COMPARISON OF TWO METHODS FOR TWO DIMENSIONAL UNSTRUCTURED MESH ADAPTATION WITH ELLIPTIC SMOOTHING

By

Matthew David O'Connell

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 Betro
Research Assistant Professor of Computational Engineering
(Committee Member)

Jerald Ainsworth
Dean of the Graduate School

COMPARISON OF TWO METHODS FOR TWO DIMENSIONAL UNSTRUCTURED MESH ADAPTATION WITH ELLIPTIC SMOOTHING

By

Matthew David O'Connell

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

A new mesh adaptation method for unstructured grids is presented. The technique uses virtual control volumes that are iteratively manipulated to conform the mesh to match either a Riemannian metric tensor field or an equal distribution of scalar weights. Forcing functions similar to those used for structured grids are employed such that the resulting meshes can be compared with those generated using the new adaptation method. Several test cases using analytic functions to drive the mesh adaptation are also presented and compared with the new method. Mesh adaptation results driven by computed flow field information are also compared to those adapted using analytical functions as well as those adapted using the virtual control volume approach. Strengths and weaknesses of the various adaptation methods are investigated, and suggestions for further research are discussed.

DEDICATION

This thesis is dedicated to my family and friends, without whom I would not have been able to complete this work.

ACKNOWLEDGEMENTS

I would like to thank Dr. Steve Karman for his guidance and patience during the research and development of this document. Also, I would like to thank David Collao for the use of his 2D spacing library and for his time helping me learn to use it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

α, β, γ, Winslow coefficients

$\Delta\vec{r}$, perturbation vector

Γ, integral length of an edge

λ, Riemannian scaling matrix

∇, differential operator

Ω, integral area

Φ, Ψ, forcing functions

ξ, η, computational coordinates

A Area of a control volume

a, b adaptation coefficient

C, equidistribution value

e_1, e_2 principal directions

f, adaptation function

h_1, h_2 spacing along principal directions

J, Jacobian of transformation

l, metric edge length

M, Riemannian metric tensor

m, Winslow weight

n, t, normal vectors

P, Q, forcing functions

R, Riemannian rotation matrix

s, central point relaxation factor

u, forcing function scaling factor

w, weight function

x, y physical coordinates

CHAPTER 1
INTRODUCTION

Importance

Adapting a grid and increasing point density at areas with high solution gradients can be desirable when numerically solving partial differential equations. In the typical adaptation cycle, a mesh by analysis codes to numerically solve the problem. That solution can then be used to modify or adapt the mesh to resolve more detail in choice areas producing, hopefully, a more refined result. The modified mesh is used to run the next step in the simulation. This cycle requires not only robust and accurate simulation code, but also a way to adapt the mesh.

There is still limited control in unstructured meshing to adapt an already generated mesh to change spacing in a desired region of the mesh. Refinement or coarsening methods where edges or cells in a mesh are subdivided or removed have been used in the past, but often they leave a mesh with poor element quality that can cause instability in computational fluid dynamics (CFD) analysis codes. To avoid this once a mesh has undergone refinement, it must then undergo optimization to increase element quality. Since the mesh smoothing is conducted with its only goal being to increase element quality, the increased resolution gained through refinement is often smeared or sometimes moved to surrounding regions of the mesh. Obtaining a suitable mesh can require more cycles of refinement to increase resolution required and can increase the computational resources demanded of the simulation code. A more elegant approach can combine the goals of both of these steps into one step, coupling the desire for good quality elements and control over resolution.

Adaptive point distribution during elliptic mesh generation has been used for decades, with notable early contributions from Winslow [1] and Anderson [2] who each used forcing functions to obtain adaptation. Anderson expanded on earlier work by Thompson [3] and Dwyer[4] to suggest a form of forcing functions for the Winslow equations that resulted in high quality meshes adapted to solution data.

Until recently the success of elliptic smoothing has been limited in unstructured grid generation due to the lack of an implied computational domain. Knupp has shown unstructured Winslow mesh smoothing on unstructured quadrilateral meshes using a locally defined computational domain[5]. Solution adaptation with Anderson-style forcing functions has been presented by Karman on unstructured meshes using an initial physical mesh as the computational mesh. Sahasrabudhe has had great success in unstructured smoothing by creating local optimized computational domains called "virtual control volumes"[6]. Masters then showed that iterative manipulation of virtual control volumes could be used to create viscous layers [7].

This document will compare two methods for mesh control on unstructured meshes that preserve good element quality. The first method replaces Winslow forcing functions with functions introduced by Anderson in structured meshes. The second is a novel method for solution based mesh adaptation by manipulating the computational domain to gain control over grid spacing. It is the comparison of these two methods that is of particular interest, as their strengths and weaknesses may drive the direction of future research.

In the following sections, an overview of elliptic mesh smoothing on unstructured two-dimensional meshes using virtual control volumes is presented, as well as a review of generation of solution gradient based forcing functions. The idea of virtual control volume manipulation is reintroduced with the goal of adapting an unstructured mesh to match a desired spacing. This is done by iteratively adapting virtual control volumes to achieve equal weights across stencil edges with the weights determined by a stationary scalar field.

Results are compared between adaptive meshes as solutions to Winslow equations with forcing functions and Winslow equations with iteratively adapted virtual control volumes.

Chapter Summaries

In Chapter 2, Winslow smoothing is discussed. The Winslow equations are introduced and a disjoint computational domain is described. The movement of boundary nodes within a Winslow solver is also described. Finally, the basic algorithm for a Winslow solver is shown.

In Chapter 3, forcing functions for the Winslow equations are reintroduced with the goal of achieving solution adaptation on unstructured meshes. Additional restraints on the computational domain required for forcing functions are discussed and a modified algorithm for a Winslow solver with forcing functions is shown.

In Chapter 4, the procedure for mesh adaptation by manipulating local computational domains, a method named central point offset, is described. The scheme to describe spacing using Riemannian metric tensors is introduced. The iterative nature of the central point offset method is discussed as are some thoughts on the local robustness of the scheme.

In Chapter 5, results of adaptation using forcing function adaptation and adaptation obtained through central point offset are presented. Some exploration is made on effects of scaling parameters present in both methods to compare the level of control each method allows. The performance of the two methods are compared on more complex problems using flow field information from an Euler solver.

Chapter 6 is a summary of the benefits and pitfalls of both adaptation methods as well as some final thoughts on Winslow smoothing and thoughts on further research using these methods.

CHAPTER 2
BACKGROUND

Winslow Smoothing

All adaptation methods explored in this document are modifications on Winslow smoothing for unstructured meshes. The Winslow equations were described by Thompson et al. [8] with forcing functions P, Q as

$$\begin{aligned} \nabla^2 \xi &= P \\ \nabla^2 \eta &= Q \end{aligned} \tag{2.1}$$

or in Φ, Ψ form

$$\begin{aligned} \nabla^2 \xi &= (\nabla \xi \cdot \nabla \xi)\Phi \\ \nabla^2 \eta &= (\nabla \eta \cdot \nabla \eta)\Psi \end{aligned} \tag{2.2}$$

The Winslow equations describe a smooth variation of the computational coordinates (ξ, η) with respect to the physical coordinates (x, y). However, it is the computational domain that is known and the physical location of the points that is desired to increase mesh quality. Therefore, the equations must be transformed onto the physical domain. A complete description of this process is given in Appendix A and the Winslow equations for solving physical locations x, y are

$$\alpha(x_{\xi\xi} + \Phi x_\xi) - 2\beta x_{\xi\eta} + \gamma(x_{\eta\eta} + \Psi x_\eta) = 0$$
$$\alpha(y_{\xi\xi} + \Phi y_\xi) - 2\beta y_{\xi\eta} + \gamma(y_{\eta\eta} + \Psi y_\eta) = 0 \tag{2.3}$$

with

$$\begin{aligned} \alpha &= x_\eta^2 + y_\eta^2 \\ \beta &= x_\xi x_\eta + y_\xi y_\eta \\ \gamma &= x_\xi^2 + y_\xi^2 \end{aligned} \tag{2.4}$$

By solving these equations numerically new locations for the physical points can be obtained that satisfy the smoothness condition and yield higher quality elements more suitable for use in simulation codes. Below is an explanation of the discretization of this system using a finite volume methodology. Later the concept of the virtual control volume, a disjoint and locally optimized computational domain, is reintroduced. This chapter closes by addressing the topic of boundary node movement and a summary of the anatomy of a Winslow solver.

Finite Volume Discretization of the Winslow Equations

To numerically solve the Winslow equations, the system must first be discretized into a linear system. The chosen method of discretization has been a finite volume formulation. In the finite volume formulation it is necessary to analyze derivatives on a stencil or control volume.

A finite volume discretization of derivatives on a stencil can be done by exploiting Gauss' divergence theorem. Given some scalar function f on the element, we can treat f as a vector function in two dimensions.

$$\vec{f} =< f, 0 >$$

Then by the use of the divergence theorem on a control volume*:

$$\iint_{\Omega} \nabla f \partial\Omega = \oint_{\Gamma} \vec{f} \cdot \hat{n} \partial\Gamma.$$

$$\frac{\partial f}{\partial x} = \sum_{i}^{e} f_i \hat{n}_{x_i} \Gamma_i \Omega^{-1} \tag{2.5}$$

where e is the number of edges, Γ the length of an outside edge and Ω the area of the control volume. The normal vector point out of the control volume is $\vec{n}$. And derivatives $\frac{\partial}{\partial y}$ done similarly,

$$\vec{f} =< 0, f >$$

$$\frac{\partial f}{\partial y} = \sum_{i}^{e} f_i \hat{n}_{y_i} \Gamma_i \Omega^{-1}$$

For an implementation on a triangle, such as the one shown in Figure 2.1. the derivatives

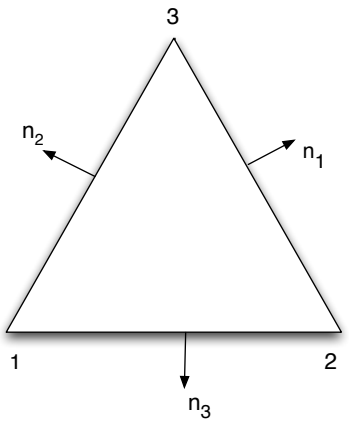


Figure 2.1 Triangle and naming conventions.

*Where the subscripts x and y denote components of the vector in the x and y directions and should not be confused with partial derivatives.

can be computed as

$$\frac{\partial f}{\partial x} = \frac{1}{2A}(f_1(n_{2x} + n_{3x}) + f_2(n_{1x} + n_{3x}) + f_3(n_{1x} + n_{2x}))$$
$$\frac{\partial f}{\partial y} = \frac{1}{2A}(f_1(n_{2y} + n_{3y}) + f_2(n_{1x} + n_{3y}) + f_3(n_{1x} + n_{2y}))$$

Noting that:

$$0 = n_{1x} + n_{2x} + n_{3x}$$

$$0 = n_{1y} + n_{2y} + n_{3y}$$

and from this substitutions can be made in the form

$$-n_{1x} = n_{2x} + n_{3x}$$

$$-n_{1y} = n_{2y} + n_{3y}$$

The derivative of the function f over the triangle is given by

$$\frac{\partial f}{\partial x} = \frac{1}{2A}(-f_1 n_{1x} - f_2 n_{2x} - f_3 n_{3x})$$
$$\frac{\partial f}{\partial y} = \frac{1}{2A}(-f_1 n_{1y} - f_2 n_{2y} - f_3 n_{3y})$$

Winslow Discretization

Armed with a method for computing derivatives on an unstructured stencil, the method can proceed to the system of equations. Starting with the integral form of the Winslow equations, second derivatives are replaced by first derivatives applied to the terms above for derivatives $x_\xi, x_\eta, y_\xi, y_\eta$ of the form in shown in equation 2.6.

$$\iint_\Omega \alpha(x_{\xi\xi} + \Phi x_\xi) - 2\beta(x_{\xi\eta}) + \gamma(x_{\eta\eta} + \Psi x_\eta)\partial\Omega = 0$$
$$\iint_\Omega \alpha(y_{\xi\xi} + \Phi y_\xi) - 2\beta(y_{\xi\eta}) + \gamma(y_{\eta\eta} + \Psi y_\eta)\partial\Omega = 0. \tag{2.6}$$

After making these substitutions, the Winslow equations are

$$\begin{aligned}
\iint_\Omega \quad & \tfrac{1}{2A}\alpha(\tfrac{\partial}{\partial\xi}(-x_1n_{1\xi}-x_2n_{2\xi}-x_3n_{3\xi})+\Phi x_\xi) \\
& -\tfrac{1}{A}\beta(\tfrac{\partial}{\partial\eta}(-x_1n_{1\xi}-x_2n_{2\xi}-x_3n_{3\xi})) \\
& +\tfrac{1}{2A}\gamma(\tfrac{\partial}{\partial\eta}(-x_1n_{1\eta}-x_2n_{2\eta}-x_3n_{3\eta})+\Psi x_\eta)\partial\Omega = 0 \\
\iint_\Omega \quad & \tfrac{1}{2A}\alpha(\tfrac{\partial}{\partial\xi}(-y_1n_{1\xi}-y_2n_{2\xi}-y_3n_{3\xi})+\Phi x_\xi) \\
& -\tfrac{1}{A}\beta(\tfrac{\partial}{\partial\eta}(-y_1n_{1\xi}-y_2n_{2\xi}-y_3n_{3\xi})) \\
& +\tfrac{1}{2A}\gamma(\tfrac{\partial}{\partial\eta}(-y_1n_{1\eta}-y_2n_{2\eta}-y_3n_{3\eta})+\Psi y_\eta)\partial\Omega = 0
\end{aligned}$$

Finally, using the divergence theorem once again, the finite volume discretization of the Winslow equations on a stencil of triangles.

$$\begin{aligned}
\alpha(\frac{1}{2A}(-x_1n_{1\xi}-x_2n_{2\xi}-x_3n_{3\xi})t_\xi &+ (\frac{x_2+x_3}{2})\Phi t_\xi) \\
\beta(\frac{1}{A}(-x_1n_{1\xi}-x_2n_{2\xi}-x_3n_{3\xi})t_\eta &+ \\
\gamma(\frac{1}{2A}(-x_1n_{1\eta}-x_2n_{2\eta}-x_3n_{3\eta})t_\eta &+ (\frac{x_2+x_3}{2})\Psi t_\eta) = 0 \\
\alpha(\frac{1}{2A}(-y_1n_{1\xi}-y_2n_{2\xi}-y_3n_{3\xi})t_\xi &+ (\frac{y_2+y_3}{2})\Phi t_\xi) \\
\beta(\frac{1}{A}(-y_1n_{1\xi}-y_2n_{2\xi}-y_3n_{3\xi})t_\eta &+ \\
\gamma(\frac{1}{2A}(-y_1n_{1\eta}-y_2n_{2\eta}-y_3n_{3\eta})t_\eta &+ (\frac{y_2+y_3}{2})\Psi t_\eta) = 0
\end{aligned} \tag{2.7}$$

It should be noted that when applying the divergence theorem on the entire stencil, only the outer most integration path contributes. All integration paths along triangle edges that are shared within the stencil cancel. For this reason, only the outward facing edge vector t needs to be used.

At each stencil, these equations are valid using the locally defined node numbers; however, the goal is to find physical locations of nodes that satisfy the Winslow equations globally.

