Winslow Elliptic Smoothing Equations Extended to Apply to General Regions of an Unstructured Mesh

A Dissertation Presented for the

Doctor of Philosophy Degree

The University of Tennessee at Chattanooga

James Steven Masters

December 2010

To the Graduate Council:

I am submitting herewith a dissertation written by James Steven Masters entitled "Winslow Elliptic Smoothing Equations Extended to Apply to General Regions of an Unstructured Mesh." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computational Engineering.

[redacted]

Dr. Steve L. Karman, Jr.
Professor,
Computational Engineering
Major Professor

We have read this dissertation
and recommend its acceptance:

[redacted]

Dr. W. Kyle Anderson
Professor,
Computational Engineering

[redacted]

Dr. Timothy W. Swafford
Professor and Head,
Computational Engineering

[redacted]

Dr. John V. Matthew III
Assistant Professor,
Mathematics

[redacted]

Dr. Greg D. Power
Technical Fellow,
Aerospace Testing Alliance
Arnold Engineering Development Center

Accepted for the Council:

[redacted]

Dr. A. Jerald Ainsworth
Dean of the Graduate School

Dedication

To:

Garrett James the Mighty Warrior

Everett Blaze the Courageous Warrior

Ellianna Grace the Beautiful Warrior

And to Sheri Marie Masters, my precious stone and helpmate.

Thank you GOD for my greatest blessings.

Acknowledgements

My deepest thanks go out to all my colleagues and supervisors at the Arnold Engineering Development Center and the Aerospace Testing Alliance. Without their support and flexibility, this dissertation would never have been written. I would especially like to thank my direct supervisors – Tracy Donegan, Bonnie Heikkinen and Dr. Ralph Jones – who worked with me to provide support for a project that could provide PhD-level research opportunities while still enhancing the capabilities of AEDC. Also, thank you to Dr. Greg Power, who was an invaluable resource and ally. The benefit gained from his command of technical issues and willingness to help cannot be underestimated.

I would also like to thank the excellent faculty and staff at the UTC SimCenter. The exceptional instruction that I received while at UTC undoubtedly made me a better engineer. Thank you to Dr. Timothy Swafford for helping me to navigate the university's system and requirements. Thank you to Wally Edmondson for all the systems administration and to Ellen Puffer for all the administrative assistance. Heartfelt thanks go especially to my adviser Dr. Steve Karman; his technical expertise and ability to explain and breakdown issues related to computational meshes is unparalleled. Working with Dr. Karman made my time at UTC both valuable and enjoyable.

Thank you to all of the people, including Dr. Rob McAmis, David Ruckstuhl, Dr. David Elrod and Colonel Arthur Huber, who supported the Accelerated Graduate Degree Program at AEDC. The AGDP provided me with the support and resources necessary to

survive the rigors of being a PhD candidate while simultaneously working as a full time engineer. Their investment into the technical excellence of AEDC through this program will no doubt be felt well into the future of the center.

Thank you to the staff of the AEDC Technical Library for all of their help doing literature searches and tracking down papers that were used for this research. Their dedication to the workforce deserves more recognition than they often receive.

Finally, I would like to thank my incredibly supportive family. Thank you to my wife Sheri Masters for supporting my pursuit of higher education and for all the sacrifices she made to make it work. Thank you to my kids – Garrett, Ellianna and Everett – who are consistently an energizing source of joy and life. Thank you GOD for the blessing of a wonderful and inspiring family and for the favor and direction You bestow.

