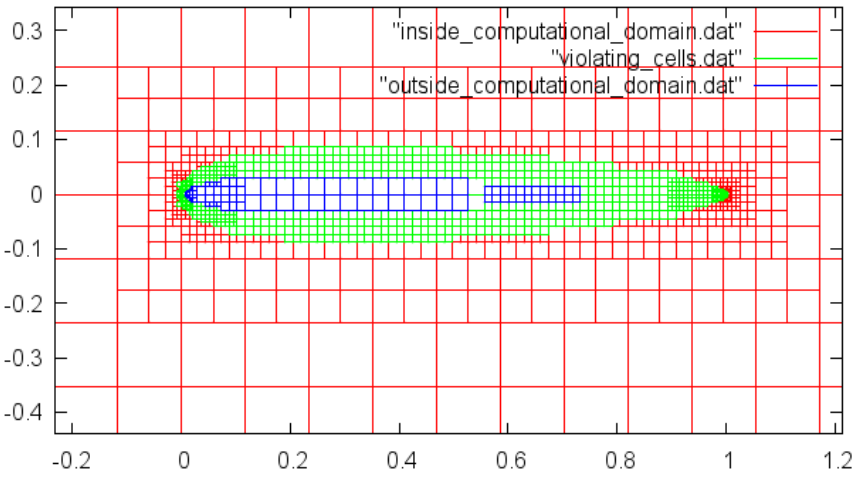


Figure 6.3 Flood fill zoomed

CHAPTER 7

EXPERIMENTAL RESULTS

This Chapter is dedicated to demonstrating that the proposed method satisfies the objectives of the research.

First, the method's ability to produce valid meshes for complex geometries is demonstrated with a battery of test geometries with different attributes. For each of these tests, full metrics are provided as evidence of the quality of the generated mesh.

Second, the method's adaptive capabilities are explored.

Robustness

Experimental results of each test case have three components: mesh visualization, quality metrics, and a brief discussion.

In each mesh visualization, the viscous regions are plotted in green, the inviscid regions in red, and the stitching layers in blue. The number of images for each test case is dependant upon the complexity of the geometry and the quantity of significant mesh features that need to be explored.

It is important to note that metrics are provided only for the viscous and stitching regions. Metrics for the cells of the background mesh are not included because they have a regular, known form. Either the element is perfectly square, within the aspect-ratio limit set by the user, or decomposed from a polygon into triangles of a known form as demonstrated in Figure 1.2 in Chapter 1.

NACA0012 Airfoil

The NACA0012 airfoil is a simple and symmetric geometry for which experimental fluid data is readily available, which makes it a popular test case for CFD applications. This, coupled with the fact that it is convex, makes it an obvious choice for the first test case. Figure 7.1 features a view of the entire mesh to show that it spans the computational domain and recovers the farfield boundaries.

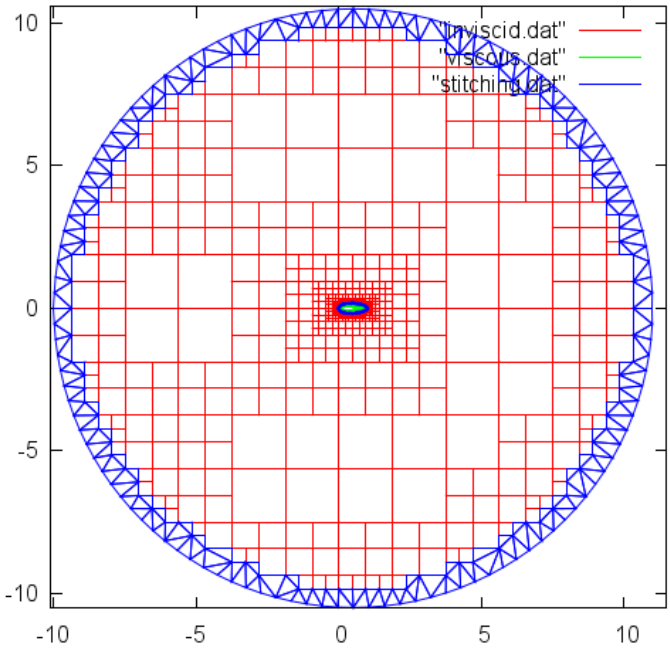


Figure 7.1 NACA0012 Airfoil farfield view

Zooming in closer to the airfoil reveals the viscous region and shows that it is reconciled to the background mesh via the stitching region, which can be seen in Figure 7.2.

In Figure 7.3, a closer view of the leading edge is shown, which makes the quality of the stitching layer in this region evident.

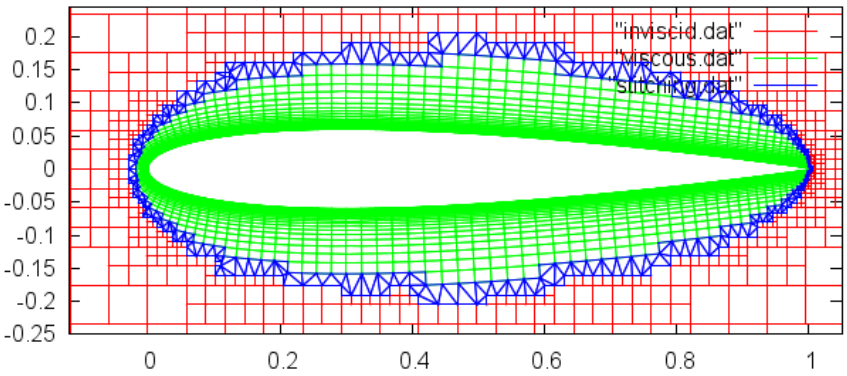


Figure 7.2 NACA0012 Airfoil zoomed

Figure 7.4 shows the blunted trailing edge of the airfoil. The quality of the elements in the stitching layer here is high, but it is important to note the large change in cell size from the last element in the stack of quadrilaterals on the blunt face to the triangle which connects it to the background mesh. This could cause problems for the solver and make it difficult to accurately resolve the wake. This issue is not currently addressed by the proposed method, but will be dealt with in future research.

Table 7.1 contains the quality metrics for the viscous and stitching regions. Note that the aspect ratios for the viscous region are very high, but the skewness of the cells is close to one. This indicates that the cells are able to provide the desired off-wall spacing with long thin elements, but that the quality of the elements is high with regard to the angles.

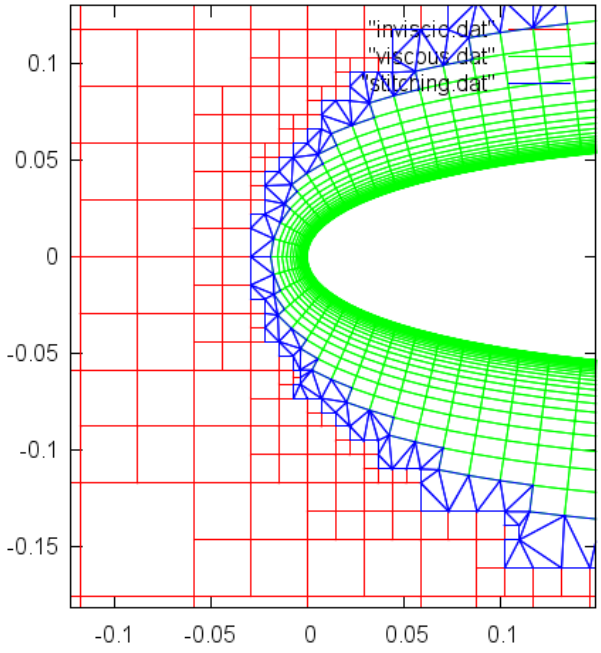


Figure 7.3 NACA0012 Airfoil leading edge

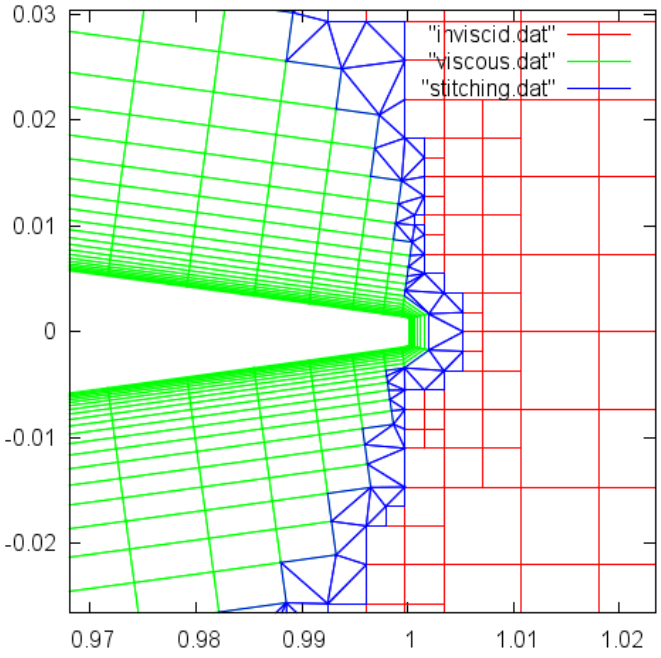


Figure 7.4 NACA0012 Airfoil trailing edge

Table 7.1 NACA0012 mesh quality metrics

	Stitching Region	Viscous Region
Number of grid points	747	3423
Number of cells	748	3261
Minimum area	4.193114e-7	6.582601e-8
Average area	6.457930e-2	5.415438e-5
Maximum area	3.596344e-1	4.903941e-4
Minimum angle	8.324267	30.787662
Average angle	60.000000	90.000000
Maximum angle	143.852820	153.950903
Minimum aspect ratio	1.000664	1.218785
Average aspect ratio	1.247216	31.151046
Maximum aspect ratio	5.391309	201.776287
Minimum skew	1.026749	1.000000
Average skew	1.575581	1.000469
Maximum skew	6.099028	1.124196
Minimum condition number	1.000585	1.019453
Average condition number	1.197931	15.659699
Maximum condition number	4.063379	100.895213
Minimum jacobian	0.144775	0.439141
Maximum jacobian	1.000000	1.000000
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	0
Number of positive cells	748	3261

NACA0012 Biplane

The proposed method is capable of generating meshes for geometries that consist of multiple closed bodies. To demonstrate this, a biplane geometry was created by duplicating and offsetting a NACA0012 airfoil. Figure 7.5 shows the farfield view of the final mesh, which demonstrates that the computational domain is spanned and the outer boundaries are resolved.