To that end equation 2.7 can be rearranged to the form

$$\begin{aligned} m_1 x_1 + m_2 x_2 + m_3 x_3 &= 0 \\ m_1 y_1 + m_2 y_2 + m_3 y_3 &= 0 \end{aligned} \tag{2.8}$$

with weights

$$\begin{aligned} m_1 &= \frac{1}{2A}\Big[-\alpha n_{1\xi} t_\xi + 2\beta n_{1\eta} t_\xi - \gamma n_{1\eta} t_\eta \Big] \\ m_2 &= \frac{1}{2A}\Big[-\alpha n_{2\xi} t_\xi + 2\beta n_{2\eta} t_\xi - \gamma n_{2\eta} t_\eta \Big] + \frac{1}{2}(\alpha\Phi t_\xi + \gamma\Psi t_\eta) \\ m_3 &= \frac{1}{2A}\Big[-\alpha n_{3\xi} t_\xi + 2\beta n_{3\eta} t_\xi - \gamma n_{3\eta} t_\eta \Big] + \frac{1}{2}(\alpha\Phi t_\xi + \gamma\Psi t_\eta) \end{aligned} \tag{2.9}$$

which solves for the physical location of (x_1, y_1). When solving this system globally, the coefficients need to be periodically updated to recompute α, β and γ since they are also functions of the physical locations and constitute a non-linearity. To build and solve the global system it may help to look at a an example mesh, such as that in figure 2.2. The four cells that make up the stencil for the center node are labeled. To build the row of the linear system for this central node, the weights from cells, 1 through 4, would have to be computed then summed into the columns corresponding the node that each weight affects. Since the central node appears in every cell, there will be more weights on the diagonal entry of a row than any off diagonal entry which will only have a maximum of 4 weights, two from each triangle shared with the center. When forcing functions Φ and Ψ are small or zero, as the case in general smoothing where adaptation is not the goal, this distribution of weights allows for a very well conditioned linear system that is well suited for a point-iterative linear solver. Before that can happen though, there must be a valid computational domain.

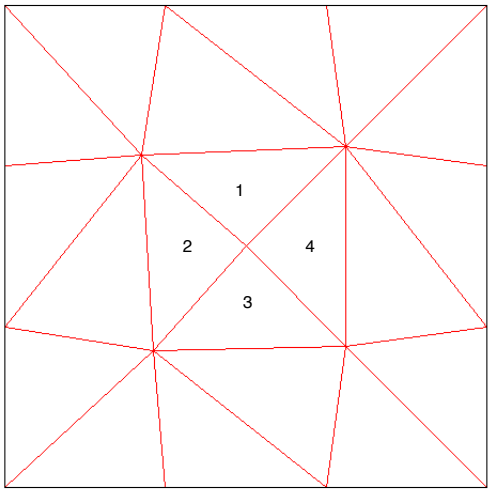


Figure 2.2 A small test mesh

Computational Domain

The Winslow equations enforce a smoothness condition between the spacing in the computational domain (ξ, η) and the physical (x, y). In structured meshes, the computational domain is a simple matter, a mesh containing the same topology but ideal spacing. For an unstructured mesh there is not a universal computational domain that matches every unstructured topology; therefore, for each unstructured mesh a computational domain must be generated that matches the topology of the physical mesh. One obvious way of obtaining a computational mesh would be to simply copy the existing physical mesh. However, if the two meshes physical and computational were identical to start the smoothness condition would be satisfied, almost by definition, and no points would be moved. The resulting mesh, being identical to the initial mesh, would not have improved quality. The key to optimizing a mesh through the use of the Winslow equations lies in using an ideal computational domain.

Disjoint Computational Domain

Simply using a copy of the physical mesh as a computational mesh will not increase mesh quality, but if an ideal computational mesh matching the physical mesh were known then there would be no need to smooth, simply use the computational mesh as the physical mesh. This circular problem can be solved by Sahasrabudhe with the introduction of ideal computational domains that are only defined locally [9]. For a stencil of elements surrounding a single node an ideal mesh can be defined by placing all connected points on the unit circle and equal distance apart. These stencils, called virtual control volumes, are locally defined and are sufficient to use as an optimal computational mesh that can be used to drive the solution of the Winslow equations.

The location of points for a virtual control volume can be computed by

$$\begin{aligned}
\text{if nt} = 0 \qquad & \theta_q = \tfrac{2\pi}{nq} \\
\text{if nq} = 0 \qquad & \theta_t = \tfrac{2\pi}{nt} \\
\text{if nq} = 1 \text{ and nt} > 1 \quad & \theta_t = \tfrac{3\pi}{2nt}, \theta_q = \tfrac{\pi}{2} \\
\text{if nq} > 1 \text{ and nt} = 1 \quad & \theta_t = \tfrac{\pi}{2}, \theta_q = \tfrac{3\pi}{2nq} \\
\text{if nq} > 1 \text{ and nt} > 1 \quad & \theta_t = \tfrac{\pi}{nt}, \theta_q = \tfrac{\pi}{nq}
\end{aligned} \tag{2.10}$$

The bottom of Fig. (2.3) shows three example virtual control volumes for three stencils with different arrangement of elements. The top meshes are example physical meshes. Notice that all triangles are used in the virtual control volumes. Sahasrabudhe also showed that the orientation of the virtual control volumes does not affect Laplacian Winslow smoothing[9]. The node of a quadrilateral that is not directly connected to the central node is cut out of the stencil. Since only directly connected edges are in the computational stencil the entire stencil consists of only triangle elements. The virtual control volumes are optimized for each node's particular stencil and is not required to agree globally, and indeed will not in all but

an ideal case. This set of virtual control volumes can be quickly generated and stored since it will not change unless the topology of the mesh changes. The ideal spacing and angles of the virtual control volume allow for a computational domain that can be used to untangle or optimize a mesh.

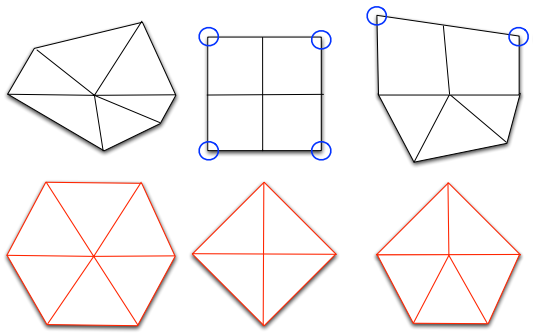


Figure 2.3 Example physical and computational control volumes

Boundary Node Movement

Smoothing by use of the Winslow equations is a boundary value problem. Dirichlet boundary conditions can be employed by simply not allowing boundary movement, taking the initial node distribution on the boundaries. But it can become necessary to allow boundary movement especially in adaptation where the spacing of nodes on a boundary may become very important either to capture rapidly changing phenomena across a boundary, or to allow correct spacing on interior points close to the boundary. To solve the Winslow equations on the boundary, each boundary node must have a closed control volume in both the physical and computational domains.

Consider node 2 on the small test mesh shown in figure 2.4 with adjacent cells 1, 2 and 3 and neighboring nodes 1, 3, 5 and 6. To complete the computational stencil, two ghost cells and a single ghost node are inserted between cells 1 and 3 before the virtual control volume is created. For the purposes of node distribution for the computational stencil, the

ghost cells are treated as quadrilaterals. The computational stencil for node 2 is shown in 2.5 where the influence of the two inserted ghost cells can be seen as the two larger triangle to the left of the central node. As can be seen here it is not necessary to align the virtual control volumes for general smoothing.

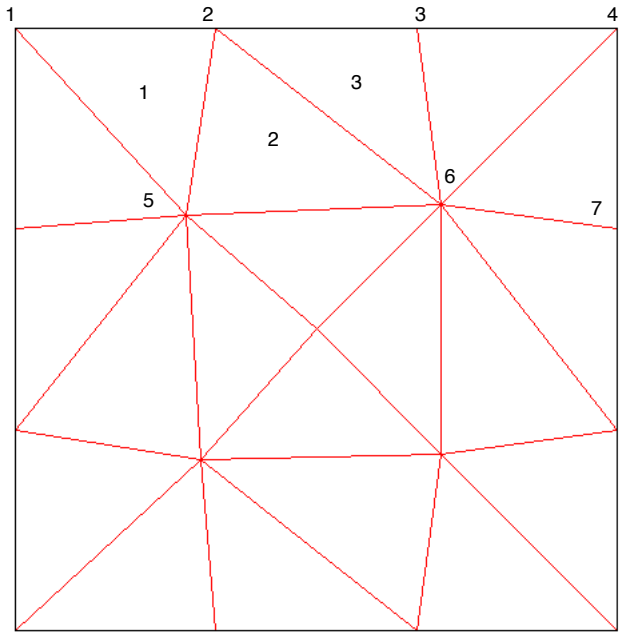


Figure 2.4 Boundary test mesh

To close the physical stencil, it is important to start by noting that by only adding one additional point to the virtual control volume exactly one point must be added in the physical stencil. Furthermore, since all gradient calculations for the Winslow weights are computed with respect to the computational domain, the type of cells inserted into the physical mesh makes no difference. The only requirement that is important is the location (x, y) of this ghost point.

There are a myriad of locations that can be chosen for the ghost point and the only restriction is that the ghost point must allow for good quality cells on the boundary. Only one scheme is used to compute the ghost point location throughout this document.

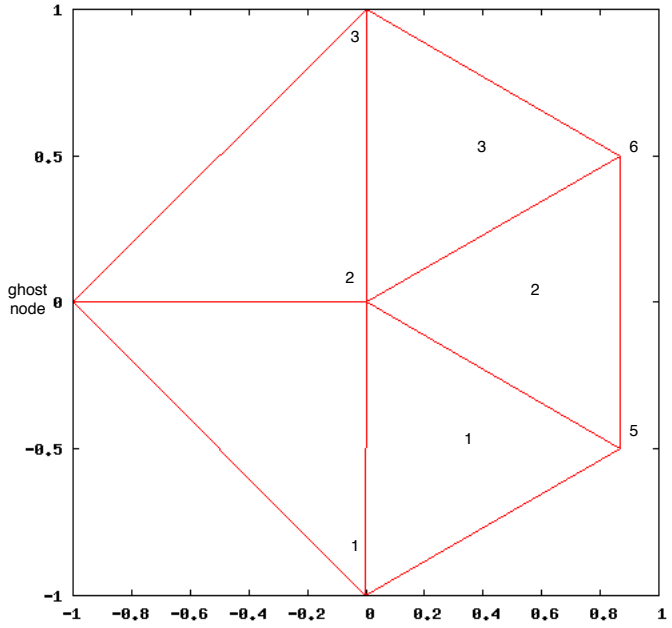


Figure 2.5 Completed boundary virtual control volume

The scheme computes the average distance to all the strictly interior nodes directly connected to the boundary node. The ghost point is then placed that average distance away perpendicular to the boundary. If a boundary node has no directly connected interior nodes, such as on a corner in a triangular mesh, then the average distance to the adjacent boundary nodes is used instead. On boundaries with curvature, the perpendicular direction is used by creating a straight line between the two adjacent boundary nodes and computing a perpendicular direction from that line. Figure 2.6 shows a topology with both cases.

When allowing boundary points to move the Winslow equations will often move boundary nodes off their boundaries. This type of boundary motion is clearly unacceptable. To allow boundary movement while preserving boundaries, a moved boundary node must be projected back onto the boundary during each point iterative solver step. This requires description of the boundary that is highly accurate and does not move during mesh smoothing. The Winslow solver written during this research made use of a C++ geometry "class" developed

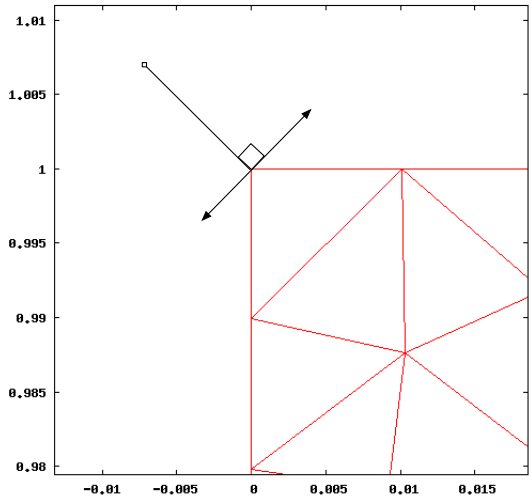


Figure 2.6 Ghost node location for corner topology

by Dr. Steve Karman Jr of the University of Tennessee at Chattanooga (UTC). This class stores a discrete representation of a boundary accurate to user specification read from a Pointwise segment file [10]. Boundary nodes are associated with their specific boundary and are placed back onto that boundary in the nearest location to that of the smoothed location. Points that cannot be moved under any circumstance are labeled as critical points. These points occur by default at the intersection of boundaries but can also be user defined.

With all steps described, the procedure for Laplacian Winslow smoothing is described below with a flowchart in figure 2.7.

```
Read in mesh
Build virtual control volumes
for i ≤ max iterations do
    Compute ghost node locations
    Compute Winslow weights
    Build Global linear system
    for j ≤ point iterative max do
        Point iterative step
        Snap boundary nodes back onto boundary using the geometry class
    end for
end for
```

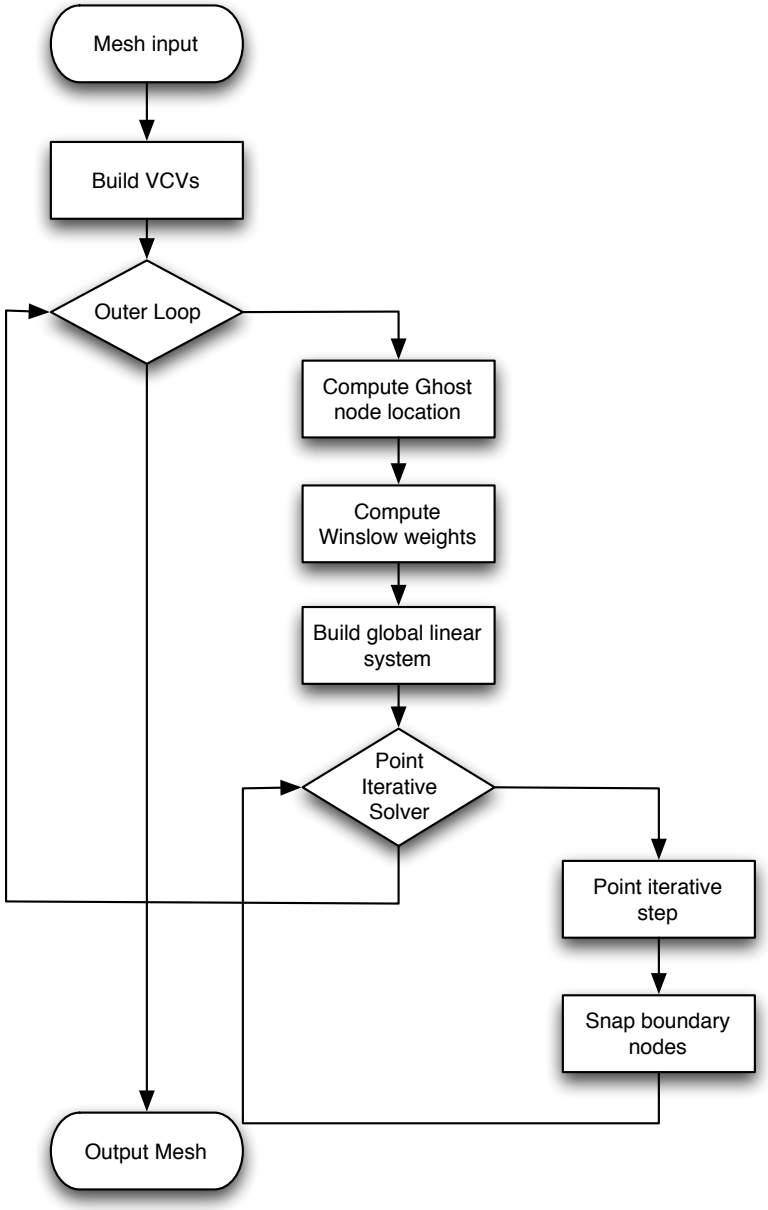


Figure 2.7 Winslow flowchart.