Abstract

In any engineering endeavor, it is important to have the ability to efficiently and intelligently break up a region of interest in order to explore and investigate the interesting and dynamic aspects of the system enclosed in the region. In the field of fluid mechanics – and, in particular, computational fluid dynamics – this region breakup (known as discretization) has traditionally been done using structured meshes but, because of their flexibility with capturing real-world geometry, unstructured meshes are being increasingly utilized. Not surprisingly, techniques that have traditionally been reserved for structured meshes are migrating to the world of unstructured meshing as well. One such technique is the application of the Winslow elliptic smoothing equations to an unstructured mesh. The Winslow equations have proven to be powerful tools for smoothing unstructured meshes and some early difficulty using the Winslow equations with unstructured meshes was alleviated when it was determined that it was not necessary for the entire computational mesh to be constructed as an overarching system of nodes and elements, but rather, each node in computational space could be treated as an individual virtual control volume. Winslow equations have been shown to be ideal for smoothing non-boundary nodes in inviscid regions but of limited use in other situations. This presented two opportunities to improve and extend the usefulness of the Winslow equations in relation to unstructured meshes. Research documented herein allowed the Winslow equations to be applied to boundary nodes which, traditionally, have been held static while interior mesh points are smoothed. In addition, several methods were tested to make the Winslow equations applicable to highly anisotropic viscous regions of a mesh. The most successful methodology that was developed and explored in relation to a viscous region was to use an iteratively adapted computational space for each of the nodes in the viscous region. This technique allowed a mesh with an amenable connectivity structure to be smoothed in such a way that it could match a desired viscous profile based on an initial off-body spacing and the geometric progression of the viscous layers of the mesh.

Table of Contents

List of Figures

List of Symbols

C_S	Riemannian metric tensor scaling factor
$\vec{d}$	Direction vector about which to reflect a point
d_{AB}	Metric length
d_{lat}	Computational space offset in the lateral direction
d_n	Computational space offset in the normal direction
$\vec{e}_1$	First basis vector for transformed space (associated with normal spacing)
$\vec{e}_2$	Second basis vector for transformed space (associated with lateral spacing)
g	Geometric progression factor
h_1	Scaling associated with normal spacing
h_2	Scaling associated with later spacing
J	Grid Jacobian: $x_\xi y_\eta - x_\eta y_\xi$
$\mathscr{J}$	Inverse Grid Jacobian: $\xi_x \eta_y - \xi_y \eta_x$
M	Riemannian metric tensor
$\vec{n}$	Outward facing normal
$\hat{n}$	Unit length normal vector
$\hat{n}_\xi$	ξ component of the unit normal vector in computational space
$\hat{n}_\eta$	η component of the unit normal vector in computational space
N	Valence count; i.e., the number of connected neighboring nodes
P_A	Average location of interior nodes connected to a boundary node
P_G	Ghost point
P_S	Surface point
P_{S-}	Surface point before P_S

P_{S+}	Surface point after P_S
$\hat{r}$	Edge vector used for refinement
R	Rotation Matrix (Eigenvectors of Riemannian metric tensor)
T	Temperature
w_{11}	X component of local node 1 (central node) on a triangle comprising a control volume
w_{21}	Y component of local node 1 (central node) on a triangle comprising a control volume
w_{12}	X component of local node 2 on a triangle comprising a control volume
w_{22}	Y component of local node 2 on a triangle comprising a control volume
w_{13}	X component of local node 3 on a triangle comprising a control volume
w_{23}	Y component of local node 3 on a triangle comprising a control volume
x_ξ	Derivative of the physical coordinate x with respect the computational coordinate ξ
x_η	Derivative of the physical coordinate x with respect the computational coordinate η
$x_{\xi\xi}$	Second derivative of x: $\partial^2 x / \partial \xi^2$
$x_{\eta\eta}$	Second derivative of x: $\partial^2 x / \partial \eta^2$
$x_{\xi\eta}$	Mixed second derivative of x: $\partial^2 x / \partial \eta \partial \xi$
y_ξ	Derivative of the physical coordinate y with respect the computational coordinate ξ
y_η	Derivative of the physical coordinate y with respect the computational coordinate η
$y_{\xi\xi}$	Second derivative of y: $\partial^2 y / \partial \xi^2$
$y_{\eta\eta}$	Second derivative of y: $\partial^2 y / \partial \eta^2$
$y_{\xi\eta}$	Mixed second derivative of y: $\partial^2 y / \partial \eta \partial \xi$
α	Winslow Coefficient: $x_\eta^2 + y_\eta^2$
β	Winslow Coefficient: $x_\xi x_\eta + y_\xi y_\eta$
Δs	Off-body spacing