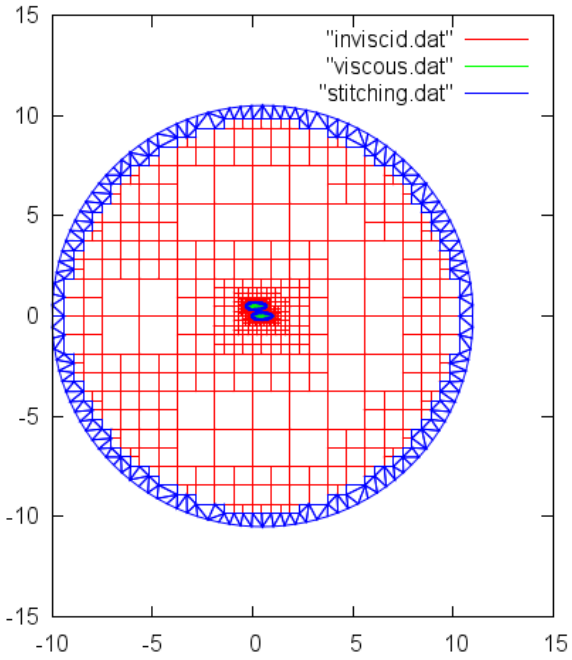


Figure 7.5 NACA0012 Biplane farfield

Zooming in closer to the pair of airfoils in Figure 7.6 reveals that viscous regions are generated around both airfoils and stitched to the background mesh. Note that the stitching layers for the airfoils are not identical. This is due to the fact that the background mesh is generated based on the edges along the viscous front, and the edges for the top airfoil do

not necessarily fall on the same subdivision of the root cell. The viscous regions themselves, however, are identical to those created in the previous test case.

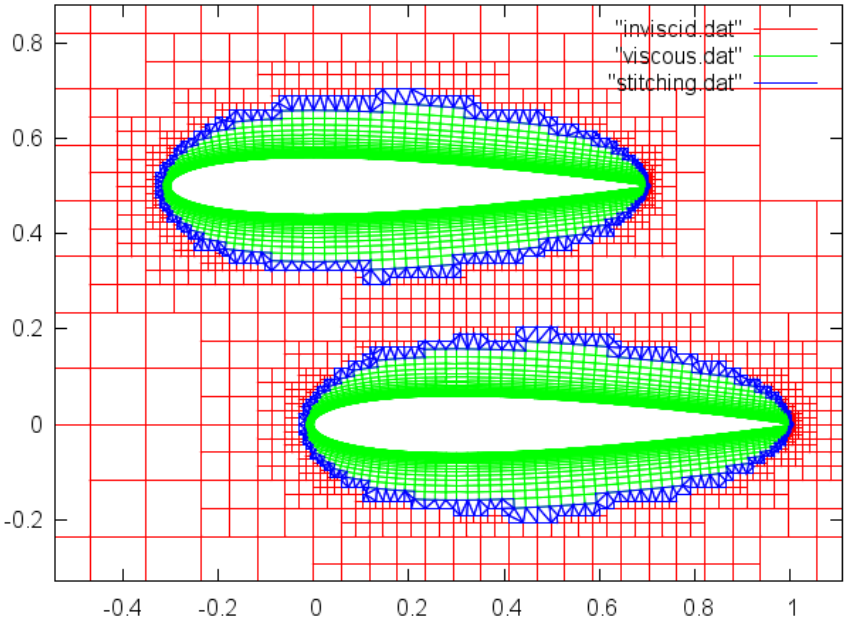


Figure 7.6 NACA0012 Biplane zoomed

Table 7.2 contains the mesh metrics for this test case. As one might expect, node and cell counts are increased compared to the previous test case, but the cell qualities are roughly the same.

Table 7.2 NACA0012 Biplane mesh quality metrics

	Stitching Region	Viscous Region
Number of grid points	1224	6846
Number of cells	1225	6522
Minimum area	4.026885e-7	6.582601e-8
Average area	3.947561e-2	5.415438e-5
Maximum area	3.596344e-1	4.903941e-4
Minimum angle	8.324267	30.787662
Average angle	60.000000	90.000000
Maximum angle	143.852820	153.950903
Minimum aspect ratio	1.000590	1.218783
Average aspect ratio	1.256484	31.151138
Maximum aspect ratio	5.391309	201.793070
Minimum skew	1.026749	1.000000
Average skew	1.603948	1.000470
Maximum skew	6.634188	1.124333
Minimum condition number	1.000523	1.019453
Average condition number	1.208859	15.659744
Maximum condition number	4.063379	100.928797
Minimum jacobian	0.144775	0.439141
Maximum jacobian	1.000000	1.000000
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	0
Number of positive cells	1225	6522

The proposed method has been shown to produce high quality meshes for convex geometries and for multi-body geometries, but it also has the ability to handle geometries which have concave surfaces. Figure 7.7 shows a farfield view of the generated mesh to verify that it spans the computational domain and properly represents the outer boundaries.

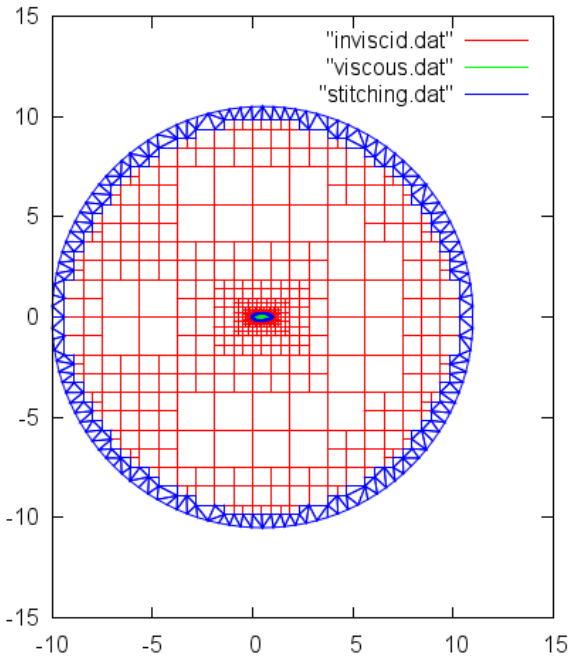


Figure 7.7 RAE2822 Airfoil farfield

Figure 7.8 shows a closer view of the mesh near the surface of the airfoil which reveals the viscous region and demonstrates that it is stitched to the background mesh. The concavity of the bottom of the airfoil is also evident in this image, but the curvature of the geometry is slight enough that the normals do not cross.

Taking a closer look at the leading edge in Figure 7.9 shows that the elements in the stitching layer have the expected qualities.

Figure 7.10 gives a similar view of the trailing edge of the airfoil. Note that the trailing edge of this airfoil was not blunted like the NACA0012, which was one of the reasons that this

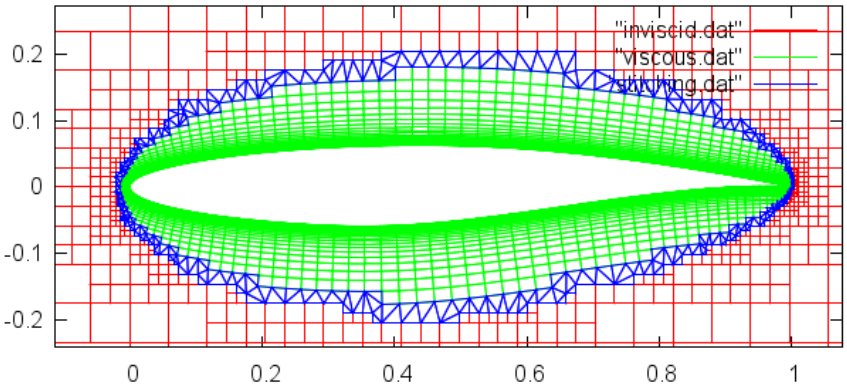


Figure 7.8 RAE2822 Airfoil zoomed

test case was included. There is still a large change in cell size from the last viscous elements in this region to the triangles in the stitching layer similar to the blunted NACA0012 case.

Table 7.3 contains the mesh quality metrics for this test case. Note that the not blunting the trailing edge had only a small effect on the skewness of the viscous elements in that region (the elements are relatively poor either way, which is why this is an area of further research).