CHAPTER 3

ADAPTION BY FORCING FUNCTIONS

In both structured and unstructured mesh generation, non-zero functions for Φ and Ψ have been used to achieve mesh adaptation to increase point density where scalar functions rapidly change. Anderson showed that this solution adaptation could be achieved by using weights given by[2]:

$$\begin{aligned} \Phi &= \frac{1}{w_1}\frac{\partial w_1}{\partial \xi} \\ \Psi &= \frac{1}{w_2}\frac{\partial w_2}{\partial \eta}. \end{aligned} \tag{3.1}$$

$$\begin{aligned} w_1 &= 1 + a\Big(\frac{\partial f}{\partial \xi}\Big)^2 \\ w_2 &= 1 + b\Big(\frac{\partial f}{\partial \eta}\Big)^2 \end{aligned} \tag{3.2}$$

In this way, weights are guaranteed to be positive and will be large in areas of high gradients in the adaptation variable f. In either structured or unstructured meshes, using a globally defined computational domain these functions can be used with little modification. However, since the calculation of weights require gradients of the adaptation function with respect to the computational domain, one needs to have a consistent computational domain. To achieve some consistency, the VCV can be rotated so that at least one edge is guaranteed to have the same orientation in both the computational and physical domains. This local mapping is still not globally valid, but it does allow for some level of consistency between each of the virtual control volumes. Furthermore, as the mesh is smoothed by a Winslow solver,

the inconsistencies will lessen as the physical mesh begins to mimic angles described in the computational mesh.

When solving the Winslow equations, the forcing functions Φ and Ψ contribute to weights corresponding to the external nodes in a stencil. These external nodes make up the off-diagonal entries in the global linear system. For stability these weights cannot grow relatively large, else they will weaken the diagonal dominance of the system. For this reason an additional relaxation parameter u is added, making the forcing functions

$$\begin{aligned} \Phi &= u\left[\frac{1}{w_1}\frac{\partial w_1}{\partial \xi}\right] \\ \Psi &= u\left[\frac{1}{w_2}\frac{\partial w_2}{\partial \eta}\right] \end{aligned} \tag{3.3}$$

This factor is user defined and can scale up or down the size of the forcing functions. In Chapter 6, examples are shown where u is used to scale forcing functions.

Backgrounding Adaptation Function

Mesh adaptation using these forcing functions completely relies on the adaptation function. When adapting a mesh using solution field data from a numerical solver, the data is typically connected to nodes or cells. Using this storage in adaptation creates a problem; when nodes move, the solution field data associated with the nodes should not move. The data must be separated from the nodes and frozen to the physical domain. Storing data to the physical domain was done using a C++ class, Quadtree, developed by Dr. Karman of UTC with modifications added by David Collao to allow for no overlap domains. The Quadtree class takes data and an extent box in 2D space that the data is located. The data is stored in a tree data structure with the root node representing the entire domain. Subsequent children split the remaining domain into quarters. This sorted tree allows for relatively fast retrieval of data across a 2D domain. Solution data is stored to the x, y plane

as data along with the dualed cell that that node represents, before any adaptation occurs. When adaptation function values are required during the calculation of the forcing function the new location of the node is sent as a query to the Quadtree and data pertaining to that region of the mesh is retrieved.

Algorithm

Adding mesh adaptation using forcing functions requires only small changes in a Winslow solver. Below is a basic outline for a Winslow solver with forcing functions and figure 3.1 shows a flowchart for the method.

1: Read in mesh
2: Build Virtual control volumes
3: Store adaptation function in the background Quadtree storage
4: **for** $i \leq$ max iterations **do**
5: Compute ghost node locations
6: Populate adaptation function values at each node
7: Compute forcing functions Φ, Ψ
8: Compute Winslow weights
9: Build global linear system
10: **for** $j \leq$ point iterative max **do**
11: Point iterative step
12: Snap boundary nodes back onto boundary using the geometry class
13: **end for**
14: **end for**

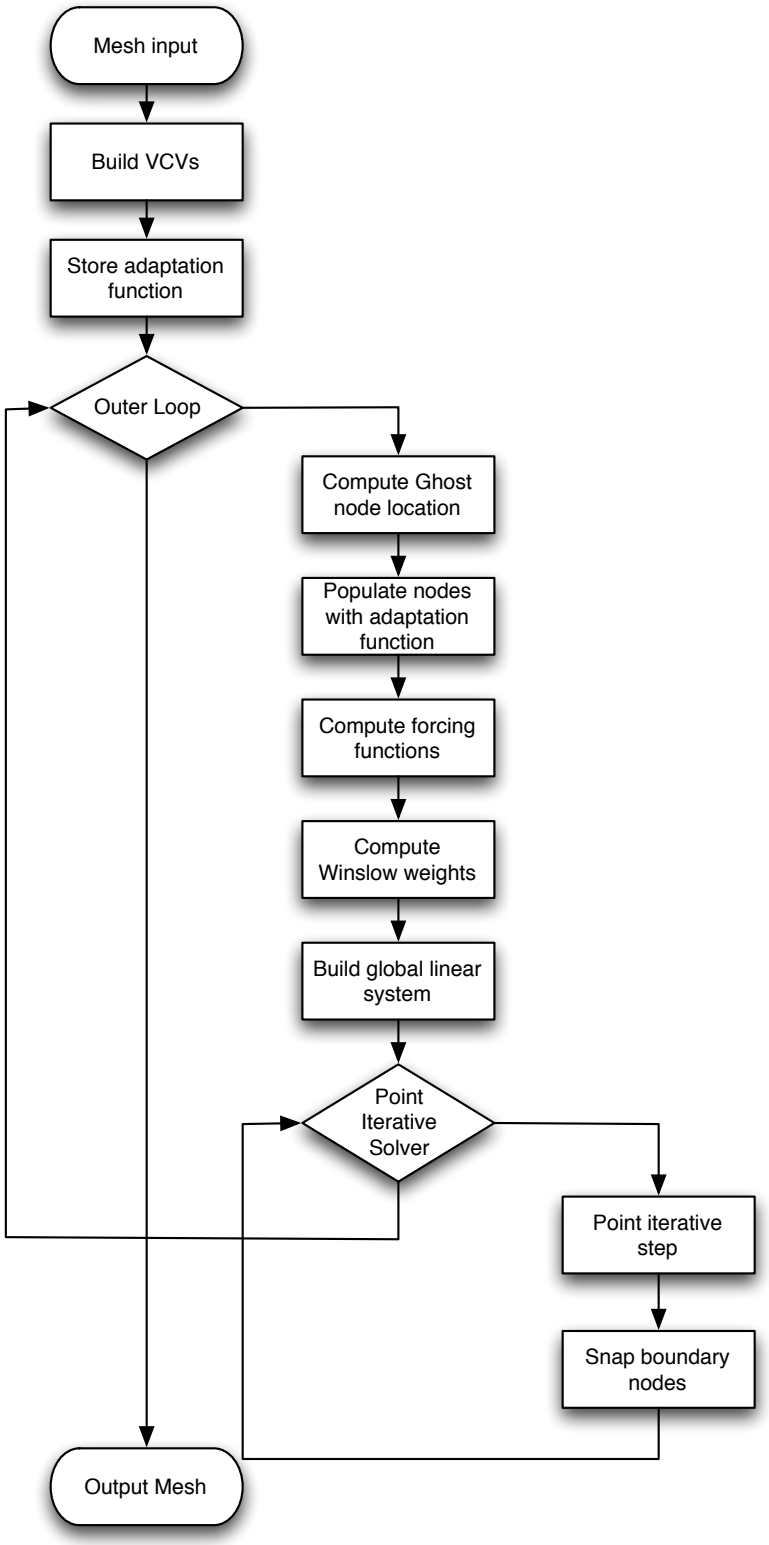


Figure 3.1 Forcing function flowchart.

CHAPTER 4

ADAPTATION BY VIRTUAL CONTROL VOLUME MANIPULATION

While solving the Winslow equations, the physical mesh will mimic changes in the computational mesh. Therefore, manipulating the virtual control volumes will cause changes in the resulting physical mesh. This can be exploited to obtain mesh adaptation. It has been determined that the scaling of the control volume seems to have no effect on the outcome of the physical mesh. Instead it is the relative length differences that are important [7]. For example, if edges are greater in length on one side of a virtual control volume then the other side, the physical mesh will show a difference in relative edge sizes accordingly. Since it is only the relative edge lengths that are important, manipulating the location of the center point in a control volume is a straightforward way to create these differences. It is then a problem of describing desired edge lengths and automating the offset of the central point.

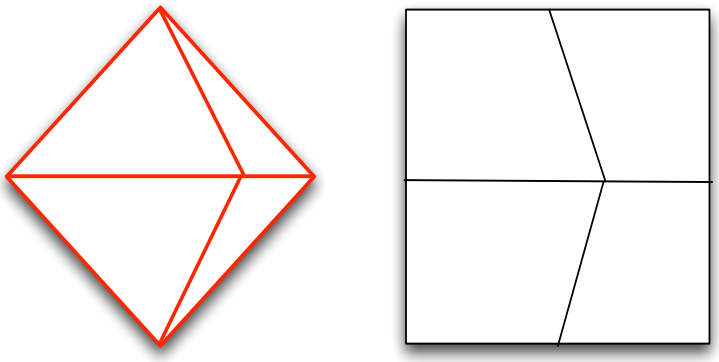


Figure 4.1 Manipulated VCV and resulting physical adaptation

Figure (4.1) shows an example of an all quadrilateral mesh and the manipulated virtual control volume for the interior point. The physical mesh reflects the alteration made in the computational domain.

Spacing Matching Adaptation

In Winslow based smoothing, Riemannian tensor fields have been used to describe spacing in a computational mesh. When the Winslow equations are satisfied, the physical spacing mimics the computational spacing. A Riemannian metric tensor (M) is a symmetric positive definite matrix that is the product of a rotation matrix R and a scaling matrix λ. The scaling matrix λ contains inverse square of desired distances along the directions described by R. The construction of M can be seen in equation 4.1.

$$M = R\lambda R^{-1} \tag{4.1}$$

$$M = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 \end{bmatrix} \begin{bmatrix} h_1^{-2} & 0 \\ 0 & h_2^{-2} \end{bmatrix} \begin{bmatrix} \vec{e}_1 \\ \vec{e}_2 \end{bmatrix}$$

Edges in the physical mesh can be measured by how well they match the desired spacing. Given an edge in the mesh, the metric length in the tensor space for that edge can be computed. An edge that exactly matches the spacing in λ has a metric length of 1.0. An edge that is smaller is less than 1.0; a larger edge is greater than 1.0. Computing the metric length of an edge between points A and B are show in equation 4.2.

$$l_{ab} = \sqrt{\overrightarrow{AB}^T \; M \; \overrightarrow{AB}} \tag{4.2}$$

To more closely match the spacing described by the Riemannian metric field, virtual control volumes can be manipulated by measuring the metric length of each directly connected

physical edge and adjusting the location of the central point of the virtual control volume by a metric length weighted average. Equation (4.4) describes the perturbation vector of a central point.

$$\Delta\vec{r} = s\sum_{i=0}^{e} l_i\vec{v}_i \tag{4.3}$$

$$p_{new} = p_{old} + \Delta\vec{r} \tag{4.4}$$

with $\vec{v}_i$ being the vector in the computational domain that points from the central point to the node i. A relaxation factor (s) can be used to control the pace at which central points can be offset. Since the computational domain must remain valid, the total offset of the central point must be restricted. Checks can be put in place to ensure that each triangle in the virtual control volume still has area larger than a tolerance. Figure (4.2) shows a virtual control volume of a node in a mesh that has a spacing in the x-direction smaller on the right side of the node than the left side. As the central point is adjusted, the edge vectors also adjust so that they are always pointing precisely from the central point to the corresponding outer point. The weighted average scheme only uses data from directly connected nodes.

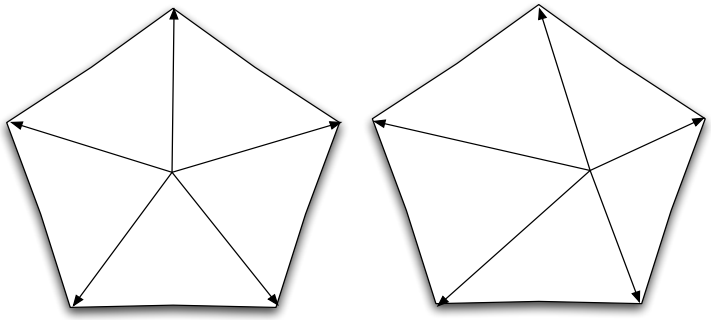


Figure 4.2 Perturbed control volume

If there is no gradient of the spacing field over a virtual control volume, the central point will only be moved to equalize edge lengths. This local control of the spacing field also gives

"built in" robustness to high gradients, as will be seen in Chapter 5, but also leads to an iterative process for determining how the virtual control volumes for each node need to be manipulated. Mesh adaptation by manipulating the central point to equalize gradients will not be able to always match spacing defined from a spacing field. Instead the method aims to equalize the metric edge lengths of each edge in the mesh.

Figure 4.3 shows a test mesh with 51 nodes initialized equally spaced along the x axis. The background tensor field is Cartesian aligned $e_1 = x$ and $e_2 = y$, $h_2 = 1.0$ everywhere and the h_1 spacing is

$$h_1 = \begin{cases} 0.2 & , 0.4 \leq x \leq 0.6 \\ 0.4 & , \text{otherwise} \end{cases}$$

Since there is no change in the spacing in the y direction, there will be no adaptation in the y direction. The central point perturbations occur in the region between these two spacings, a node that initially has central point manipulation is marked and followed as iterations progress. Four iterations later the central point of the marked node has to now decrease as it enters a zero gradient area but nodes behind it that previously had no manipulation are now seeing offsets. In the final mesh the marked node and many nodes behind it have gone through perturbation region and the mesh has settled down. Figure 4.4 shows the range of metric lengths on the mesh as computed by an average for each node of the metric length of each edge connected to the node. Notice that the range decreases between jumps. These jumps are caused by a new layer of nodes being moved from the outside region to the inner region. It is worth noting that even for this topology that should be able to match the metric length at each edge perfectly, the mesh does not converge to equal metric edge lengths everywhere, instead settling on a metric length range near 0.03.