ϕ	Generic scalar, equipotential function
γ	Winslow coefficient: $x_\xi^2 + y_\xi^2$
Γ_i	Length of face i for a triangle
Λ	Scaling matrix (Eigenvalues of Riemannian metric tensor)
η	Second component of computational space
λ_1	Length of axis associated with $\vec{e}_1$
λ_2	Length of axis associated with $\vec{e}_2$
ω_z	Vorticity
ξ	First component of computational space
$\vec{\xi}_C$	Point specifically defined to be on a unit circle in computational space
ψ	Stream function

Chapter 1 – Introduction

In any engineering field, it is important to have the ability to break up a given system into manageable parts in order to explore and investigate the interesting and dynamic aspects of the system. This is true for the study of thermodynamic systems or for the use of finite element analysis to study static and dynamic properties of parts and facilities. It is also true within the field of fluid mechanics and especially the sub-discipline of computational fluid dynamics (CFD).

The analysis of aerodynamic systems is often broken down into three separate but complimentary categories: flight test, ground test (wind tunnel testing) and CFD. CFD deals directly with applying the laws of conservation (mass, momentum and energy) to an aerodynamic system. It relies on a logical and effective breakup of the system of interest in order to properly apply the conservative laws of fluid dynamics to the system in a way that properly captures the necessary physics to realistically define the system.

Further delving into the field of computational fluid dynamics is the study of mesh generation. This is the discipline that explores and analyzes the breakup (known as discretization) of a system into finite discrete segments so that the characteristics of the overarching system can be explored using the Navier-Stokes equations (or, in the case of inviscid flow, the Euler equations). This system of equations – named after the Frenchman Claude-Louis Navier and the Englishman Gabriel Stokes – are a system of partial differential equations. Analytical solutions to partial differential equations involve closed-form expressions which give the variation of the dependent variables continuously

throughout a domain [1]. However, to analyze a physical system computationally, both the defining equations and the domain must be discretized. Discretization of the equations is accomplished by replacing the original system of partial differential equations with a system of algebraic equations that can be solved to calculate the values of the flow-field variables (pressure, temperature, velocity etc.) at discrete points (these points are often referred to as grid points, mesh points or nodes). Depending on the way the equations are discretized, a finite difference or finite volume numerical solution is obtained. In contrast to differential equations, numerical solutions generally give answers only at discrete points in the domain; effectively generating these discrete grid point locations is at the heart of mesh generation.

There are many ways that a region where the flow is to be analyzed might be discretized. Traditionally, the region has been discretized using structured meshes, where the discretization of the domain reflects some type of consistent geometrical regularity. However, because of their flexibility with capturing real-world geometry, unstructured meshes (where mesh points are placed in the flow field in a very irregular fashion) are being utilized more and more regularly [1]. As computational power and storage increase and as unstructured meshes are becoming better understood and handled with increasing skill, techniques that have, in the past, been reserved for structured meshes are migrating to the world of unstructured meshing as well. One such technique is the application of the Winslow elliptic smoothing equations to a mesh.

1.1 – Mesh Smoothing and Adaptive Mesh Refinement

Mesh smoothing, which is the primary topic of this dissertation, falls loosely into the field of study known as Adaptive Mesh Refinement (AMR). AMR is employed with the purpose of reducing the computational cost of generating a flow solution while maintaining a given level of accuracy for the solution. In a finite element context, adaptive mesh refinement is generally classified into three categories: h-refinement (enrichment), r-refinement (movement) and p-refinement (reconnection).

H-refinement schemes often use an error estimator to determine regions of the grid where the solution is under-resolved and add elements locally to improve the resolution. This is done with the intent of improving the accuracy of the solution. While h-refinement schemes have demonstrated the ability to improve solution accuracy through local refinement, there is added overhead due the increased number of mesh elements [2]. The p-refinement approach increases the element order by increasing the degree of the piecewise polynomials over the triangular mesh elements. The r-refinement approach involves the redistribution of points. R-refinement involves moving the nodes of the mesh while maintaining the cell connectivity and the size of the mesh [3]. R-refinement is the category that includes mesh smoothing. Conceptually, r-refinement is relatively straightforward but it can produce highly skewed cells and prohibitively small (or negative) cell volumes if not performed correctly. AMR generally refers to adapting a mesh to a particular flow solution. The primary research performed here deals only with adapting a mesh to changes in geometry.