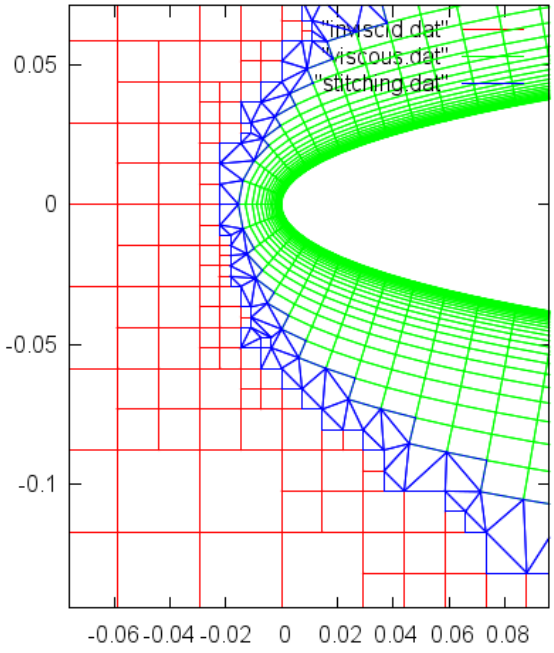


Figure 7.9 RAE2822 Airfoil leading edge

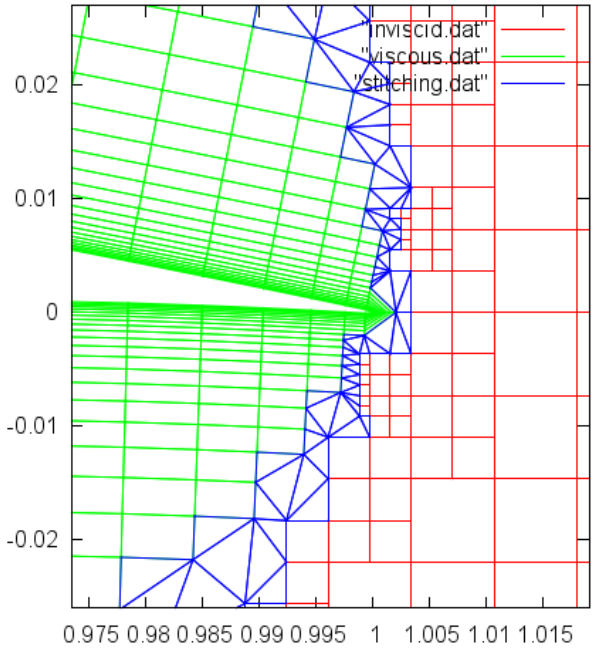


Figure 7.10 RAE2822 Airfoil trailing edge

Table 7.3 RAE2822 mesh quality metrics

	Stitching Region	Viscous Region
Number of grid points	731	3359
Number of cells	733	3199
Minimum area	3.635114e-7	5.325168e-8
Average area	6.590201e-2	5.628183e-5
Maximum area	3.596344-2	5.220440e-4
Minimum angle	9.394898	12.476478
Average angle	60.000000	90.000000
Maximum angle	135.017636	174.76525
Minimum aspect ratio	1.000382	1.225349
Average aspect ratio	1.247770	31.612420
Maximum aspect ratio	5.167589	205.833136
Minimum skew	1.022483	1.000000
Average skew	1.578627	1.000517
Maximum skew	4.477986	1.125022
Minimum condition number	1.000341	1.020474
Average condition number	1.199350	15.895431
Maximum condition number	3.700857	102.925706
Minimum jacobian	0.163238	0.091237
Maximum jacobian	1.000000	1.000000
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	0
Number of positive cells	733	3199

30P30N Multi-Element Airfoil

The proposed method has a robust viscous layer production method that can create viscous layers without introducing negative elements caused by crossed normals. To showcase this, the 30P30N airfoil was chosen. Figure 7.11 shows the farfield view of the mesh, demonstrating that it spans the computational domain and that the square farfield boundaries are preserved.

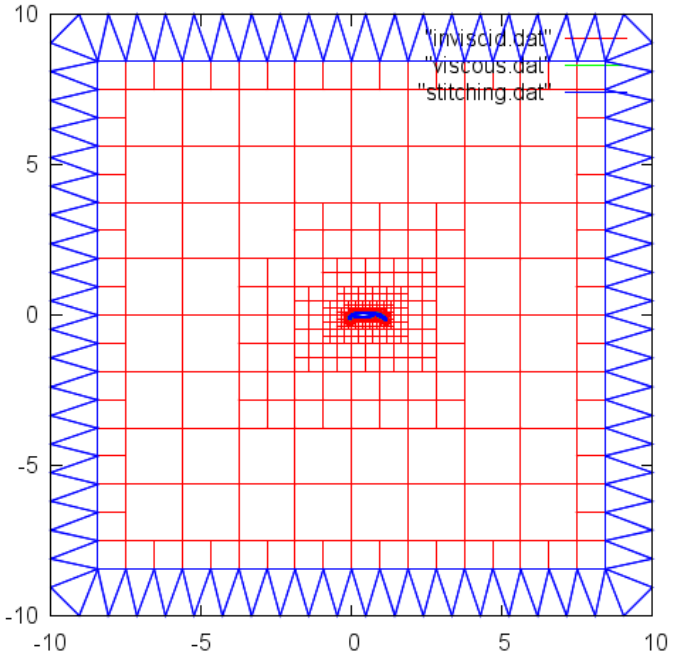


Figure 7.11 30P30N Multi-Element Airfoil: farfield view

Zooming in on the slat element, Figure 7.12 shows that viscous layers are created on the slat and the leading edge of the wing.

Taking a closer look at the bottom trailing edge of the slat in Figure 7.13 shows that the expected issues are present at the sharp point, but the mesh is valid in the region.

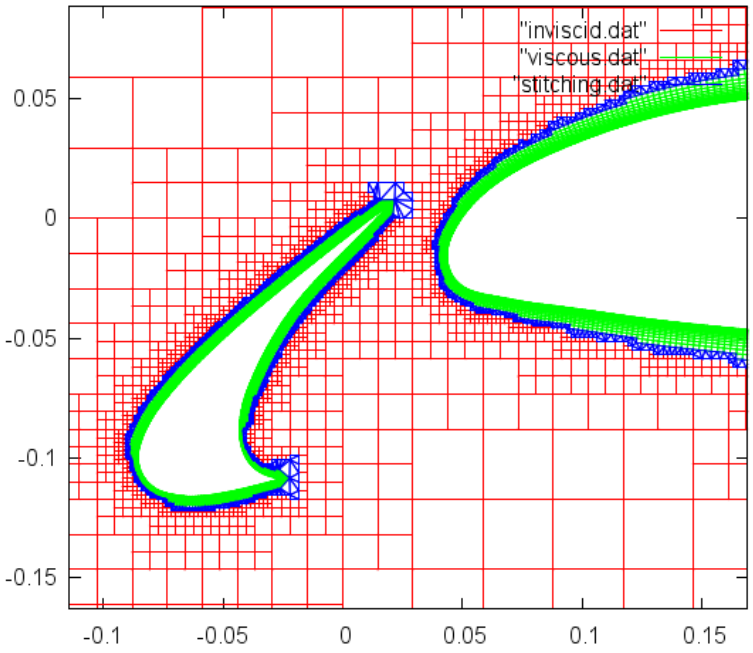


Figure 7.12 30P30N Multi-Element Airfoil: slat

The same is true of the top trailing edge of the slat as shown in Figure 7.14. Note that the elements in the stitching region are generally of high quality and appropriate size; it is only the areas where the viscous elements are skewed that the cell size gradation changes drastically between the viscous elements and the stitching elements.

Moving to the flap in Figure 7.15, it is evident that viscous layers are created at the trailing edge of the wing and around the flap element.

Zooming in on the trailing edge of the flap in Figure 7.16 reveals that it has been blunted. The mesh in this region reflects what would be expected based on the previous test cases.

Figure 7.17 shows a closer view of the region between the trailing edge of the wing and the leading edge of the flap. Again, the skewed elements at the sharp point cause a large change in cell size from the viscous elements to the stitching layer. In this case the viscous

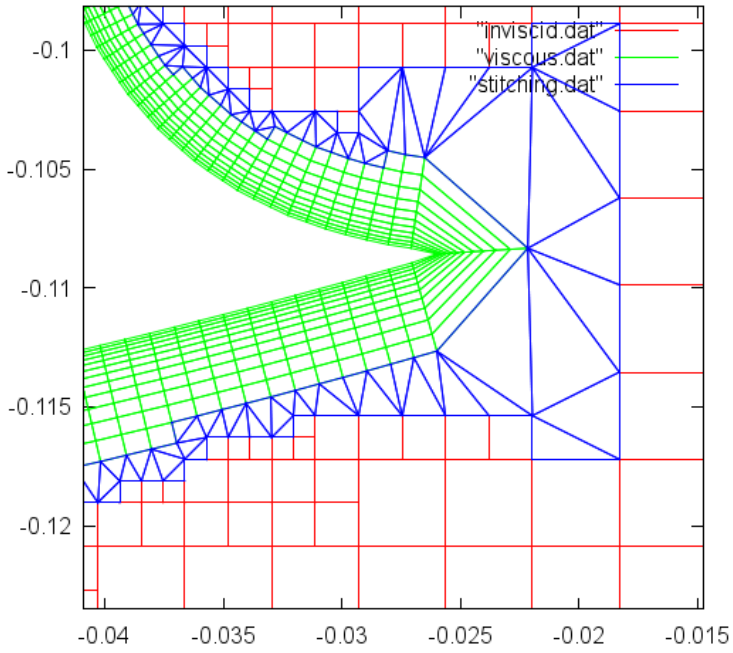


Figure 7.13 30P30N Multi-Element Airfoil: slat 2

layers are close enough that the stitching layers of the wing and the flap are joined, but this causes no problems for the triangulation method.

The most difficult region on the airfoil to generate viscous layers lies within a 90 degree corner on the wing as seen in Figure 7.18.

Figure 7.19 then shows a close up view of the mesh in this region. Notice that the height of the stacks increases as the distance from the corner increases and the likelyhood of encountering crossed normals decreases. The method seeks to create as many viscous layers as possible without violating any of the constraints that would lead to the creation of invalid elements. The elements in the corner are slightly skewed because of the extreme concavity, but they are still valid elements.

Table 7.4 contains the mesh metrics for this test case. Even with the 90 degree corner, the quality of the mesh is similar to that of the previous airfoil test cases.