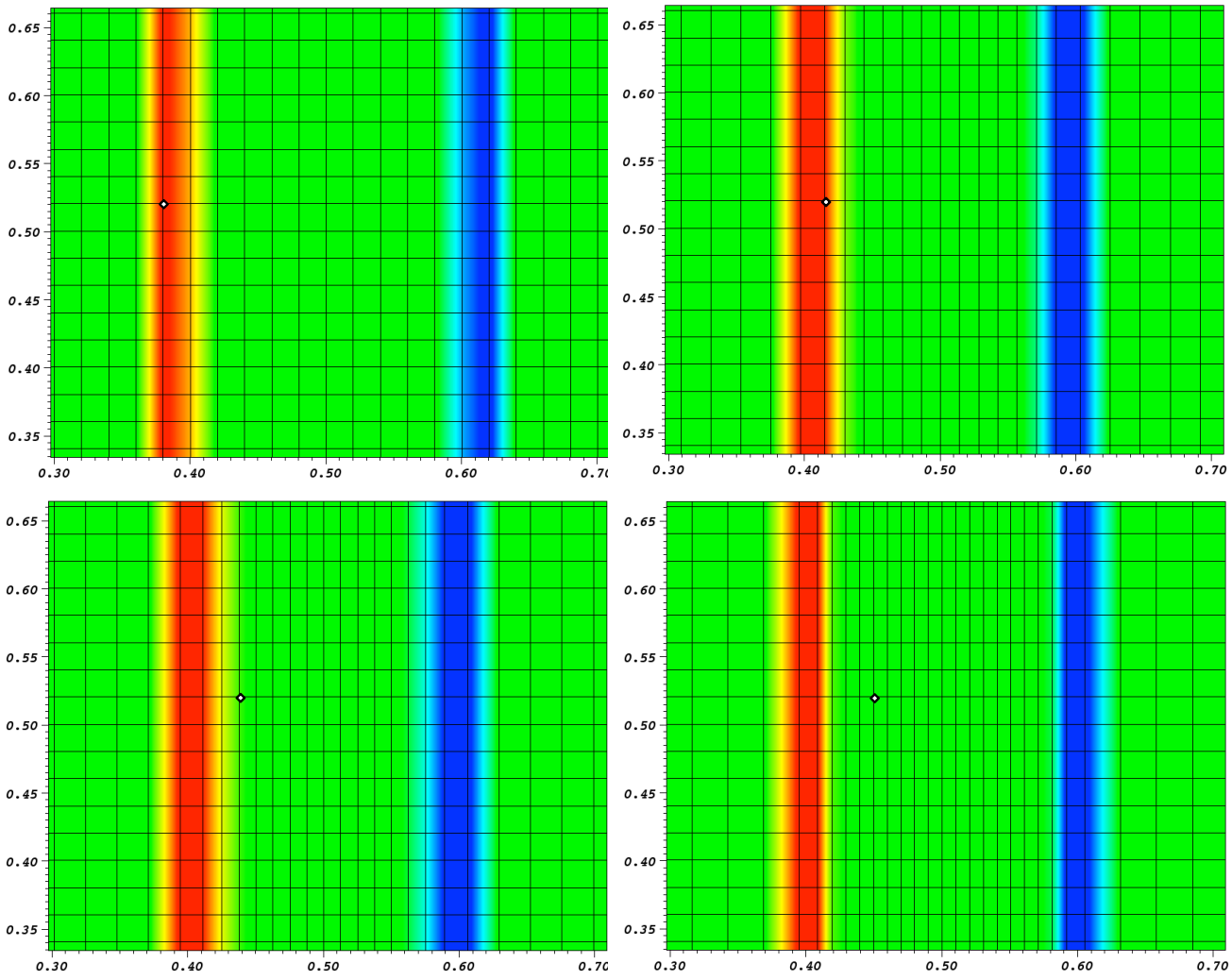


Figure 4.3 Slow central point adaptation

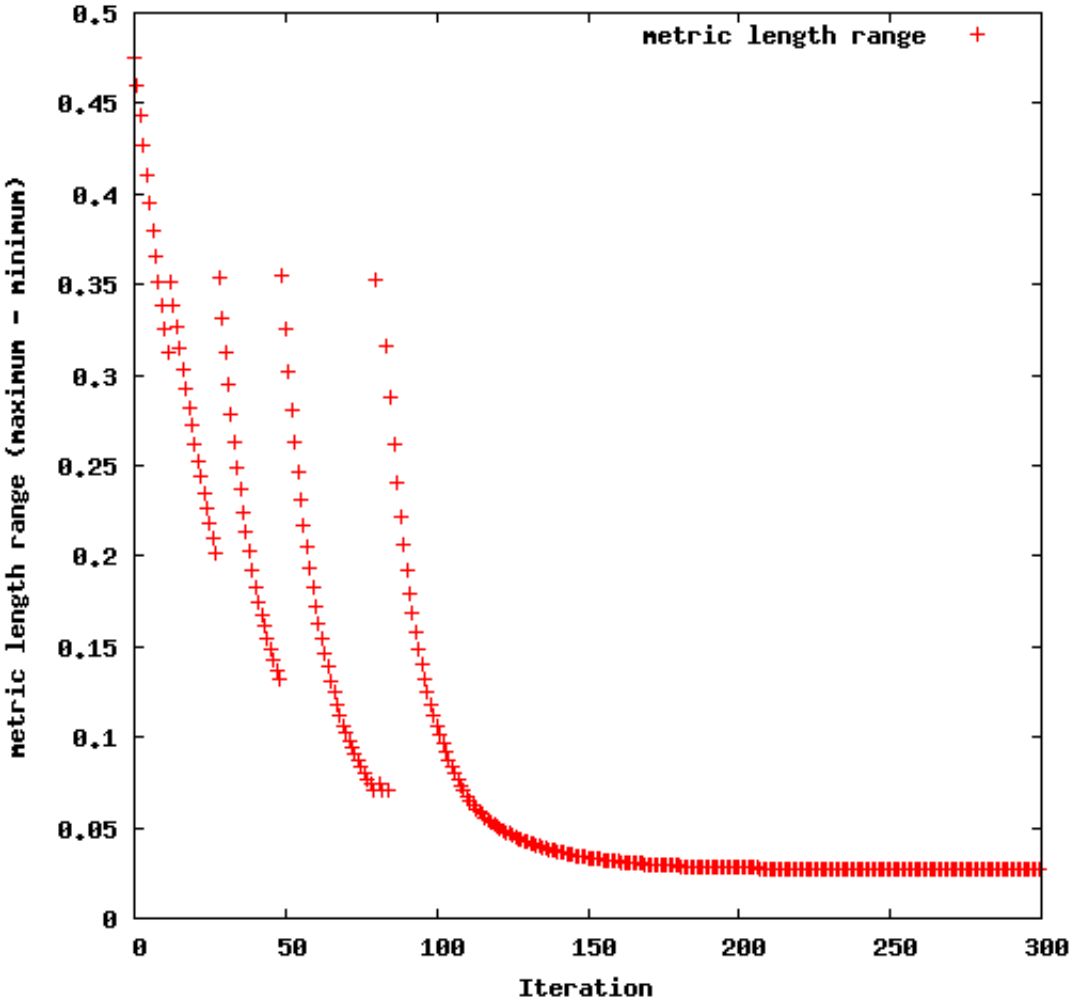


Figure 4.4 Progression of the metric length range

Calculation of Riemannian Metric Tensor Field

Riemannian metric tensor fields can be determined by a practitioner to gain some desired spacing. It can also be computed automatically to meet an equidistribution of gradients in some function. A simple two part equidistribution scheme is shown in equations 4.5 and 4.6.

$$wd^p = C \tag{4.5}$$

$$d_{\text{new}} = \sqrt[p]{\frac{\bar{C}}{w}} \tag{4.6}$$

In this scheme w is some weight function valid across the mesh, d the length of an edge and the power p can control smoothness C field. Using this an average value of C is computed across the entire mesh. By substituting $\bar{C}$ back into 4.6 and solving for length, a new length for each edge in the mesh can be computed using local values of w. Choosing the weights w to be based on gradients of some adaptation function will result in equally distributing the amount of change in the adaptation field between any two connected nodes in the mesh. The weights used are very similar to the weights proposed by Anderson [2].

$$w = 1 + a_s|\nabla f|^2 \tag{4.7}$$

where gradients are computed at node points and d is an edge length. A Riemannian metric tensor is computed in a region by aligning e_1 to the gradient vector and using values of d_{new} for h_1. The perpendicular spacing also be set to this spacing or chosen as an average distance between nodes in that region. The procedure for adaptation with central point offset is described below, followed by a flowchart in figure 4.5

```
1: Read in mesh
2: Build virtual control volumes
3: Store adaptation function into Quadtree storage, or Riemannian metric tensor field in
   the Spacing Library
4: for k ≤ max outer iterations do
5:    Evaluate metric lengths of edges
6:    Perturb the central point of virtual control volumes
7:    for i ≤ Winslow max iterations  do
8:       Compute ghost node locations
9:       Compute Winslow weights
10:      Build global linear system
11:      for j ≤ point iterative max do
12:         Point iterative step
13:         Snap boundary nodes back onto boundary using the geometry class
14:      end for
15:   end for
16: end for
```

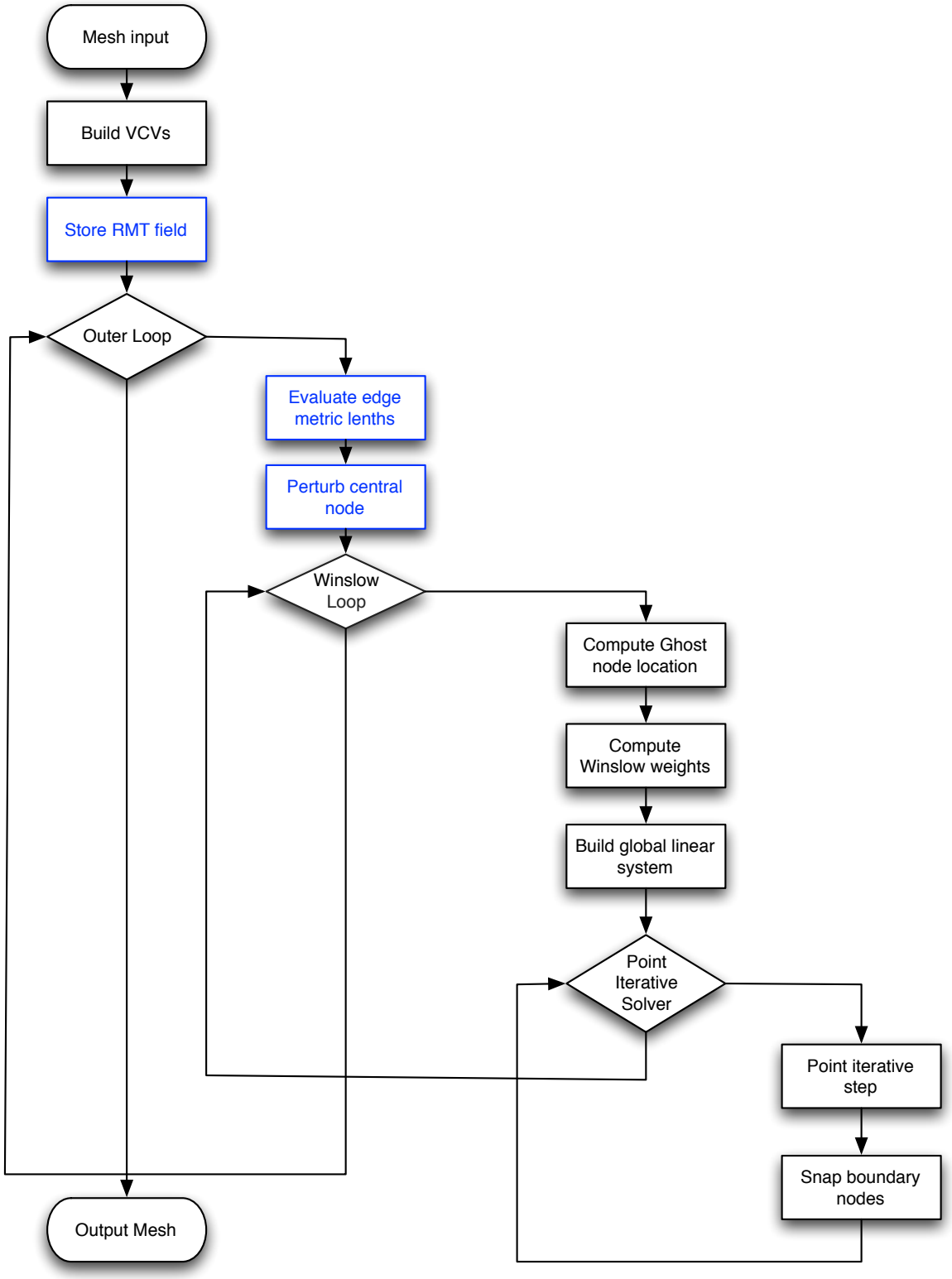


Figure 4.5 Central Point method flowchart.

CHAPTER 5
COMPARISON

The central point offset method and the method using forcing functions have been described. This chapter documents the comparison of the two methods. The first section compares some of the features and characteristics that each method has by exploring simple test adaptation functions on a small all quadrilateral mesh. In this section each method will be tested for control in adaptation in specified directions and for the level of adaptation capable of each method.

The second section focuses on comparing the performance of each method on more complex problems. First using a complex adaptation function, then using solution data from an Euler flow solver on a NACA 0012 airfoil.

Analytic Cases

For these cases, analytic functions are used as test adaptation functions that work to drive adaptation using either the central point offset method or the forcing function method. In all cases, the initial mesh is an equally spaced all quadrilateral mesh. The central point is restricted to a total offset of 0.7 to ensure that the virtual control volumes remain valid. The first example is a rapidly changing scalar field over a diagonal region. The analytic function adapted is on the following page.

$$f = \begin{cases} 1 & : x < \frac{y+10}{2} \\ \frac{7-x}{2} + \frac{y}{4} & : \frac{y+10}{2} \leq x \leq \frac{y+14}{2} \\ 0 & : x > \frac{y+14}{2} \end{cases}$$

This case is well suited for adaptation only in the x direction and the level of mesh refinement will be a measure of the number of points inside this shock region and $\pm$ 0.5 in the horizontal direction. The second case is adapted using scalar function:

$$f = \begin{cases} 0 & : 0 \leq y \leq 11 + 4sin(\pi x/12) \\ 0.5(y - 11 - 4\sin(\pi x/12)) & : 11 + 4\sin(\pi x/12) \leq y \leq 13 + 4\sin(\pi x/12) \\ 1.0 & : 13 + 4\sin(\pi x/12) \leq y \leq 24 \end{cases}$$

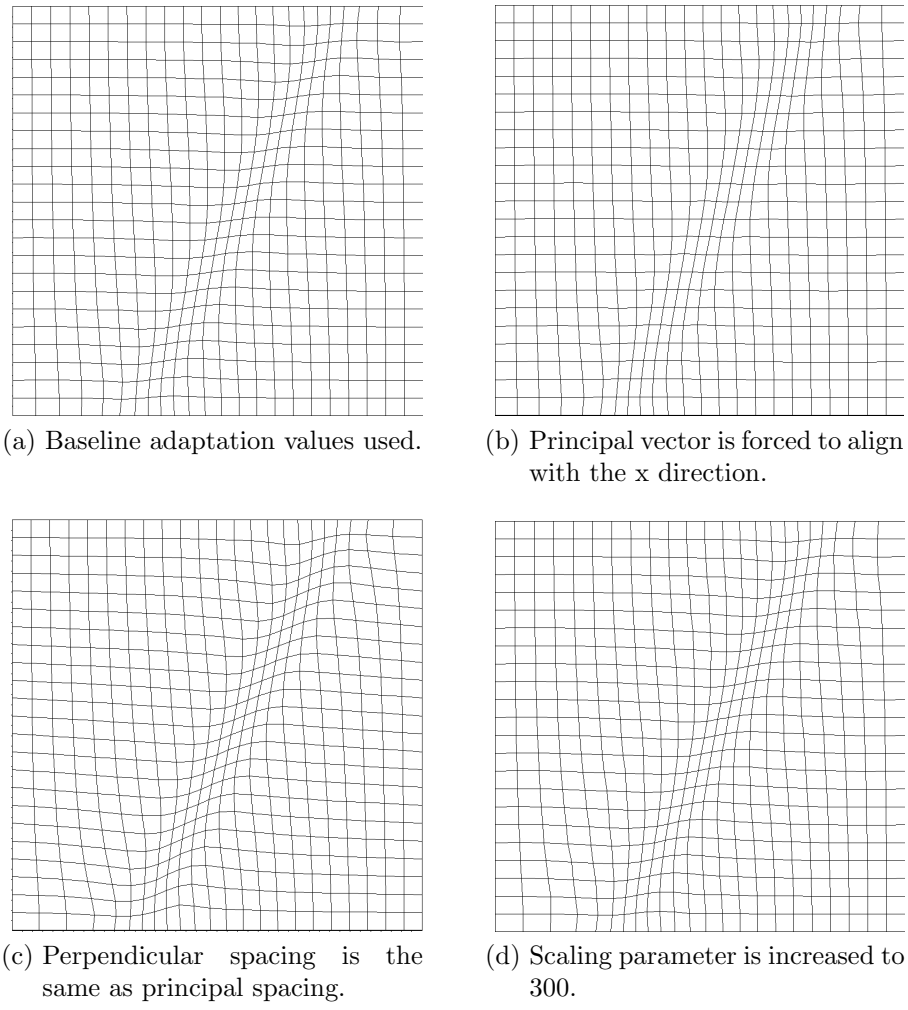
The many parameters described in Chapters 3 and 4 are explored here to determine their overall effect on the final mesh. Specifically, each method has a means to control the level of adaptation in a desired direction. For central point offset the orientation of the Riemannian metric tensor at a point is usually determined by the orientation of the gradient, however it can just as easily be forced to any specified direction. The two cases below do just that, forcing the alignment of e_1 to the x direction in the case of the diagonal shock, and the y direction for the sine wave case. The method using forcing functions has scaling parameters a, b which scale the gradient contribution in the ξ and η directions as shown in equation 3.2. Forcing a or b to be zero allows the construction of Φ and Ψ with no contributions from the gradient in that direction.