1.2 – Winslow Elliptic Smoothing Equations

The Winslow elliptic smoothing equations, first proposed by Alan Winslow in 1967 [4], are derived from Poisson's Equation (or Laplace's Equation for the homogeneous case) for a parameter distribution over a region. In structured meshing there is an implicit computational space that lends itself well to the application of the Winslow equations. However, no such implicit computational space exists for unstructured meshes and the computational space must, therefore, be explicitly defined. It can be shown that a computational space can be constructed using the initial discretized physical space. This works well for many applications and often provides a well-defined connectivity and spatial definition for a mesh that is applicable even if the geometry that defined the mesh was then allowed to move. However, this also creates the requirement that an initial valid mesh exists. This is not always the case; an initial unstructured mesh might not exist and, if one does exist, it is possible that there might exist sections in the mesh that are invalid due to mesh crossing, negative volumes or other problems. Using the original mesh as the computational space is also problematic if there is a requirement that certain mesh characteristics (such as viscous spacing) need to be varied.

It is apparent that there are significant limitations with using the original mesh as the computational space but a mathematically expedient way of defining a general overarching unstructured computational space has proven elusive. Because the unstructured mesh connectivity needs to be explicitly defined and the valence count (the number of connected nodes or neighbors) varies throughout the domain, no theory currently exists to generate a global computational space that reflects a consistent geometric regularity as was done with structured meshes. Many difficulties that were due

to the lack of a global unstructured computational space were alleviated when a breakthrough was made [5][6] that showed it was not necessary for the entire computational mesh to be constructed as an overarching system of nodes and elements but rather, each node in computational space could be broken off from the overarching system and coupled only though the coordinates in physical space, which were treated as free variables.

The simplest way to discretize a physical region using structured meshes is to break up the area into quadrilaterals (quads) in two dimensions and hexahedra (hexes) in three dimensions. The simplest way to discretize a physical region using unstructured meshes is to break up the area into triangles in two dimensions and tetrahedra (tets) in three dimensions. There are many other ways to discretize a region as well but this dissertation will only address the simplicial geometry elements. Examples of a simple two-dimensional region that has been discretized using both structured and unstructured methodologies, as well as the corresponding computational spaces, are shown on Figure 1 and Figure 2. The convention used within this document is that physical space is generally displayed with a red mesh while computational space is generally displayed with a black or gray mesh; this convention is illustrated in Figure 1 and Figure 2. Additional information concerning grid metrics, which describe the relationship (or mapping) between physical and computational space is assembled in Appendix A.

When applied to an unstructured mesh, the Winslow elliptic smoothing equations allow mesh points to be moved and smoothed to conform to a moving (or non-moving) surface. The Winslow equations deal with derivatives of physical space (x,y) with respect to computational space (ξ,η). A detailed derivation of the Winslow equations is

demonstrated in Appendix B and it is interesting to note how much more complex the equations become when they are transformed from the relatively simple form with respect to coordinates in physical space compared to the form where the derivatives are with respect to the computational coordinates.

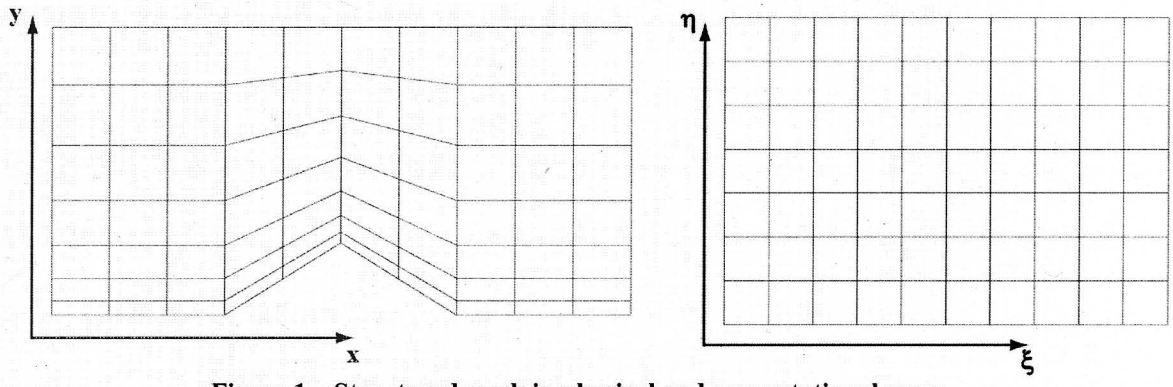


Figure 1 – Structured mesh in physical and computational space.