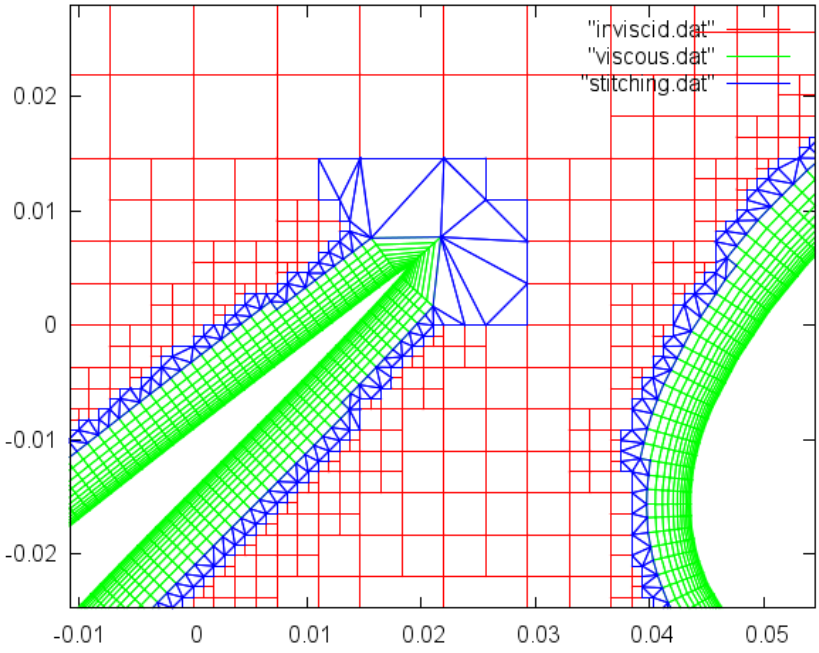


Figure 7.14 30P30N Multi-Element Airfoil: slat 3

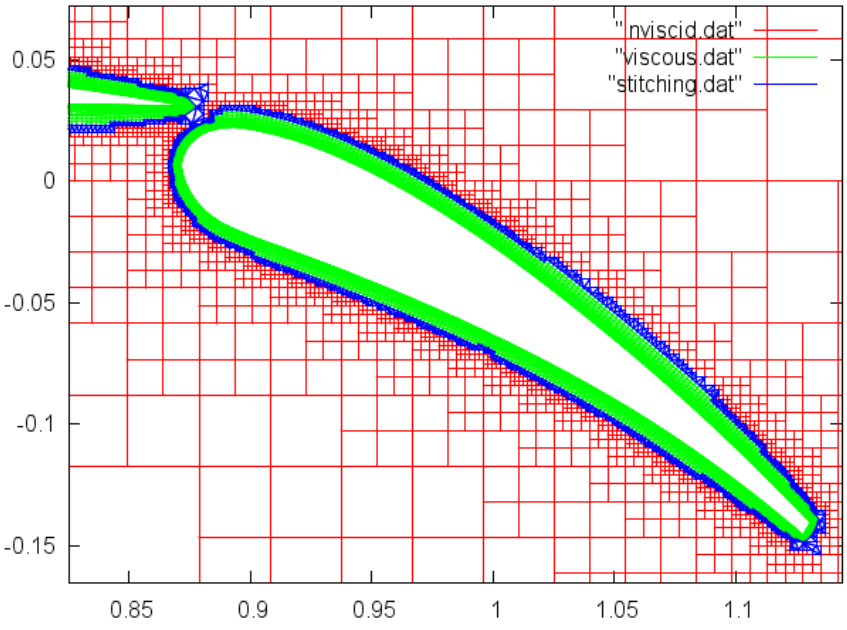


Figure 7.15 30P30N Multi-Element Airfoil: flap

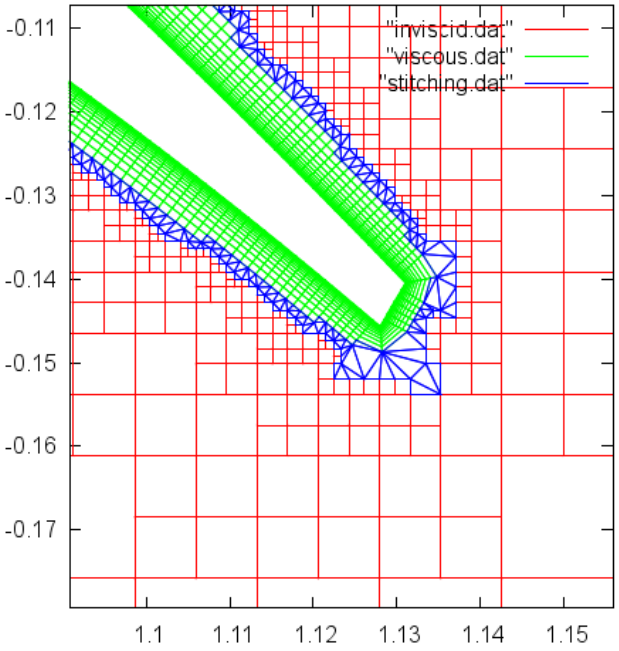


Figure 7.16 30P30N Multi-Element Airfoil: flap trailing edge

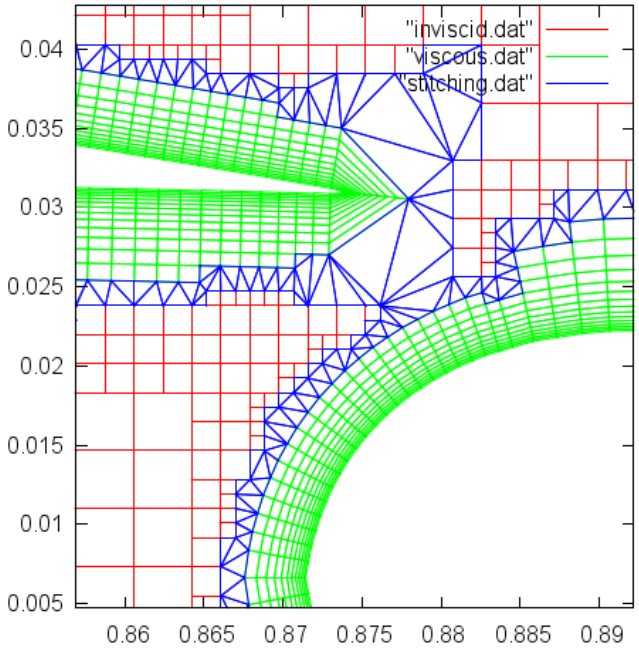


Figure 7.17 30P30N Multi-Element Airfoil: main wing trailing edge

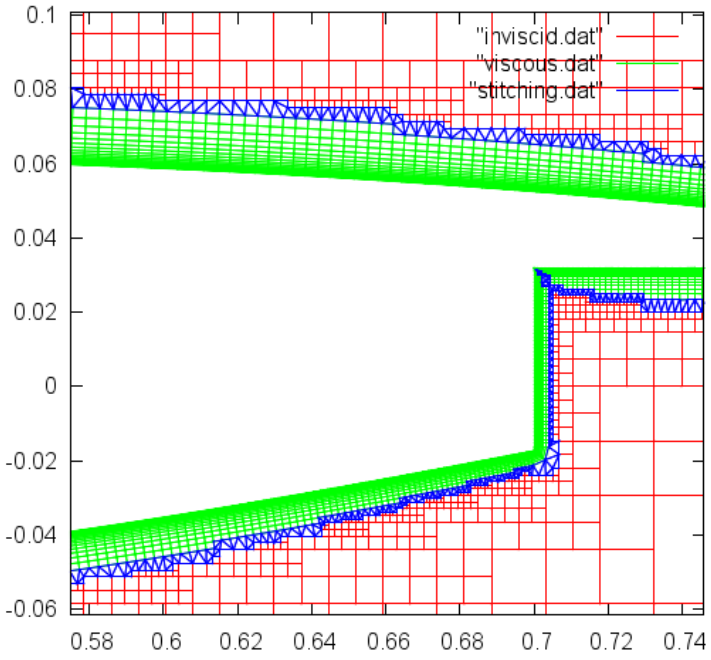


Figure 7.18 30P30N Multi-Element Airfoil: 90 degree corner

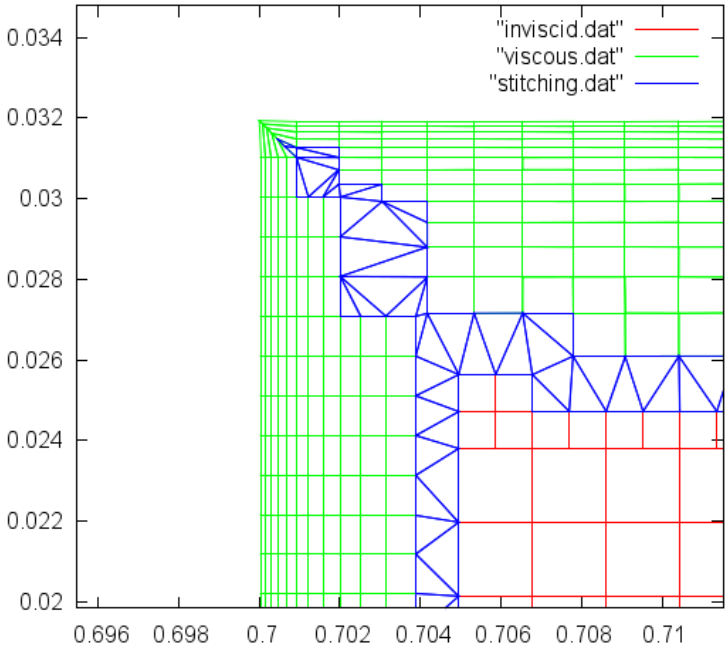


Figure 7.19 30P30N Multi-Element Airfoil: 90 degree corner zoomed

Table 7.4 30P30N mesh quality metrics

	Stitching Region	Viscous Region
Number of grid points	4160	22250
Number of cells	4163	20696
Minimum area	3.197837e-8	5.436849e-8
Average area	2.768253e-2	1.287013e-6
Maximum area	1.034581e0	1.264742e-5
Minimum angle	9.316266	13.673160
Average angle	60.000000	90.000000
Maximum angle	142.091205	171.004274
Minimum aspect ratio	1.000055	1.007600
Average aspect ratio	1.242001	6.184910
Maximum aspect ratio	5.127023	32.808325
Minimum skew	1.008561	1.000000
Average skew	1.592621	1.001729
Maximum skew	4.992068	1.507928
Minimum condition number	1.000048	1.000117
Average condition number	1.200451	3.245912
Maximum condition number	3.684121	29.757965
Minimum jacobian	0.161884	0.156361
Maximum jacobian	1.000000	1.000000
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	0
Number of positive cells	4163	20696

2D Fuel Cell Slice

The geometry for this test case is a 2 dimensional slice of a hypothetical fuel cell. It was chosen to showcase the proposed method's ability to handle many different geometries at once, and also to demonstrate that it can be used to generate grids for applications other than airfoils. Figure 7.20 shows a farfield view which shows the configuration of the rods.

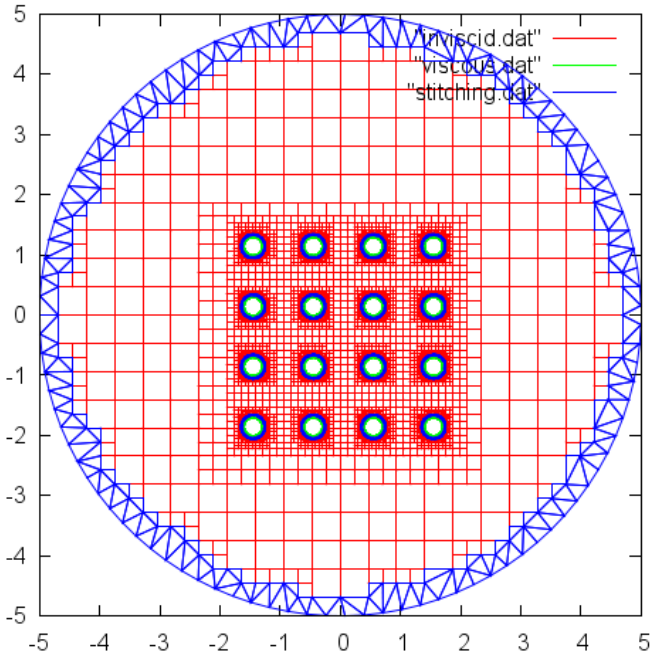


Figure 7.20 2D Fuel Cell Slice

Zooming in on four of the elements reveals that each rod is encapsulated by viscous layers (Figure 7.21).