Figure 5.1 shows four meshes, each the result of the central point offset adaptation scheme on the diagonal shock test case, quality metrics and refinement metrics are shown in table 5.1

For the first mesh (5.1a) aligned e_1 with the direction of the gradient vector, the result is less skewed elements in the adapted region. With second mesh (5.1b) the principal direction (e_1) of each Riemannian metric tensor was aligned with the positive x direction. This resulted in minimal point movement in the y direction and the horizontal lines remain level. The third (5.1c) mesh forced the perpendicular spacing (h_2) to be the same as the principal spacing but actually increases the maximum and average aspect ratios. The final mesh (5.1d) shows the effects of increasing the scaling weight a_s, first described in equation 4.7. In this case a_s was set to 300 where as all other cases used $a_s = 100$. Increasing the scaling did noticeably increase the amount of adaptation, but not to a great extent, adding only 16 points to the adapted shock region.

Table 5.1 Mesh quality metrics for central point adapted shock wave meshes

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
a)	117	1.65	1.64	99.25	1.19	1.05
b)	117	1.49	1.41	109.42	1.37	1.08
c)	133	1.71	1.43	122.97	1.75	1.17
d)	127	1.63	1.19	113.97	1.42	1.10



(a) Baseline adaptation values used.

(b) Principal vector is forced to align with the x direction.

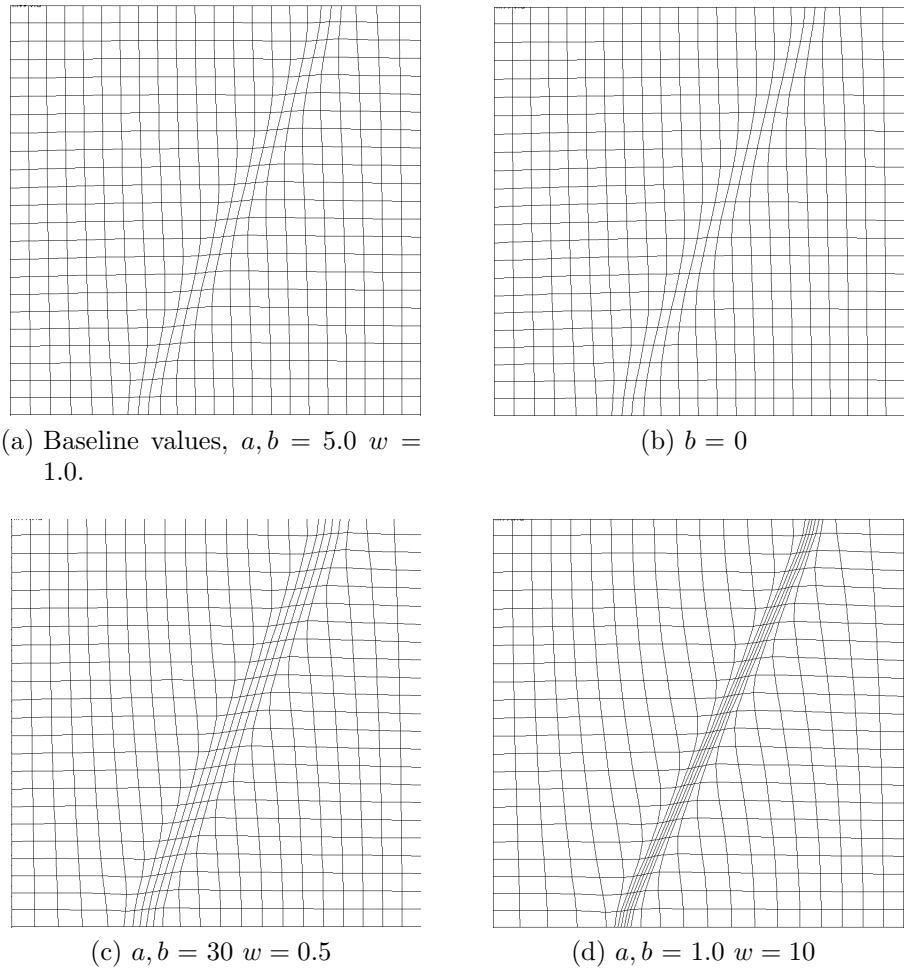
(c) Perpendicular spacing is the same as principal spacing.

(d) Scaling parameter is increased to 300.

Figure 5.1 Adapted meshes using central point offset

The forcing function adaptation of the diagonal shock case is shown in Figure 5.2 with quality metrics and refinement metrics in table 5.2. The first mesh (5.2a) is a baseline mesh using scaling parameters $a, b, w = 5.0$. The second mesh (5.2b) is with b= 0 to allow no contribution of the gradient in the y direction to affect the forcing functions. Successfully limiting adaptation in this way suggests that the alignment of the virtual control volumes was successful, and also may be desirable to simply control the level of adaptation along just one cardinal direction. The result is almost no adaptation in the y direction. Mesh (5.2c)

scales up $a, b = 30$ and scales down the relaxation scaling to $w = 0.5$. The refinement in the adapted region increases throughout a wide band. The final mesh (5.2d) with a, b =1.0, w = 10 has very fine resolution inside a more narrow adapted region almost doubling the number of points in the adapted region.



(a) Baseline values, $a, b = 5.0$ $w = 1.0$.

(b) $b = 0$

(c) $a, b = 30$ $w = 0.5$

(d) $a, b = 1.0$ $w = 10$

Figure 5.2 Adapted meshes using forcing functions for a diagonal shock

For this diagonal shock case central point offset could produce lower aspect ratio elements than the forcing function method, and each method was able to reasonably limit the spacing in the y direction. The forcing function method was able to greatly increase the level of refinement inside the adapted region.

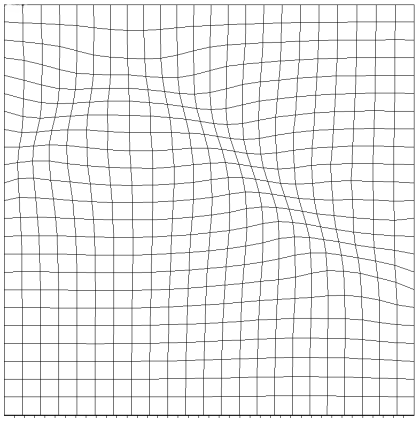
Table 5.2 Mesh quality metrics for forcing function shock wave meshes

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
a)	120	1.89	1.17	109.26	1.30	1.05
b)	120	1.95	1.17	104.78	1.22	1.04
c)	170	2.89	1.52	116.30	1.32	1.09
d)	236	7.40	2.88	123.72	1.32	1.09

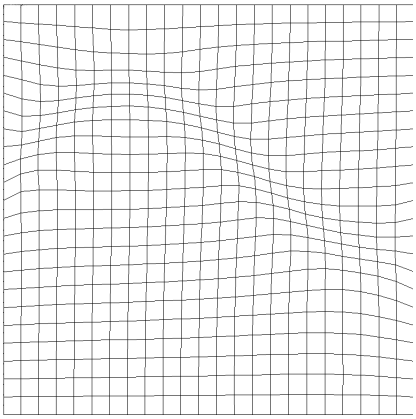
The next analytical test case uses an adaptation function with more curvature than the diagonal shock case. Figure 5.3 shows meshes resulting from central point offset adaptation. The first mesh (5.3a) is again using baseline values a_s= 100 with the principal vector aligned locally with the direction of the gradient vector. The second mesh (5.3b) forces the principal to be aligned with the y direction. Though there is some point movement in the x direction. The third mesh (5.3c) keeps the perpendicular spacing equal to the principal spacing which actually increases the maximum aspect ratio and maximum angle. The final mesh (5.3d) increases the scaling coefficient to 300, again having a small effect on the level of refinement in the adapted region, only 7 nodes more than the baseline case.

Table 5.3 Mesh quality metrics for central point adaptation for sine wave meshes

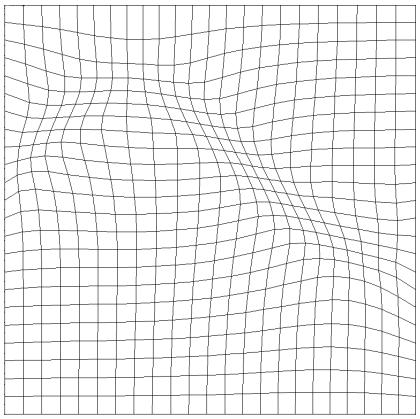
mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
a)	115	1.68	1.10	118.23	1.63	1.11
b)	112	1.56	1.12	118.21	1.59	1.10
c)	135	2.03	1.14	130.78	1.95	1.20
d)	122	1.87	1.16	122.08	1.722	1.13



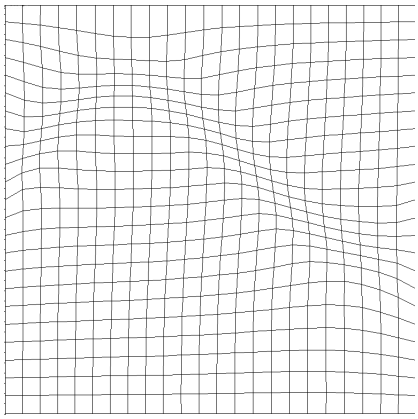
(a) Mesh with baseline adaptation values.



(b) Principal vector is forced to align in the y direction.

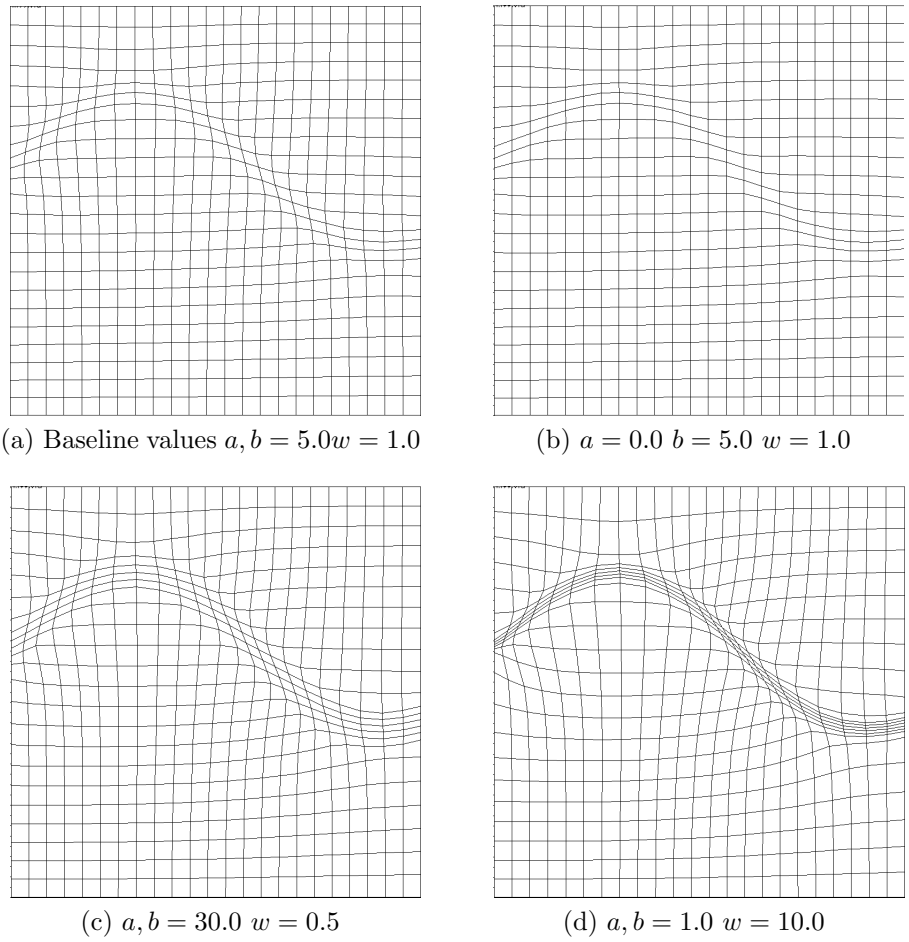


(c) Perpendicular spacing equal to principal spacing.



(d) Principal vector is forced to align in the y direction and the scaling parameter is set to 300.

Figure 5.3 Adapted meshes using central point offset for a sine wave



(a) Baseline values $a, b = 5.0 w = 1.0$ (b) $a = 0.0$ $b = 5.0$ $w = 1.0$

(c) $a, b = 30.0$ $w = 0.5$ (d) $a, b = 1.0$ $w = 10.0$

Figure 5.4 Adapted meshes using forcing functions for a sine wave

Adaptation using forcing functions is shown in figure 5.4. The baseline values for this case were again $a, b = 5.0$ $w = 1.0$ and the results of this configuration are shown in the first mesh (5.4a). Adaptation in the x direction is successfully restricted by setting the scaling parameter $a = 0$ in the second mesh (5.4b). The level of adaptation in the adapted region was increased by increasing scaling parameters a and b to 30 in the third mesh (5.4c), but not as much as by increasing the overall scaling parameter from $w = 1.0$ to $w = 10.0$.

The adaptation given by the sine wave test case is consistent with the first case. The forcing function method allows for greater control over the amount of spacing in an adapted

Table 5.4 Mesh quality metrics for forcing function sine wave meshes

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
a)	120	2.27	1.14	127.09	1.87	1.08
b)	120	1.95	1.17	104.78	1.22	1.04
c)	170	2.89	1.52	116.30	1.32	1.09
d)	229	9.51	2.40	159.22	3.02	1.25

region. As a surprising result from the central point scheme, attempting to refine edges in the x and y directions simultaneously decreases overall mesh quality in terms of skewness and aspect ratio.

Supersonic Shocktube

Analytic adaptation functions are useful for exploration, but ultimately mesh adaptation is meant to be used to adapt using field data determined from a simulation. For the following cases, the adaptation function used is Mach number from an Euler CFD solver. The solver used was written by the Author. The solver has not been validated, but still proves to be an adequate source for solution gradients for the purpose of testing. The geometry is a tube with flow constricted by a forward ramp and released by a backward ramp. The initial mesh is an all triangle mesh with just over 10,000 nodes. The mesh was generated using Pointwise [10] with no refined areas.