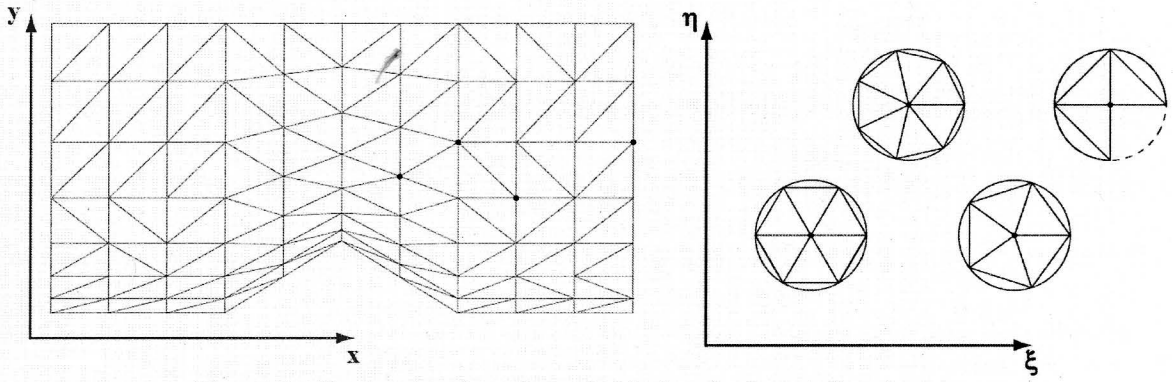


Figure 2 – Unstructured mesh in physical and computational space.

Note that, in unstructured computational space, the nodes, which correspond to the four nodes shown as black dots in physical space, are broken off from the overarching system and are coupled only through the coordinates in physical space. Each node will have a unique computational space similar to the ones shown in the figure. Note that the computational space for a boundary node does not have a complete neighbor-node stencil.

1.3 – Extension of the Winslow Equations

Note from the unstructured computational space shown on Figure 2 that the central node in each of the computational spaces is surrounded by neighbors using equal angles and equal edge-lengths and that it is necessary to be fully surrounded by neighboring nodes for the computational stencil to be complete. (A note on terminology: the node-of-interest – i.e. the node being smoothed at a given instant – will often be referred to in this document as the central node and is always located at the origin in computational space.) The characteristics of the Winslow equations make them ideal for smoothing inviscid mesh elements in the interior region of a mesh because the equations drive the mesh elements to display an isotropic behavior; that is, the triangular mesh elements in physical space will all become as close to equilateral as possible within the constraints of the overall system.

However, the need for a complete computational neighbor-node stencil and the tendency to drive mesh elements to exhibit isotropic behavior makes the conventional Winslow methodology unsuitable to two situations that are common in fluid mechanics. The first situation is where the nodes on the boundaries need to move to accommodate the movement of the interior mesh nodes. This situation may arise because, often, keeping the boundary nodes static while interior nodes move will result in elements that exhibit a sufficient amount of skew that the numerical solutions generated on a mesh will become unstable or unreliable. The second situation is one where the viscous properties of the flow are important. In order to capture the viscous boundary layer in an effective manner, it is necessary that the mesh elements near the viscous surface are not isotropic but rather have high aspect ratios in order capture the gradients in the flow where the

flow field variables are changing very rapidly normal to the surface but much more mildly in the direction parallel to the surface.

These two common situations are addressed in detail within this dissertation. A methodology is presented and described that will make it possible to apply the Winslow elliptic smoothing equations to a node on a boundary. Implementation of this methodology will allow the boundary nodes to float and greatly improve the mesh in certain situations. Several examples are shown that illustrate the benefit of implementing the Winslow equations is such a way that the surface nodes are allowed to float.