Zooming in further in Figure 7.22 shows that the viscous layers are uniform and the stitching layer consists of high quality elements.

Table 7.5 contains the mesh metrics for this test case, and it confirms that the mesh is indeed a high quality mesh.

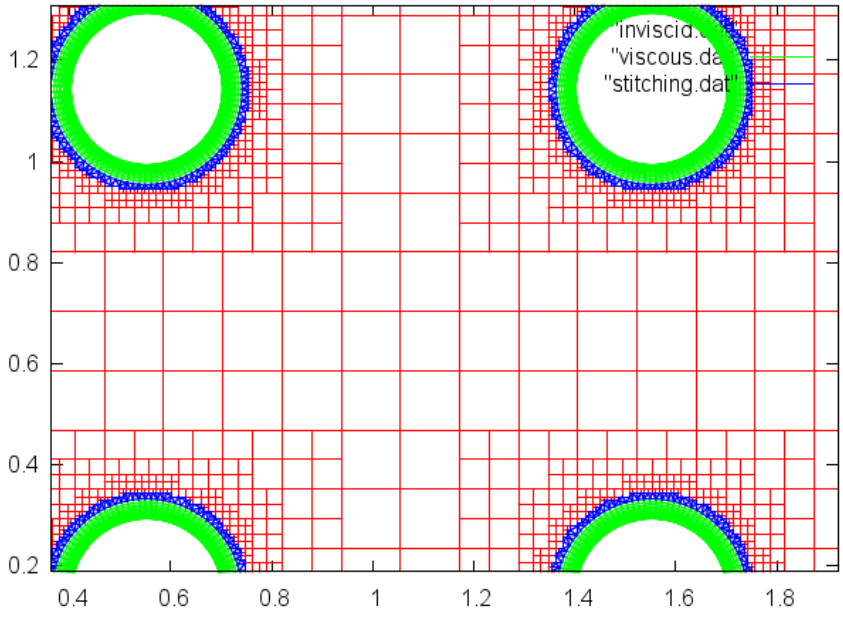


Figure 7.21 2D Fuel Cell Slice zoomed

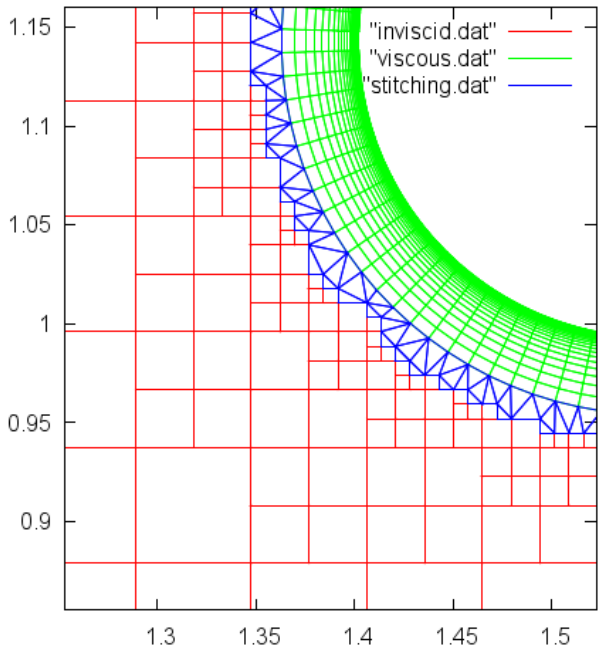


Figure 7.22 2D Fuel Cell Slice zoomed again

Table 7.5 2D Fuel cell slice mesh metrics

	Stitching Region	Viscous Region
Number of grid points	4979	38016
Number of cells	4979	36432
Minimum area	2.682049e-5	1.137010e-6
Average area	2.470077e-3	1.831704e-5
Maximum area	9.310863e-2	7.813419e-5
Minimum angle	21.604212	87.692903
Average angle	60.000000	90.000000
Maximum angle	120.251172	92.313717
Minimum aspect ratio	1.000065	1.781668
Average aspect ratio	1.223053	20.670826
Maximum aspect ratio	2.231387	79.852458
Minimum skew	1.008442	1.000000
Average skew	1.564593	1.000026
Maximum skew	2.676641	1.000174
Minimum condition number	1.000058	1.160965
Average condition number	1.185140	10.414876
Maximum condition number	1.817112	40.073491
Minimum jacobian	0.368193	0.999185
Maximum jacobian	1.000000	0.999732
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	0
Number of positive cells	4979	36432

Random Complex Geometry

The previous test cases have shown that the proposed method is robust and capable of handling many different geometries, but none have really pushed the method to the extreme. This random, multi-body, complex geometry was created to do just that. Figure 7.23 shows a farfield view of the mesh to demonstrate that it spans the computational domain and show the complexity of both geometries. It is evident that viscous layers exist around both geometries, but there are many regions that must be explored more deeply to see the full power of the proposed method.

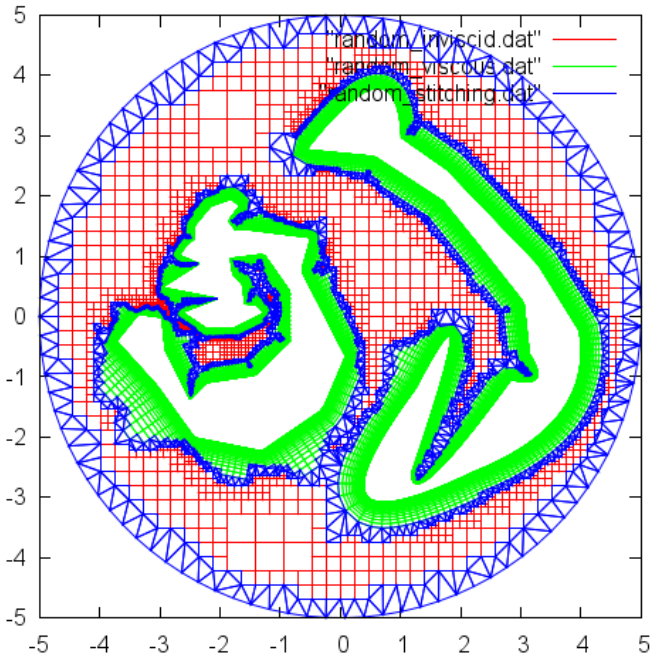


Figure 7.23 Random Complex Geometry

First, Figure 7.24 gives a closer look at the bottom half of the right geometry. It is evident that there are multiple regions where the viscous layers have been cut short of their

desired height to avoid colliding viscous fronts. There is a highly concave region near (3,-1) that appears to be densely popuplated, but requires closer inspection.

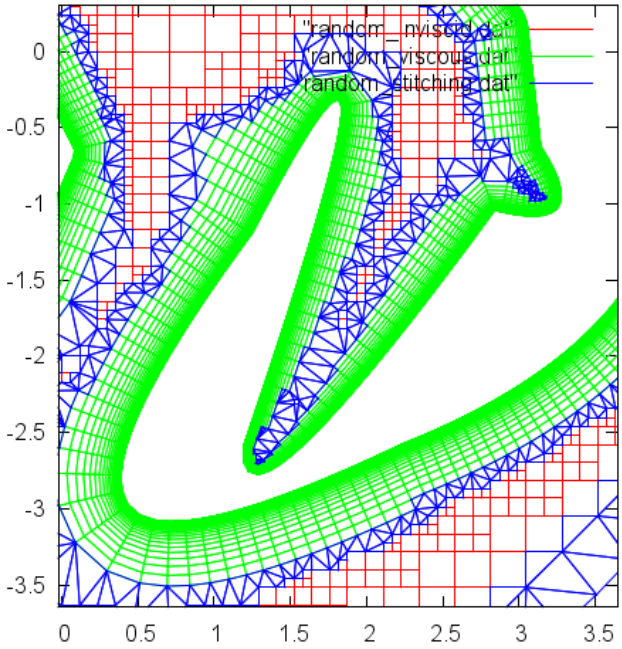


Figure 7.24 Random Complex Geometry

Figure 7.25 zooms in on this region, revealing that there are some quadrilateral elements that survived the cutting process, but are isolated in this concave region, which aids greatly in the production of the stitching region.

Figure 7.26 shows a similar highly concave region in which no background Cartesian elements exist to bridge the gap between the opposing viscous fronts, but the fronts are in close proximity such that the stitching layer still contains high quality elements.

The area between the two geometries is shown in Figure 7.27, which further demonstrates the ability to handle very broken background meshes and still reconcile all of the fronts.

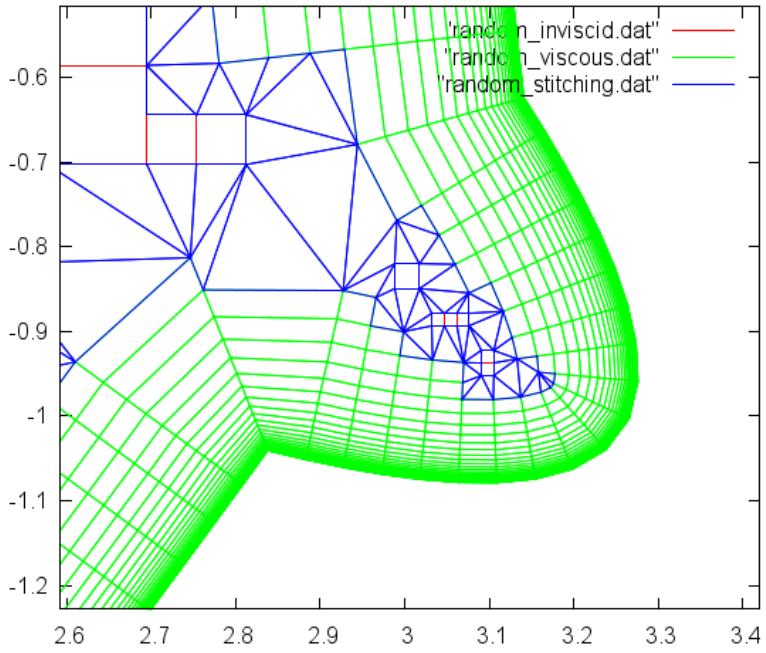


Figure 7.25 Random Complex Geometry

Figure 7.28 features the top half of the right geometry which has convex surfaces, concave surfaces and sharp points. Note that the sharp point in this case has the same kind of issues encountered in the airfoil cases, but again, the mesh is valid.