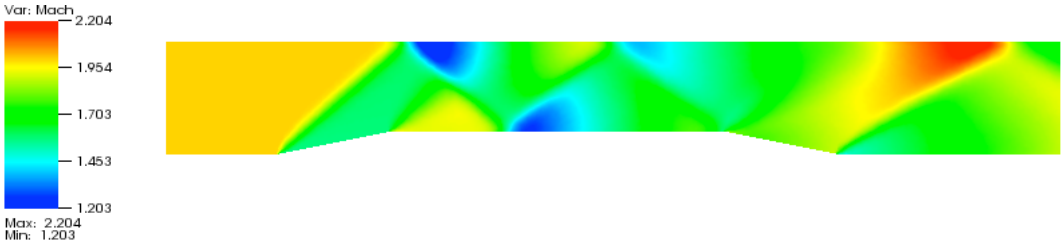


Figure 5.5 Mach number computed on the initial mesh for a supersonic tube case

Figure 5.6 shows the mesh adapted after using central point offset. The relaxation factor used was 0.01 ramped to 0.05 for 60 iterations and the scaling factor was 3.0. To speed up the method, only one inner point iterative step was taken at each outer iteration except the final iteration where 100 point iterative steps were taken. The mesh shows good adaptation and cell quality as shown by mesh quality and refinement metrics in table 5.5.

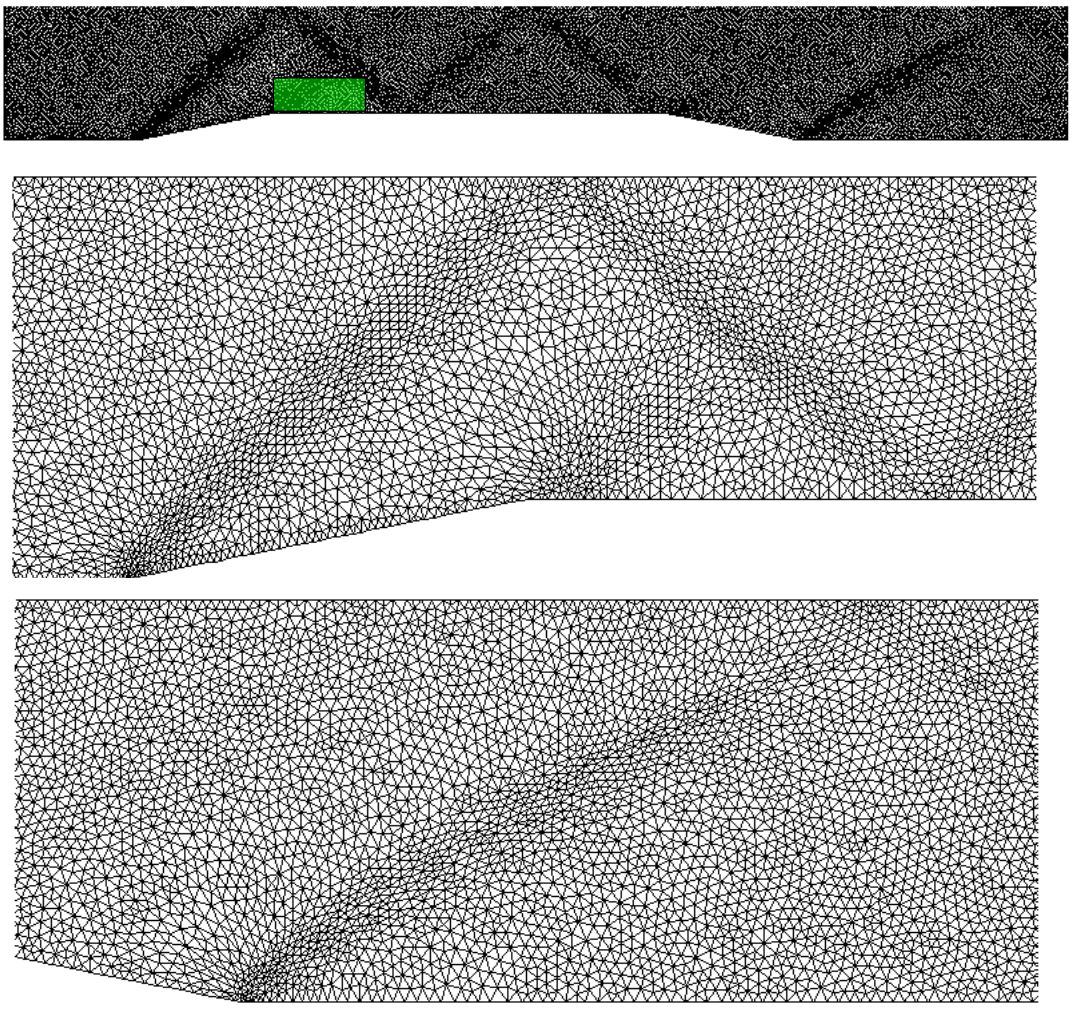


Figure 5.6 Shocktube mesh adapted with central point offset

Adaptation using forcing functions was not so easy. If the relaxation parameter was too small then no noticeable adaptation was produced. If the relaxation factor was too large

than the solution did not converge and quickly yielded invalid cells. There did not appear to be any middle ground. The reason for this volatile behavior the l can be seen in Figure 5.7 when looking at the values of Φ across the mesh. Two locations of the domain had forcing functions over 10 times greater than any other in the mesh, Ψ was similar. The gradient at these two locations were much larger than the rest of the domain, causing the forcing function variation. Scaling the forcing functions up causes instability from these two locations, and too small doesn't allow the rest of the mesh to become large enough to create meaningful adaptation.

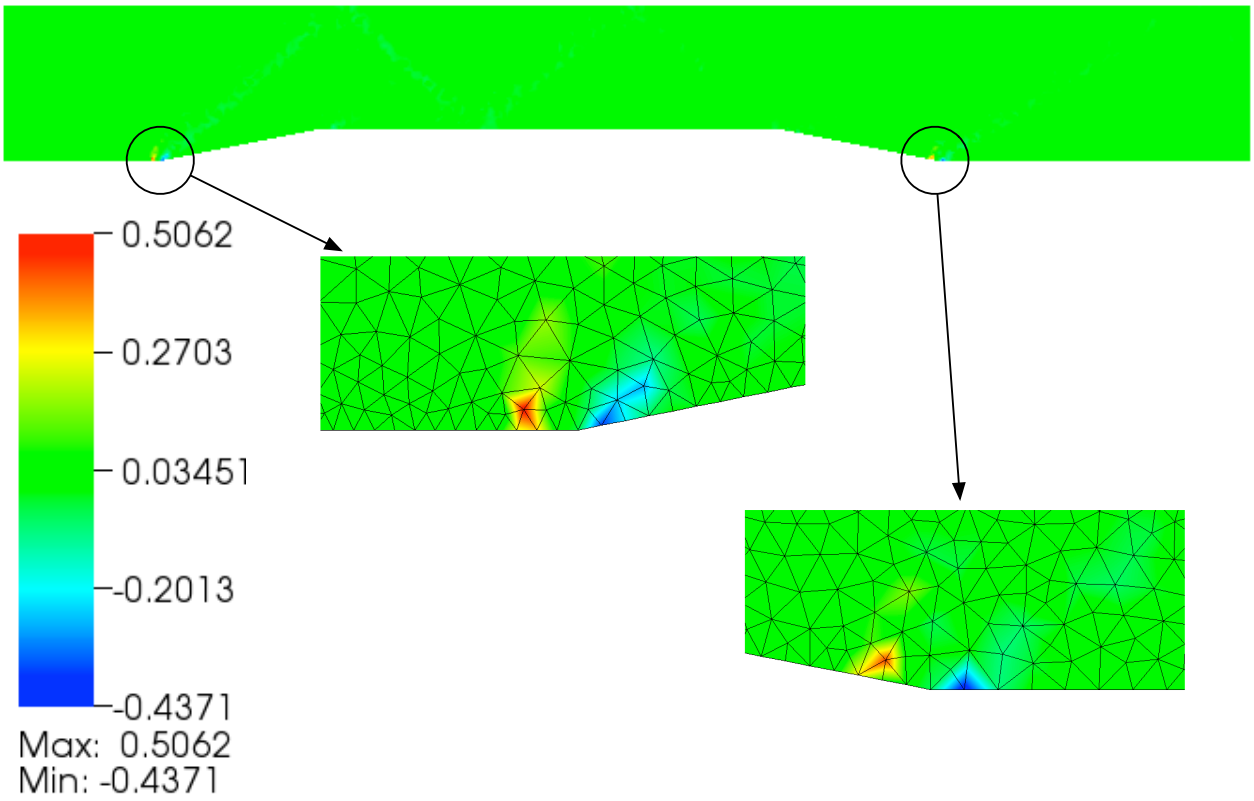


Figure 5.7 Φ for the shocktube test adapting to Mach number

This demonstrates the lack of robustness against high gradients that is a weakness of the forcing function method. The magnitude of the gradient of the adaptation function is shown in Figure 5.8. The flow field has very high gradients of Mach number at the bottom of the ramps causing very large variation in the forcing function through the mesh.

The central point offset method did not have difficulty with this large variation because the limitation placed on the virtual control volume to keep the central point inside the stencil. This hard limit effectively ignores all contributions of relatively high gradients past a threshold value. To achieve similar results using forcing functions, an artificial limit was placed on gradients. Nodes with gradient values more than some multiple of the standard deviation away from the average were clipped, forcing the range of the gradient to be smaller and more manageable. The resulting magnitude is shown in figure 5.9. To further manage the range of forcing functions, scaling of gradient magnitudes and hard limits on the values of forcing functions can also be used either in place of or in addition to the gradient magnitude range manipulation.

Figure 5.8 Magnitude of gradients in Mach number before range restraint

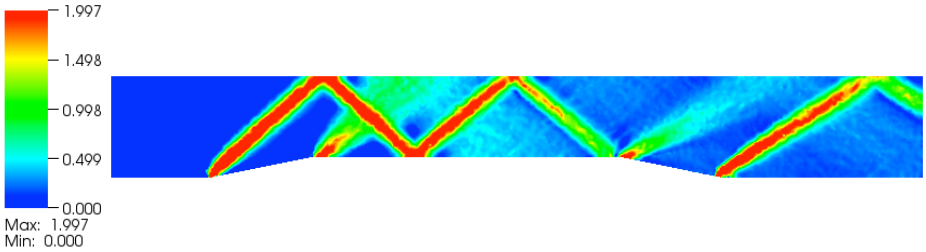


Figure 5.9 Magnitude of gradients in Mach number after range restraint

Because the large variations in forcing functions cannot, be used subtle changes in a flow field cannot be adapted to using the forcing function method as well as the central point offset method. This can be seen by the level of refinement in the adapted region marked as the green box in figures 5.6 and 5.10. Table 5.5 shows some mesh quality metrics and the number of nodes in the adapted region for all three meshes.

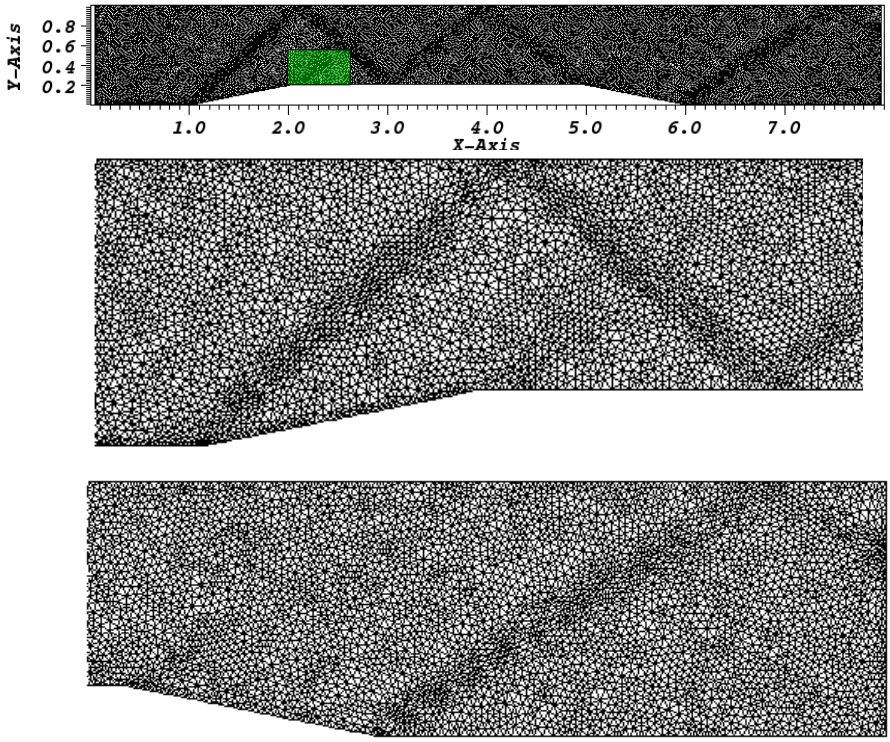


Figure 5.10 Shocktube mesh adapted with forcing functions after gradient range adjustment

Table 5.5 Mesh quality metrics and level of refinement for the three shocktube meshes

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
Initial	257	1.97	1.08	116.67	2.07	1.30
CP	280	3.68	1.08	135.86	2.84	1.31
FF	258	2.52	1.07	125.10	2.75	1.27

NACA 0012 Airfoil

The two following cases used the numerical solution obtained from an Euler flow solver, on a 6,300 node, all triangular mesh. The test cases were run for two steady state cases of a NACA 0012 airfoil. Though these test cases are much more complex than the previous tests. These cases should test each methods ability to adapt large regions of a solution field simultaneously, especially with problems that have large variations in the magnitude of the solution gradient.

The first case has freestream Mach number is 0.75 with an angle of attack of 4.0 degrees. A plot of the Mach number in the numerical solution is shown in figure 5.11. There is shock developing about $\frac{2}{3}$ down the chord can be seen in this solution, but it is of considerably poor quality.

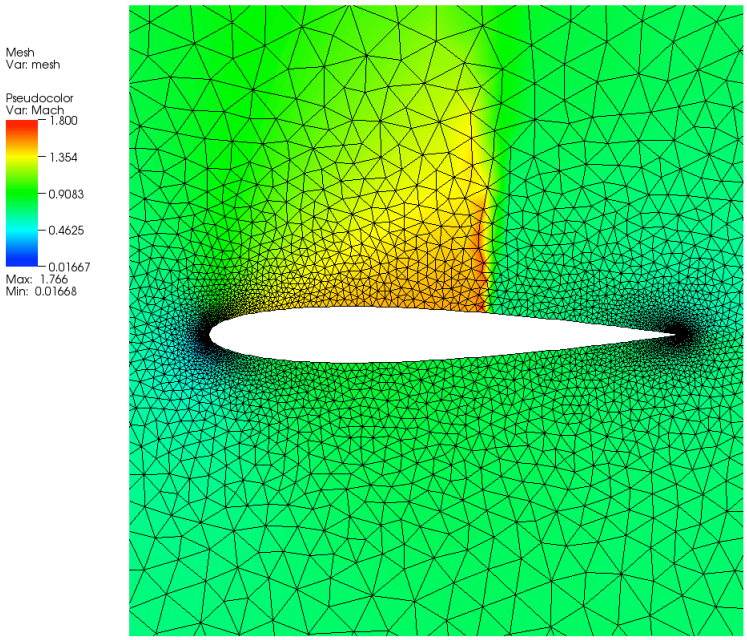


Figure 5.11 NACA 0012 airfoil numerical solution

The solution data run on this initial mesh was stored in the 2D spacing library developed by David Collao of UTC [11]. As points were adapted, the adaptation function remained fixed rather than moving as the points moved. Figure 5.12 shows the meshes that were adapted using both the central point offset method (top) and the forcing function method (bottom). The meshes were then used in the Euler solver to compute the flow solution again. The central offset point method led to the solution of a straighter shock off the airfoil, but the shock developed upstream of the majority of the adaptation intended for the shock.