Figure 7.29 shows a curved surface meeting a flat surface, which forced the proposed method to handle crossing normal vectors. There are some skewed elements in the corner of the viscous mesh, which are caused by the fact that the normals were very close to the tolerance.

A closer look in Figure 7.30 reveals that all of the elements are still valid.

In Figure 7.31, a region is shown where normals from three different surfaces would cross, and the proposed method deals with this appropriately.

Figure 7.32 shows the most complex region of the mesh that contains multiple highly concave regions, multiple sharp corners, and multiple regions where elements from the

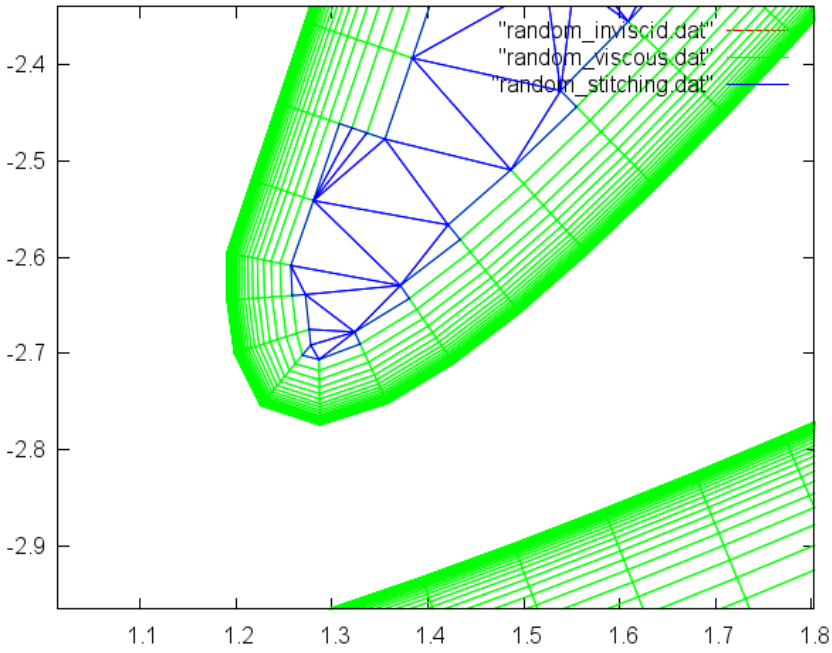


Figure 7.26 Random Complex Geometry

background mesh are isolated from the rest of the Cartesian cells. The proposed method deals with each of these situations simultaneously.

Figure 7.33 shows a view of the left element that reveals that it has multiple sharp points, and one extremely sharp corner with an angle of roughly 30 degrees.

Zooming in (Figure 7.34) shows that viscous layers on the top and bottom surfaces terminate early enough to allow for background Cartesian cells to survive the cutting process, which yields a better quality stitching region deep into the crevasse.

Finally, in Figure 7.35, a small number of viscous layers are shown to exist at the sharp corner, but they have very poor quality, which is to be expected in such a case.

Table 7.6 contains the mesh quality metrics for this test case. Several elements are positive skewed, and there are some very poor quality elements in the mesh, but the proposed method was able to create a valid mesh for an extremely complex, multi-body geometry,

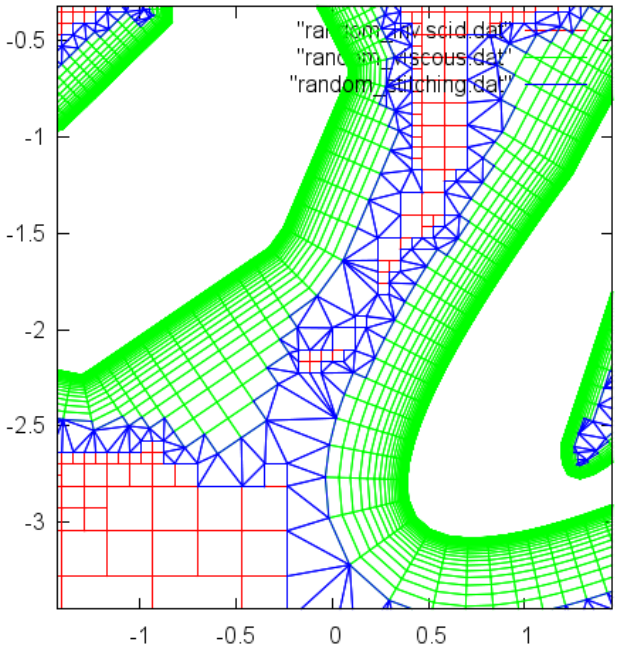


Figure 7.27 Random Complex Geometry

complete with viscous layers, high quality Cartesian cells in the background mesh, and a valid stitching layer that reconciled the viscous and inviscid meshes.

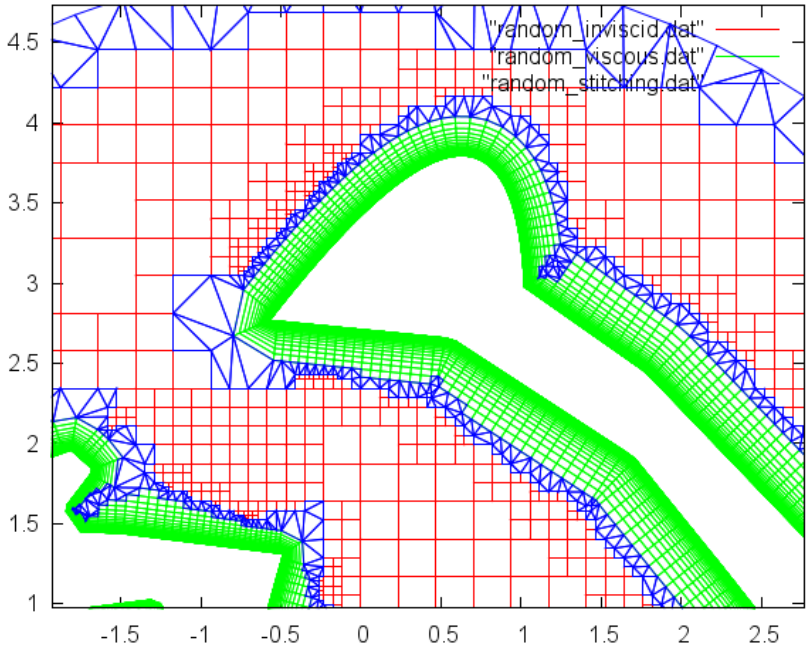


Figure 7.28 Random Complex Geometry

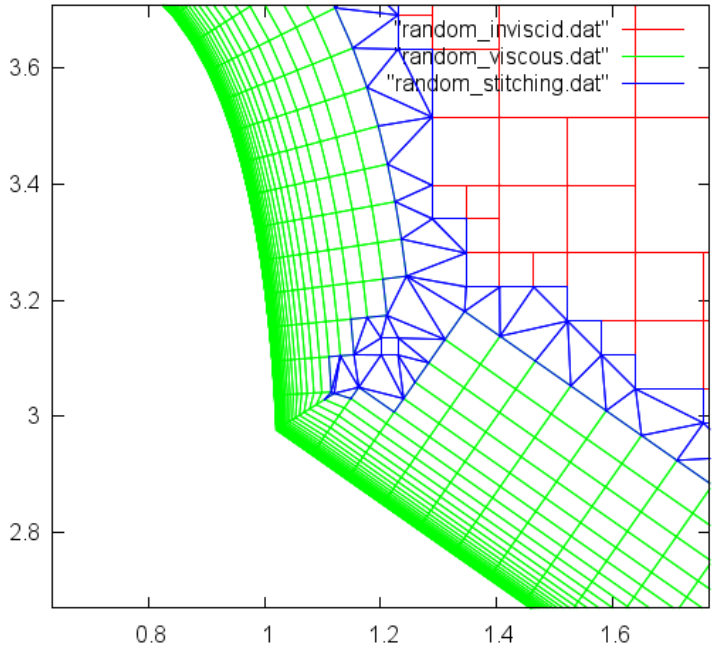


Figure 7.29 Random Complex Geometry

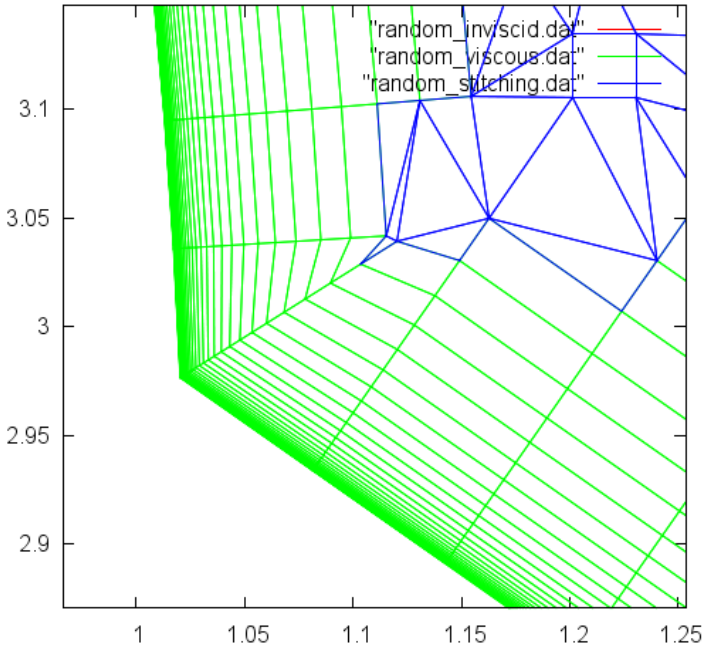


Figure 7.30 Random Complex Geometry

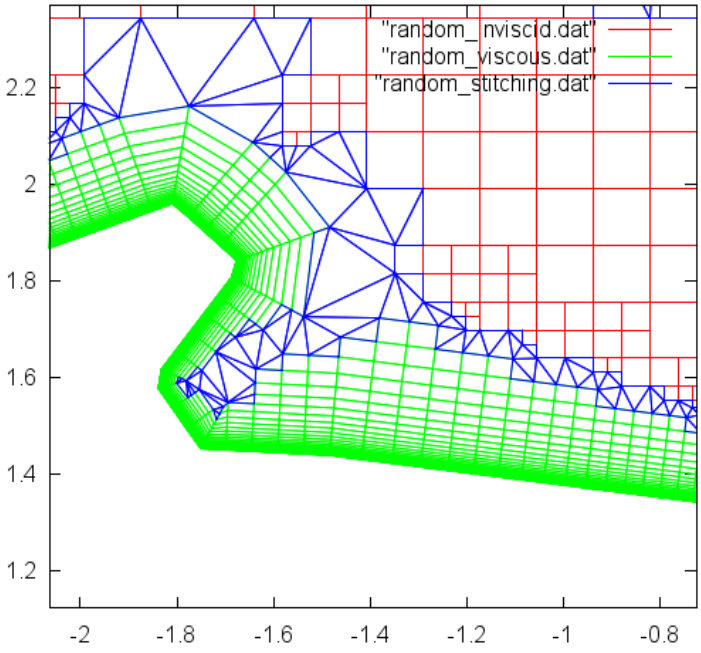


Figure 7.31 Random Complex Geometry

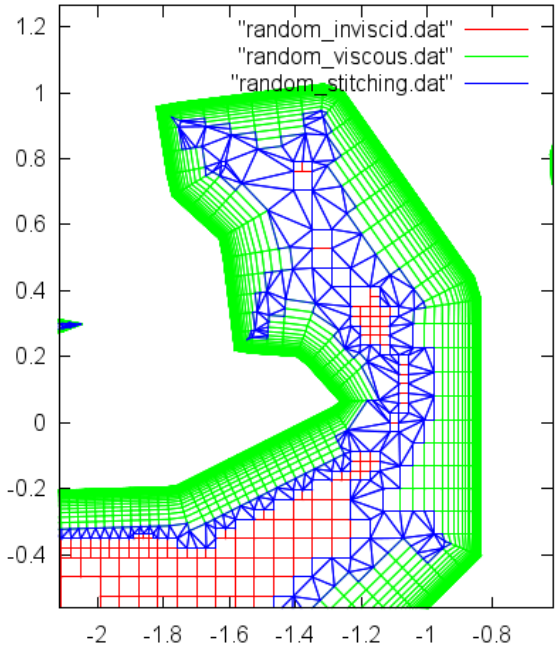


Figure 7.32 Random Complex Geometry

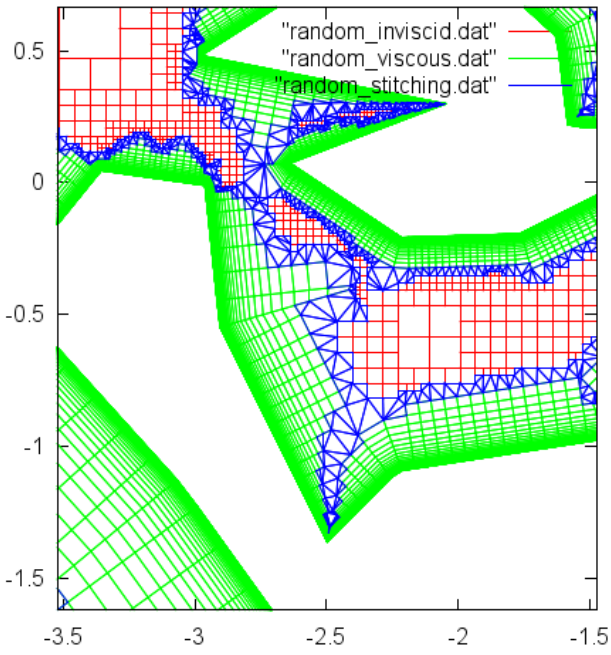


Figure 7.33 Random Complex Geometry

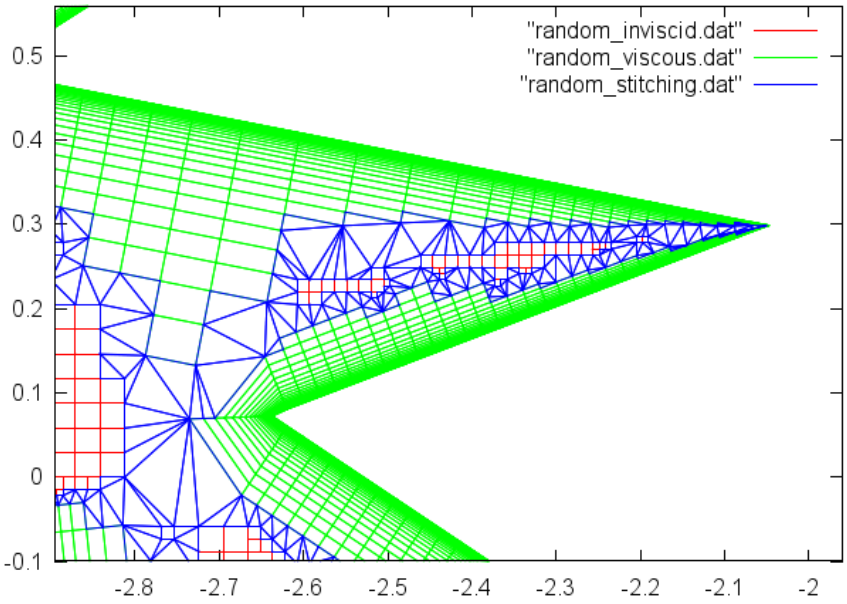


Figure 7.34 Random Complex Geometry

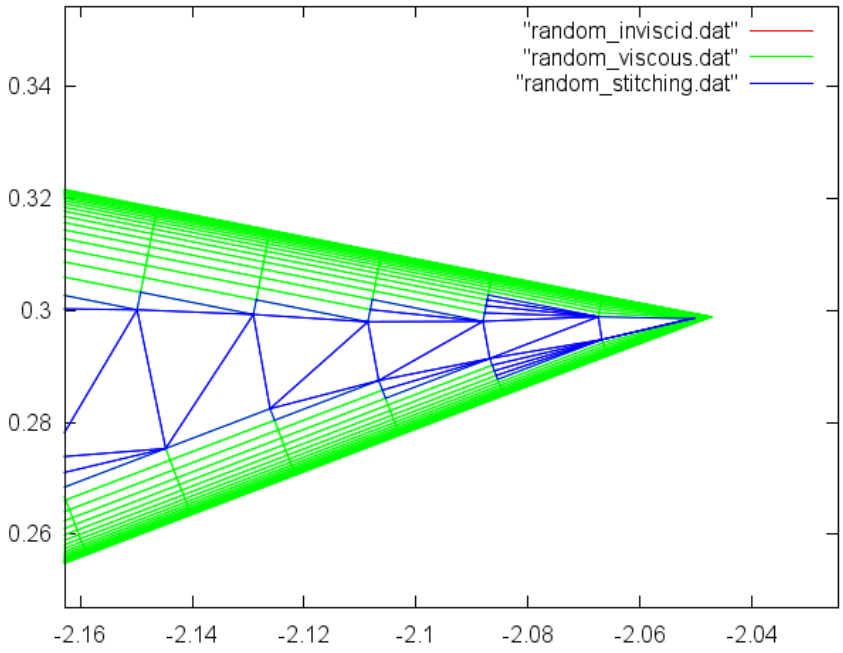


Figure 7.35 Random Complex Geometry

Table 7.6 Random Complex Geometry mesh metrics

	Stitching Region	Viscous Region
Number of grid points	2927	26244
Number of cells	2980	25227
Minimum area	8.387968e-7	3.234116e-7
Average area	6.251711e-3	5.717960e-4
Maximum area	7.878117e-2	2.852301e-2
Minimum angle	0.348303	0.057105
Average angle	60.000000	89.998979
Maximum angle	172.013826	179.881624
Minimum aspect ratio	1.000037	1.012385
Average aspect ratio	1.664087	97.864382
Maximum aspect ratio	618.960794	1612.924081
Minimum skew	1.006945	1.000000
Average skew	1.932967	1.003282
Maximum skew	26.839629	2.62686
Minimum condition number	1.000032	1.000050
Average condition number	1.424226	49.395522
Maximum condition number	95.162814	9240.481136
Minimum jacobian	0.006079	-0.782341
Maximum jacobian	1.000000	1.000000
Number of negative cells	0	0
Number of negative skewed cells	0	0
Number of positive skewed cells	0	7
Number of positive cells	2980	25220

Adaptive Capabilities

Figures 7.36 and 7.37 showcase background mesh adaptation based on the CFD solution for a NACA0012 test run. Note that the areas of high gradient are captured by the refinement, but that the minimum spacing of the grid was reached, which caused the spacing to be locally uniform. The proposed method's adaptive capabilities were also demonstrated in Chapter 3. Figure 7.38, visualized in VisIt [18], contains a psuedocolor plot of the density, on which the refinement was based.