The forcing function method originally produced no adaptation near the shock. Again the magnitude of the gradient field was clipped at each node that had a gradient magnitude greater than 1.5 times the standard deviation of the entire mesh. This finally allowed for the mesh shown. The forcing function method produced a weaker shock that was not as straight.

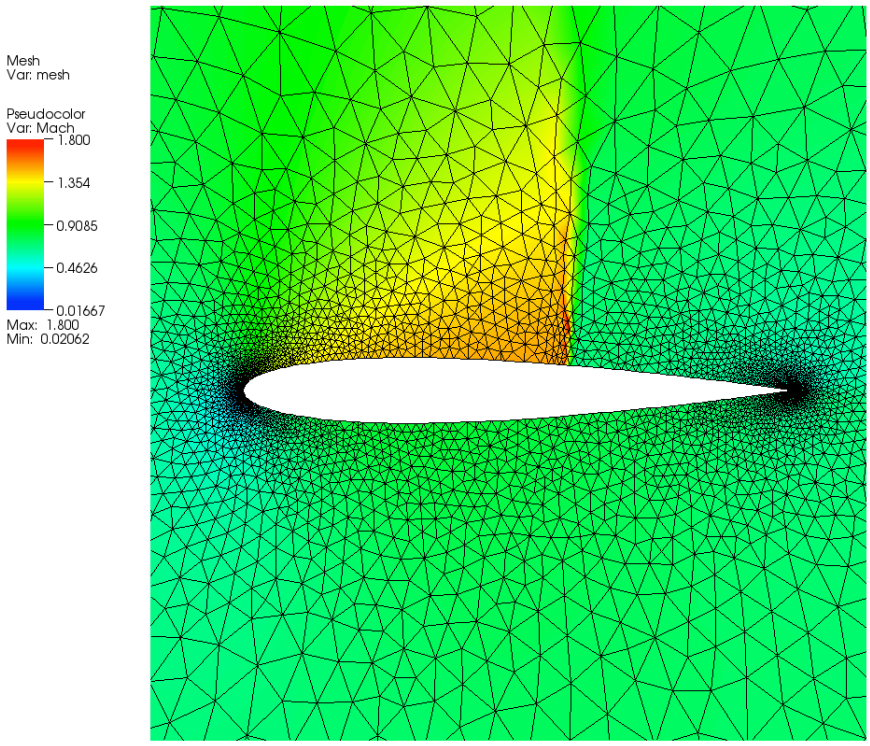
The convergence rate of the two meshes was also different than the initial mesh. Both adapted meshes converged almost 100 iterations faster than the initial mesh. All cases were run using the same CFL.

Adapting to this CFD solution also gives the opportunity to talk briefly about the computational cost of adaptation with respect to the computational cost for a CFD solution on the same mesh. On the initial mesh, running 500 iterations of the Euler solver took 85.9 seconds running on a desktop computer with using a 2.7 GHz Intel i5 processor. Using the central point adapted mesh the solver took 86.5 seconds, and the forcing function mesh took 72.8 seconds. On average the solver took 81.7 seconds to run 500 iterations. As figure 5.13 shows, 500 iterations was sufficient to converge the system. Adapting the mesh using the central point method to obtain the mesh shown in figure 5.12a took 7.2 seconds while the adapting using the forcing function method took 3.7 seconds. For this problem, as a percentage of the CFD solution time, that places adaptation between 4.5% and 8.8% of the computational time required of a steady state solution for the same mesh. This analysis does

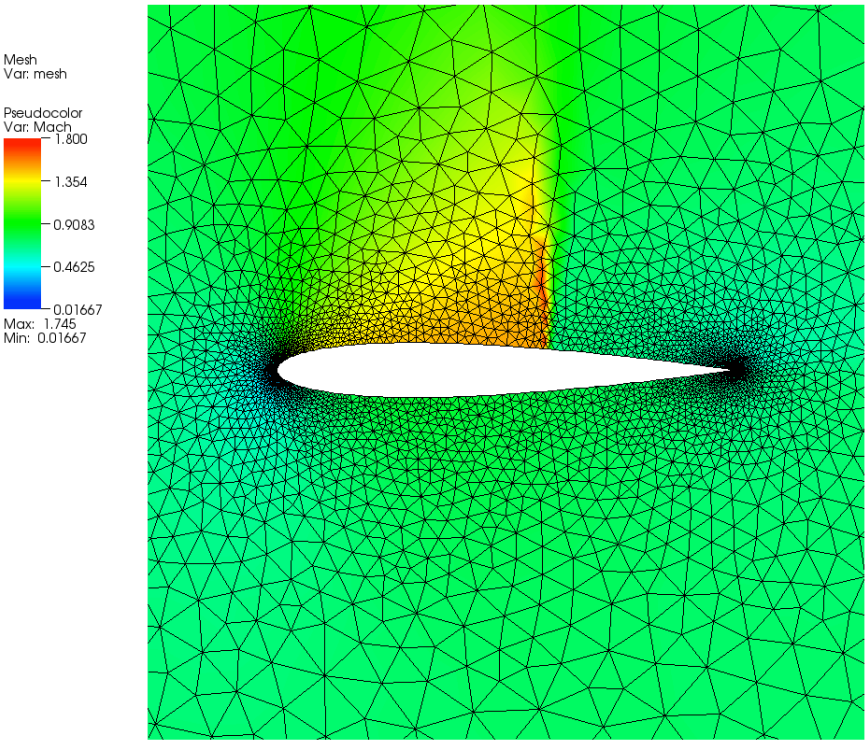
not take into consideration the increased convergence rate that both adaptation methods yielded.

Table 5.6 Mesh metrics for Mach of 0.75 and an angle of attack of 4.0 degrees

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
initial	52	1.86	1.07	114.23	2.09	1.30./
CP	77	4.13	1.06	124.00	7.43	1.27
FF	63	2.52	1.06	120.58	2.77	1.25



(a) Mesh adapted using central point offset.



(b) Mesh adapted using forcing functions.

Figure 5.12 Mach number solution on adapted meshes

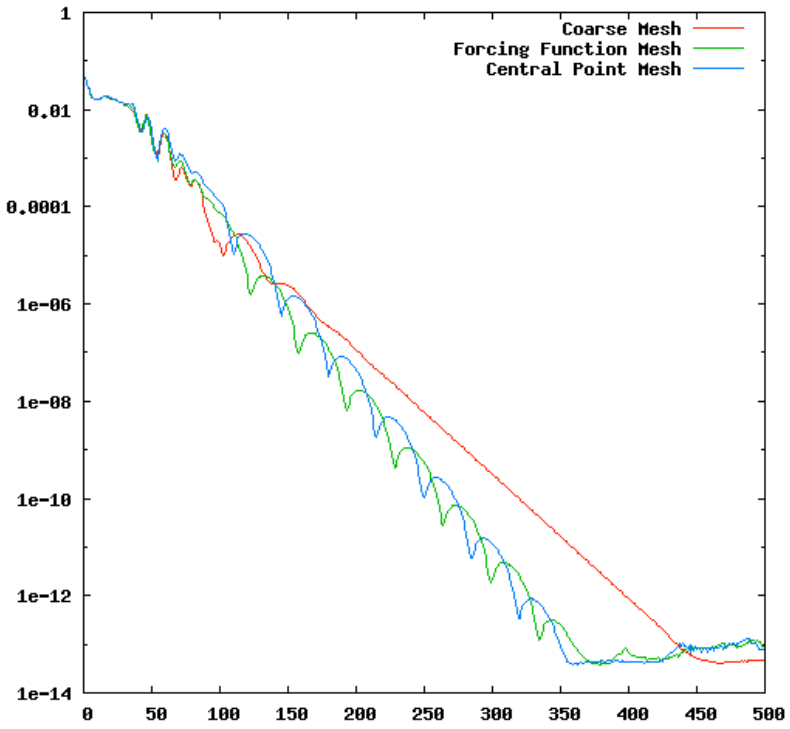


Figure 5.13 Convergence rate of adapted NACA meshes

The second case was again done on the NACA 0012 airfoil, with freestream Mach number of 0.95 and a zero angle of attack. The resulting meshes from each method are shown in figure 5.14. In this case the forcing function adaptation produced higher refinement in the test adaptation region while the central point method yielded a worse maximum aspect ratio and maximum skew, though averages for most quality metrics were similar.

All three meshes (central point, forcing function and baseline) converged within 25 iterations when used to compute the flow solution. However differences can be seen in figure 5.15 between the computed solution using the central point method and the forcing function method. The forcing function produced results almost indistinguishable from the initial coarse mesh. The forcing function adapted mesh was more adapted and did produce smoother contour lines for the shocks.

Table 5.7 Mesh metrics for Mach of 0.95

mesh	adapted points	max aspect ratio	avg aspect ratio	max angle	max skew	avg skew
initial	47	1.86	1.07	114.23	2.09	1.30
CP	56	10.87	1.06	120.95	20.11	1.27
FF	85	3.82	1.06	133.49	2.78	1.25

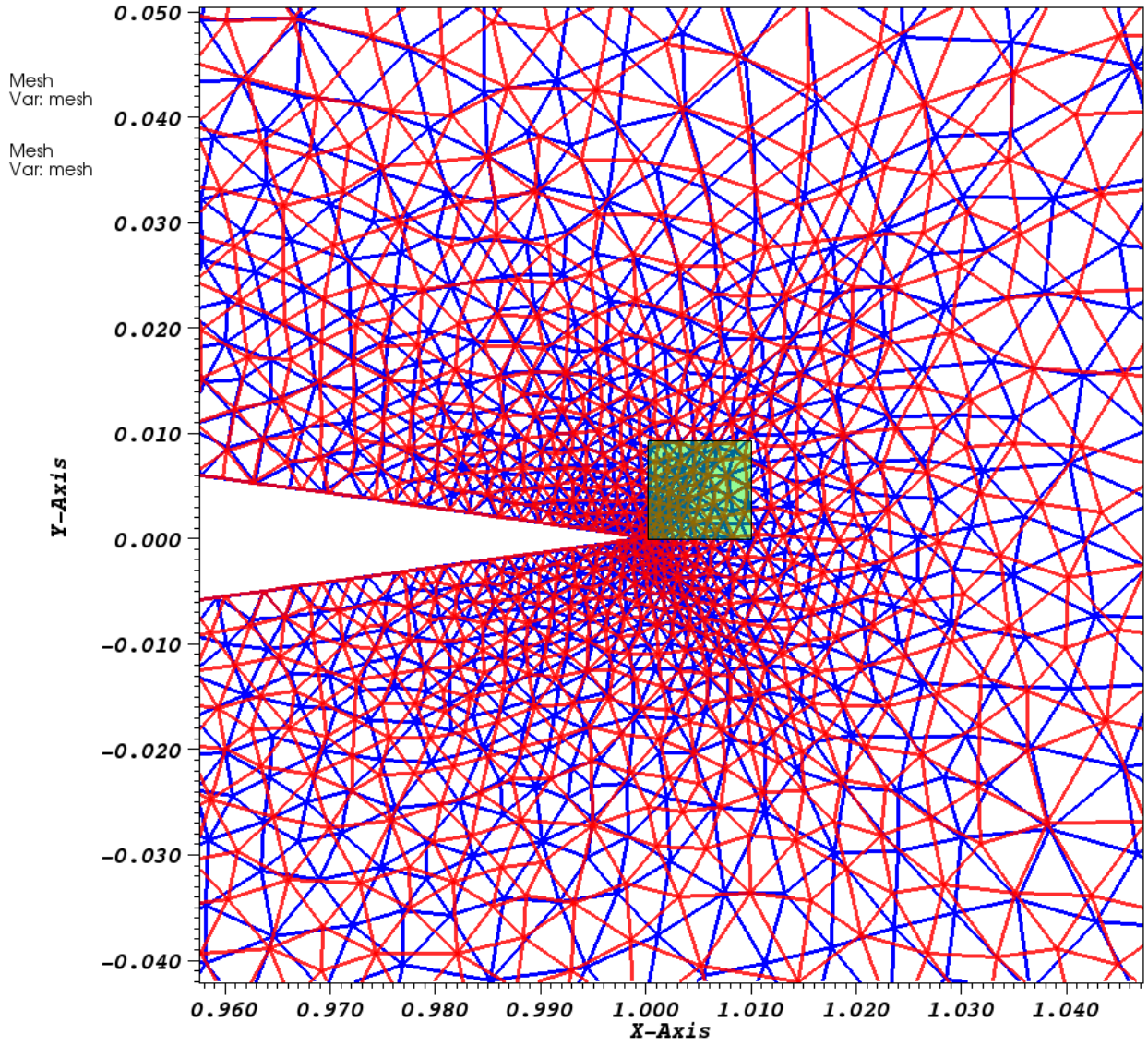


Figure 5.14 Adapted NACA 0012 mesh, tail shocks

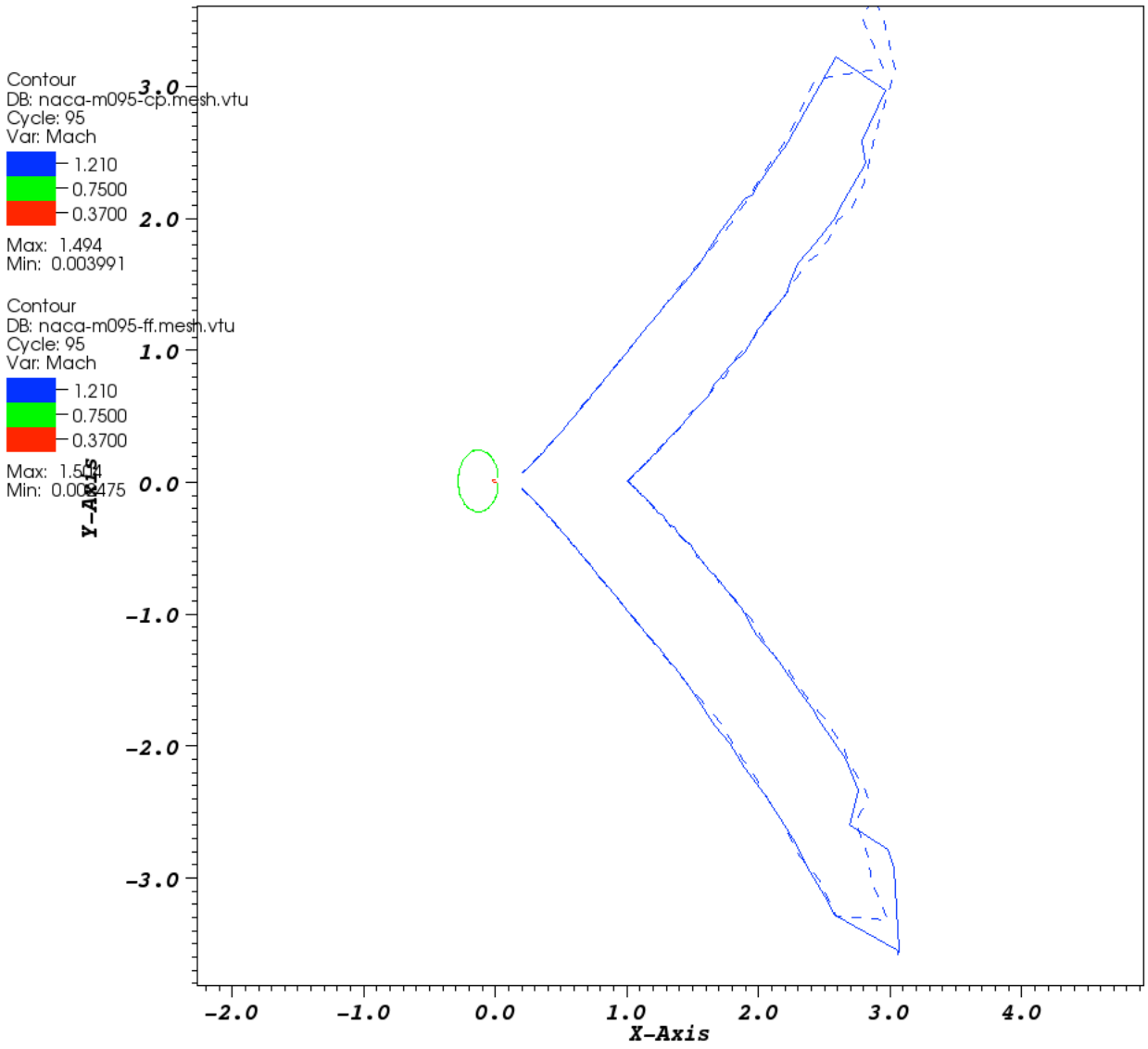


Figure 5.15 Mach number computed using the adapted mesh from central points and forcing function

CHAPTER 6
CONCLUSIONS

A new method for mesh adaptation, called the central point offset method, has been presented using automated and iterative manipulation of the computational domain that, in the context of a Winslow based elliptical solver can produce good quality meshes. This new method is intrinsically robust when adapting to solution fields with high variations in the magnitude of the gradient field. This robustness is due to an intuitive restriction that while virtual control volumes can be manipulated, the computational domain must remain valid. While robust, manipulation of the virtual control volume could not achieve the level of refinement during some analytic test cases. The method also has the ability to use a spacing field that is user defined to adapt the mesh.

In contrast, adaptation through forcing functions was shown to produce high quality meshes with more control on the level and direction of adaptation when using simple analytic adaptation functions. However, the method became unstable when applied to more complex problems using actual simulation data. This instability was addressed by limiting the range of gradients in the solution field. Unfortunately, limiting the solution gradients caused the degree of adaptation be become unsatisfactorily small in some of the test cases.

Neither method can be considered a mature technology, further research is needed for both. The virtual control volume manipulation used here only altered the location of a single point in the computational stencil. To achieve the control and refinement presented by the forcing functions, virtual control volume manipulation may have to be extended to include alteration of the entire computational stencil.

The robustness of the central point method came about through an intuitive implementation of limits on the level of virtual control volume manipulation. While the same limits do not apply to the forcing function method, additional constraints could be implemented that would increase the robustness of the method. Further research with forcing function adaptation should definitely include using a more stable linear solver such as GMRES. The stability problem in this method seem to be a problem with the range and scale of the forcing functions. Or possibly by using a gradient calculation method that is more dissipative such as a least squares gradient. Research into scaling or skewing the forcing functions should also be pursued. This may lead to a new, more general, way of computing forcing functions to control the mesh. A way of automating the calculation forcing functions to match some desired mesh quality may be possible.

While both methods currently have issues that need to be addressed, the methods do show promise. The computational time required for adaptation has been shown as well under 10% of the computational time required for CFD analysis. And solution based adaptation was also able to increase the convergence rate of the CFD solver.

In the future these methods must be implemented for a distributed memory parallel paradigm to allow for adaptation of very large meshes. While parallel implementations always introduce new complexities, both adaptation methods should be as well suited for parallelization as CFD analysis codes are. The only step in each method that would require communication would be solving the pseudo-linear system.

One final note, the two methods are not mutually exclusive. It could be possible to combine the methods and do virtual control volume manipulation and forcing functions simultaneously. A hybrid method for adaptation may be able to produce the control over refinement from forcing functions while still retaining robustness.

REFERENCES

[1] Winslow, A. M., "Adaptive-Mesh Zoning By The Equipotential Method," *Lawrence Livermore Labratory Report*, Vol. UCID-19062, 1987. 2

[2] Anderson, D. A., "Equidistribution Schemes, Poisson Generators, and Adaptive Grids," *Applied Mathematics and Computation*, Vol. 24, 1987, pp. 211–227. 2, 18, 28

[3] Thompson, P. D. and Middlecoff, J., "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptical equations," *AIAA*, Vol. 18, No. 6, 1979, pp. 652–656. 2

[4] Dwyer, H. A., "Adaptive Grid Method for Problems in Fluid Mechanics and Heat Transfer," *AIAA*, Vol. 18, No. 10, 1980, pp. 1205–1212. 2

[5] Knupp, P. M., "Winslow Smoothing on Two-Dimensional Unstructured Meshes," *Engineering With Computers*, Vol. 15, 1999, pp. 263–268. 2

[6] Karman, S. L. and Sahasrabudhe, M., "Unstructured Adaptive Elliptic Smoothing," *Aerospace Science Meeting and Exhibit*, 45. 2

[7] Masters, J. S., *Winslow Elliptic Smoothing Extended to Apply to General Regions of an Unstructured Mesh*, Ph.D. thesis, University of Tennessee at Chattanooga, 2010. 2, 22

[8] Thompson, J. F., Thames, F. C., and Mastin, C. W., "Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Fields Containing any Number of Arbitrary Two-Dimensional Bodies," *Journal of Computational Physics*, Vol. 15, 1974, pp. 299–319. 4

[9] Sahasrabudhe, M., *Unstructured Mesh Generation and Manipulation based on Elliptic smoothing and Optimization*, Ph.D. thesis, University of Tennessee at Chattanooga, August 2008. 11

[10] "www.Pointwise.com," 2011. 15, 39

[11] Collao, D., *Generation and Optimization of Spacing Fields*, Master's thesis, University of Tennessee at Chattanooga, 2011. 45, 62

APPENDIX A

TRANSFORMATION OF WINSLOW EQUATIONS

The Winslow equations are expressed as a Laplacian operator on the computational coordinates set equal to zero or set to forcing functions P, Q. The Winslow equations are poisson equations in the computational space, shown below in the P, Q form:

$$\nabla^2 \xi = P$$
$$\nabla^2 \eta = Q \tag{A.1}$$
$$\tag{A.2}$$

in Φ, Ψ form:

$$\nabla^2 \xi = (\nabla \xi \cdot \nabla \xi)\Phi$$
$$\nabla^2 \eta = (\nabla \eta \cdot \nabla \eta)\Psi \tag{A.3}$$

with the computational space represented by (ξ, η) and the physical space represented as (x, y). The mapping between the computational space and physical space is defined by:

$$\xi = \xi(x, y), x = x(\xi, \eta)$$
$$\eta = \eta(x, y), y = y(\xi, \eta) \tag{A.4}$$

To transform from the computational domain to the physical, the metric linear identities in 2D are

$$\xi_x = \tfrac{y_\eta}{J}, \quad \xi_y = \frac{-x_\eta}{J}$$
$$\eta_x = \tfrac{-y_\xi}{J}, \quad \eta_y = \frac{x_\eta}{J}$$
$$J = x_\xi y_\eta - x_\eta y_\xi \tag{A.5}$$

Applying the chain rule for differentiation

$$f_x = f_\xi \xi_x + f_\eta \eta_x \tag{A.6}$$

yielding

$$f_{xx} = f_{x_x} = f_{x\xi}\xi_x + f_{x_\eta}\eta_x \tag{A.7}$$

Substituting the and simplifying we have

$$\begin{aligned} \xi_{xx} &= \frac{y_\eta}{J^3}(Jy_{\eta\xi} - y_\eta J_\xi) - \frac{y_\xi}{J^3}(Jy_{\eta\eta} - y_\eta J\eta) \\ \xi_{yy} &= \frac{-x_\eta}{J^3}(J_\xi x_\eta - Jx_{\eta_\xi}) + \frac{x_\xi}{J^3}(x_\eta J_\eta - Jx_{\eta\eta}) \end{aligned} \tag{A.8}$$

with

$$\begin{aligned} J_\xi &= x_\xi y_{\eta\xi} + y_\eta x_{\xi\xi} - x_\eta y_{\xi\xi} - y_\xi x_{\xi\eta} \\ J_\eta &= x_\xi y_{\eta\eta} + y_\eta x_{\eta\xi} - x_\eta y_{\eta\xi} - y_\xi x_{\eta\eta} \end{aligned} \tag{A.9}$$

So substituting and simplifying*

$$\begin{aligned} \xi_{xx} = &\ \tfrac{1}{J^3}(-2x_\eta y_\xi y_\eta y_{\eta\xi} - y_\eta^3 x_{\xi\xi} + y_\eta^2 x_\eta y_{\xi\xi} + 2y_\eta^2 y_\xi x_{\eta\xi} + x_\eta y_\xi^2 y_{\eta\eta} - y_\eta y_\xi^2 x_{\eta\eta}) \\ \xi_{yy} = &\ \tfrac{1}{J^3}(-2x_\eta^2 x_\xi y_{\eta\xi} - x_\eta^2 y_\eta x_{\xi\xi} + x_\eta^3 y_{\xi\xi} + 2x_\xi x_\eta y_\eta x_{\eta\xi} + x_\eta x_\xi^2 y_{\eta\eta} - x_\xi y_\eta x_{\eta\eta}) \end{aligned}$$

With substitutions of α, β and γ

$$\nabla^2\xi = \frac{-y_\eta(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta}) + x_\eta(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta})}{J^3} \tag{A.10}$$

*and making use of the mixed derivative theorem

Defining the operator

$$G(\cdot) = \alpha \frac{\partial^{2(\cdot)}}{\partial \xi^2} - 2\beta \frac{\partial^2(\cdot)}{\partial \xi \eta} + \gamma \frac{\partial^2(\cdot)}{\partial \eta^2} \tag{A.11}$$

Which gives

$$\begin{aligned} \nabla^2 \xi &= \frac{-y_\eta G(x) + \alpha x_\eta G(y)}{J^3} \\ &= \frac{-(\xi_x G(x) + \xi + y G(y))}{J^2} \\ &= \frac{-\nabla \xi \cdot G(r)}{J^2} \end{aligned} \tag{A.12}$$

Similarly $\nabla^2 \eta$ can be determined as

$$\nabla^2 \eta = \frac{-y_\xi(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta}) + x_\xi(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta})}{J^3} \tag{A.13}$$

yielding

$$\nabla^2 \eta = \frac{-\nabla \eta \cdot G(r)}{J^2} \tag{A.14}$$

The Winslow equations in P, Q form can then be described as:

$$\begin{aligned} \nabla^2 \xi &= \frac{-\nabla \xi \cdot G(r)}{J^2} = P \\ \nabla^2 \eta &= \frac{-\nabla \eta \cdot G(r)}{J^2} = Q \end{aligned} \tag{A.15}$$

or

$$\begin{aligned} G(r) &= -(\nabla \xi)^{-1} \cdot J^2 P = -J^2(x_\xi P + y_\xi P) \\ G(r) &= -(\nabla \eta)^{-1} \cdot J^2 Q = -J^2(x_\eta Q + y_\eta Q) \end{aligned} \tag{A.16}$$

and finally

$$\begin{aligned} G(x) &= (\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = -J^2(x_\xi P + x_\eta Q) \\ G(y) &= (\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = -J^2(y_\xi P + y_\eta Q) \end{aligned} \tag{A.17}$$

or in Φ, Ψ form

$$\begin{aligned} \alpha(x_{\xi\xi} + \Phi x_{xi}) - 2\beta x_{\xi\eta} + \gamma(x_{\eta\eta} + \Psi x_\eta) = 0 \\ \alpha(y_{\xi\xi} + \Phi y_{xi}) - 2\beta y_{\xi\eta} + \gamma(y_{\eta\eta} + \Psi y_\eta) = 0 \end{aligned} \tag{A.18}$$

with

$$\begin{aligned} \alpha &= x_{\eta\eta}^2 + y_{\eta\eta}^2 \\ \beta &= x_\xi x_\eta + y_\xi y_\eta \\ \gamma &= x_{\xi\xi}^2 + y_{\xi\xi}^2 \end{aligned} \tag{A.19}$$

APPENDIX B

HIERARCHAL STORAGE OF TWO DIMENSIONAL DATA

Hierarchal storage can be used to store adaptation functions or spacing data to a background x,y plane. For two dimensional mesh generation it is becoming common to use a quadtree to location based data. A quadtree is a tree data structure where each node in the tree has four children. Beginning with the root node, the root node of the tree encapsulates the entire domain and the four children of the root node subdivide the domain into four equal quadrants. Then recursively each quadrant can be subdivided into four, more refined, quadrants. The "Quadtree" C++ implementation of this data structure stores data using an associated extent box. An extent box is the cartesian aligned rectangular region as defined by an upper left point and a lower right point, that represents the location of the data. At any level of the tree, extent boxes that would cross the next quadrant divide are stored at that level of the tree while extent boxes that would be completely encapsulated are stored in lower levels. Figure B.1 shows a domain and the subdivides that would be the quadrants of the next level in the tree. The red extent box crosses the divide that will be the next layer of quadrants and is stored on the current layer of the tree. The blue extent box is completely encapsulated by the subdivide, and is stored in a lower layer. When retrieving data from the quadtree, all the data whose associated extent box contained the query location are returned.

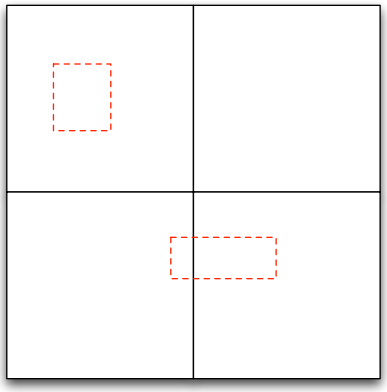


Figure B.1 Extent boxes on a quadrant

Storing data in the quadtree that completely covers a domain cannot easily be done because of the rigid nature of the extent boxes. Storing data from a mesh using the extent box of cells or dualed-cells would cause extent boxes to overlap for all but the simplest structured mesh case. When retrieving data from a location where extent boxes overlap a decision must be made about which extent box best represents the location. Figure B.2 shows two cells and their extent boxes. If the red dot was a query location then both data sets would be returned with no way of differentiating which set more accurately represents the data.

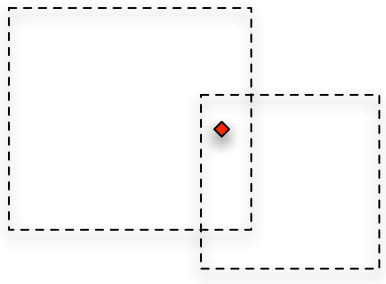


Figure B.2 Overlapping extents

This problem was first solved by Collao [11] by storing the outline of cells or dualed-cells that the extent box encapsulates. When both data sets are returned, the cells can be used to test whether the point is inside our outside each cell. As figure B.3 shows, by adding cells to the information stored the decision on which data set to use can be made.

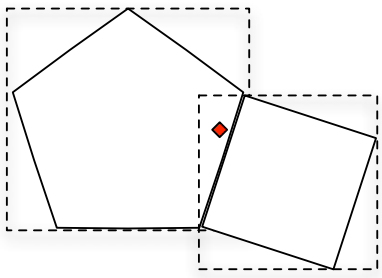


Figure B.3 Overlapping extents with associated cells

VITA

Matthew O'Connell was born in Syracuse, NY. He attended elementary school in New York before relocating to Clarksville, TN where he completed his high school education. His undergraduate degree was completed in the spring of 2009 with a bachelor's degree in Physics from Austin Peay State University. The following fall Matthew enrolled in the M.S. program at the University of Tennessee at Chattanooga SimCenter: National Center for Computational Engineering. Upon completion of the masters program Matthew plans to continue research in mesh generation and adaptation as a Ph.D. student at the SimCenter.