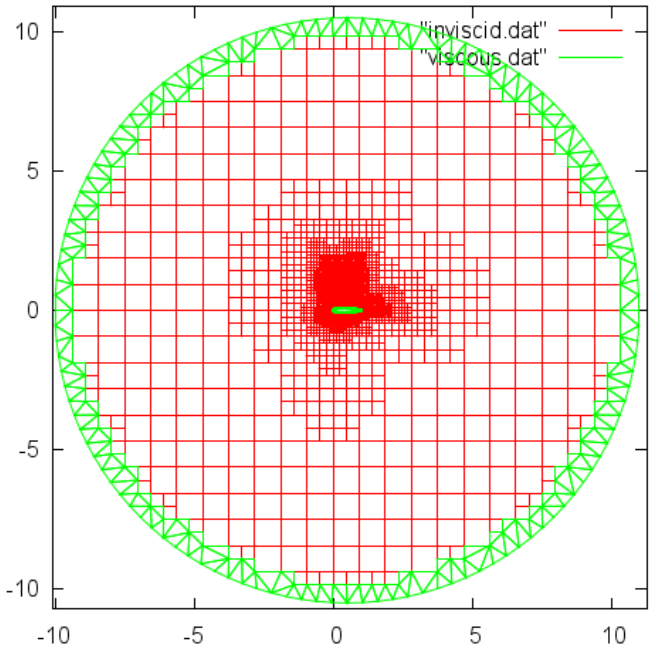


Figure 7.36 NACA0012 Solution Adaption: farfield view

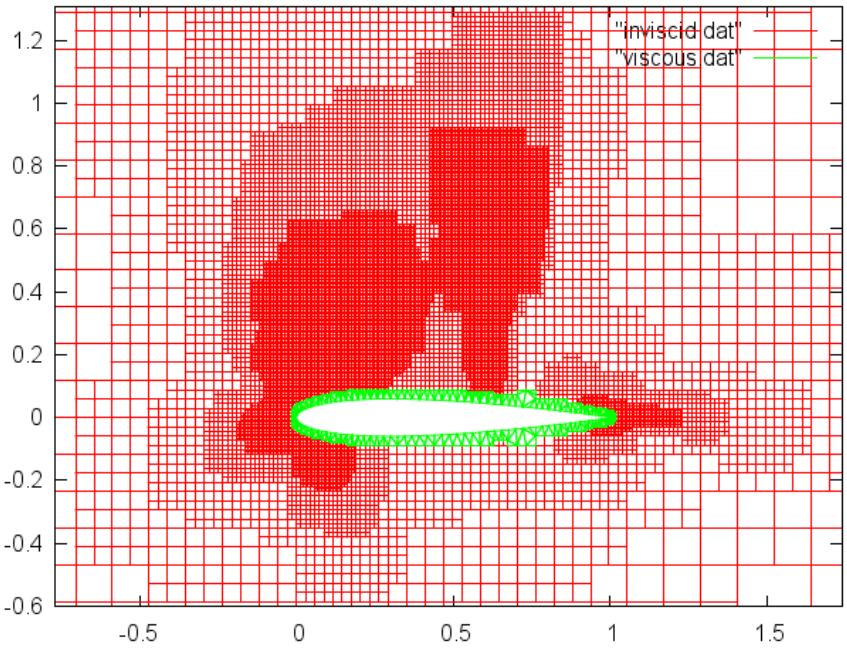


Figure 7.37 NACA0012 Solution Adaption: zoomed view

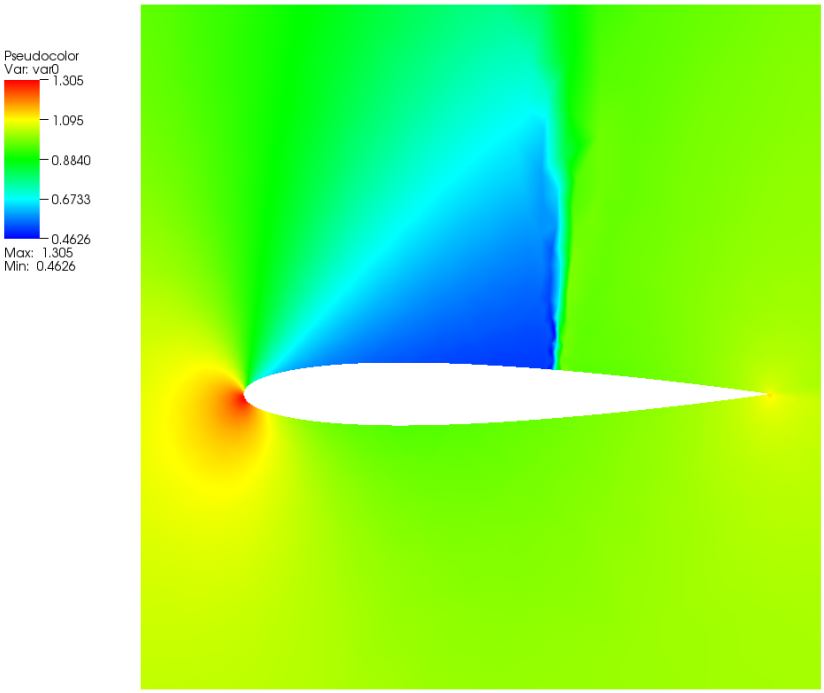


Figure 7.38 NACA0012 Solution Adaption: plot of density

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

Conclusions

To minimize the work done by the practitioner while retaining precise control over grid spacing, a hybrid mesh generation method for complex geometries was developed. The method consists of an anisotropic Cartesian method to generate a background mesh, an extrusion method to create viscous layers, and a Delaunay triangulation that reconciles the opposing fronts of the viscous and inviscid regions. The developed method has the ability to refine a background mesh based upon arbitrary criteria including input geometry, tensors generated from solution data, or spacing boxes. This is accomplished using a split-tree data structure that allows the creation of both unit and higher aspect ratio elements. The practitioner is only required to provide a closed geometry with appropriate spacing, but tools are made available for increased control when desired. Additional spacing requirements are optional and can be automatically created if used in conjunction with a solver and the spacing library developed by Collao [16].

The proposed method's ability to adapt to spacing criteria from a variety of sources was demonstrated in Chapter 3. The proposed method was demonstrated to be suitable for several different airfoil cases including the NACA0012, RAE2822, and the 30P30N multi-element airfoil. The method's ability to handle large numbers of geometries was showcased in a fuel cell scenario, and the robustness of the method, particularly the viscous layer production, was put to the ultimate test with a random, complex, multi-body geometry.

In each of the shown cases, full mesh metrics were provided as evidence of the method's successful creation of a valid mesh.

Future Work

Higher Quality Viscous Layer Production

Currently, the options are only to create skewed elements or to remove the skewed elements and leaving geometry segments exposed. While both of these provide valid meshes, certain applications may have specific cell quality requirements in those regions that could be assuaged by creating some additional functionality. In this case, splitting the poor quality elements in half when they surpass a tolerance has been shown by Thompson (cite Thompson) to provide better results.

Improving Quality of Stitching Layer

The quality of the elements in the stitching layer by increasing the size of the gap region and using a mature technology such as an advancing front method [12] to insert additional points.

Increased Gradation Control

The meshes generated by this method tend to be too coarse prior to adaptation, which could be assuaged by having greater control over cell size gradation across the mesh. One proposed option is to process the geometry multiple times. Each time a segment is processed, its extent box is increased to a defined size, but its tensor is adapted to relax the spacing requirements. The method will then have the ability to create concentric regions around the geometry of similar spacing.

Verifying Quad and Polygon Solutions

Currently, meshes from this method have only been used in conjunction with a solver that supports triangular meshes, but it has the ability to output mixed element meshes that contain triangles and quadrilaterals or triangles, quadrilaterals, and polygons. There is a solver currently being developed with the intention of using polygonal meshes, but it has yet to be verified via Method of Manufactured solutions. Once the solver has been verified, solutions for meshes with higher order elements and lower element counts will be compared to those of triangular and mixed triangle/quad meshes. The polygonal meshes are hypothesized to reach equivalent solutions with lower overall element counts and less work by the solver.

REFERENCES

[1] Inc., P., "www.pointwise.com," 2011. 2

[2] Jr., S. L. K., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," *AIAA*, January 1995. 2

[3] Pember, R. B., Bell, J. B., Colella, P., Crutchfield, W. Y., and Welcome, M. W., "Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow," *AIAA*, June 1991. 2

[4] Landsberg, A. and Boris, J., "An Efficient Method for Solving Flows around Complex Bodies," *NRL Review*, 1993. 2

[5] Aftmosis, M. J. and Berger, M. J., "Aspects (and Aspect Ratios) of Cartesian Mesh Methods," *International Conference on Numberical Methods in Fluid Mechanics*, 1998. 2

[6] Coirier, W. J. and Powell, K. G., "Solution Adaptive Cartesian Cell Approach for Viscous and Inviscid Flows," *AIAA*, Vol. 34, No. 5, May 1996. 2

[7] Domel, N. and Jr., S. L. K., "SPLITFLOW: Progress in 3D CFD with Cartesian Omni-tree GRids for Complex Geometries," *AIAA*, January 2000. 3

[8] Jr., S. L. K., "Hierarchical Unstructured Mesh Generation," *AIAA*, January 2004. 3

[9] Wang, Z. J. and Chen, R. F., "Anisotropic Solution-Adaptive Viscous Cartesian Grid Method for Turbulent Flow Simulation," *AIAA*, Vol. 40, No. 10, October 2002. 3

[10] Thompson, D. and Chalasani, S., "Quality Improvements in Extruded Meshes Using Topologically Adaptive Generalized Elements," *International Journal for Numerical Methods in Engineering*, Vol. 60, 2004, pp. 1139–1159. 3, 41

[11] Dawes, W. N., Harvey, S. A., Fellows, S., Favaretto, C. F., and Vewlivelli, A., "Viscous Layer Meshes from Level Sets on Cartesian Meshes," *AIAA*, Vol. 45, January 2007. 3

[12] Lohner, R. and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method," *AIAA*, Vol. 26, January 1988. 4, 94

[13] Liu, C. and Hwang, C., "A New Strategy for Unstructured Mesh Generation," *AIAA*, June 2000. 4

[14] Lohner, R., Luo, H., and Spiegel, S., "Hybrid Grid Generation Method for Complex Geometries," *AIAA*, Vol. 48, No. 11, November 2010, pp. 2639–2646. 4, 8

[15] Betro, V. C., *Fully Anisotropic Split-Tree Adaptive Refinement Mesh Generation Using Tetrahedral Mesh Stitching*, Ph.D. thesis, University of Tennessee, Chattanooga, August 2010. 4, 43

[16] Collao, M. D., *Generation and Optimization of Spacing Fields*, Master's thesis, University of Tennessee, Chattanooga, 2011. 8, 18, 93

[17] "wikipedia.org/wiki/Voxel," 2011. 27

[18] DOE, "https://wci.llnl.gov/codes/visit," 2011. 90

VITA

Cameron was born in May of 1986 in Port Clinton Ohio. He earned a B.S. in Phyics at Austin Peay State University (Let's Go Peay!), where he developed a strong interest in computer science, mathematics, and physics. He then continued his studies at the SimCenter, where he could persue all three.