Several methodologies are also presented that allow the Winslow elliptic smoothing equations to be applied to viscous regions of a mesh, where it is required that the mesh be highly anisotropic. These different methodologies are compared for effectiveness and several examples are shown that illustrate the benefit of applying the Winslow equations is such a way that viscous properties of the mesh are preserved.

Chapter 2 – Background Methodology

2.1 – Divergence Theorem

The gradient in an unstructured element is derived by manipulating the Divergence Theorem (often alternately referred to as the Gauss Divergence Theorem or, in 2D, as Green's Theorem). The Divergence Theorem states that if there is a solid region (i.e., a control volume) whose boundary surface has positive orientation (outward facing surface normals, $\vec{n}$) and let $\vec{F}$ be a vector field whose component functions have continuous partial derivatives on an open region containing the control volume [7], then:

$$\iiint \nabla \cdot \vec{F}\, dV = \oiint \vec{F} \cdot \vec{n} dS \tag{2.1}$$

Note that the equation above is for a three-dimensional control volume. The majority of the research that was performed for this dissertation was limited to 2 dimensions; this required that the divergence theorem be simplified to two dimensions, which is done in the following section. Because much of the work performed herein was proof-of-concept level work, it made sense to test the theory in 2 dimensions and work out the details of the algorithms and logic in a 2D environment where the results could be viewed more quickly and efficiently. However, many concepts were also expanded to three dimensions as it is the intention of the author to extend this work to 3D in postdoctoral work. Certain concepts that have been extended to 3D can be found in Appendix C.

2.2 – Calculating Gradients

In two dimensions, the Divergence Theorem, which is shown in the previous section can be simplified to equation (2.2).

$$\iint \nabla \cdot \vec{F}\, dA = \oint \vec{F} \cdot \vec{n} dS \tag{2.2}$$

The control volume in 2D is effectively a "control area" but will continue to be referred to as a control volume herein. Consider a two dimensional domain broken up into an unstructured mesh comprised of triangles similar to the one shown below in Figure 3. Each triangle becomes a control volume and the gradient within the control volume (which is treated as constant throughout the control volume and discontinuous at the surface of the control volume) can be found by manipulating the Divergence Theorem. Note that because the gradient is constant within the control volume, the mesh must be refined into smaller control volumes if a more precise gradient is desired in a given region.

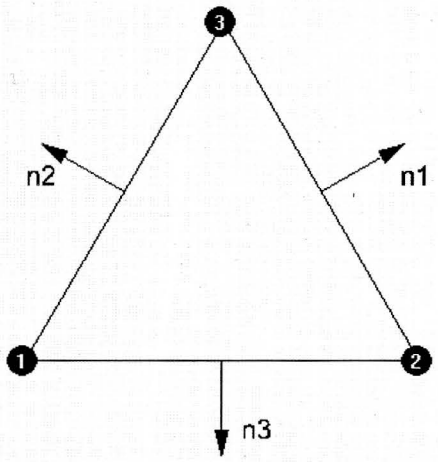


Figure 3 – Two-dimensional triangular control volume with outward facing normals.

A generic scalar function (denoted as ϕ) is used to illustrate the process for calculating the gradient in a triangular control volume. Any scalar function that is continuous in the region can be calculated using the following methodology.

The gradient of a scalar function is a vector function. In two dimensions, this gradient can be written:

$$\nabla\phi = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix}$$

The averages of the components of the gradient in a 2D control volume can be calculated as:

$$\begin{aligned} \overline{\frac{\partial\phi}{\partial x}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial x}\,dA \\ \overline{\frac{\partial\phi}{\partial y}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial y}\,dA \end{aligned} \tag{2.3}$$

Consider a vector, $\vec{F}$; the divergence of the vector, $\nabla\cdot\vec{F}$, is a scalar and can be related to either of the two components of the two dimensional gradient as follows:

$$\vec{F} = \begin{bmatrix} \phi \\ 0 \end{bmatrix} \Rightarrow \nabla\cdot\vec{F} = \frac{\partial\phi}{\partial x} + 0 = \frac{\partial\phi}{\partial x}$$

$$\vec{F} = \begin{bmatrix} 0 \\ \phi \end{bmatrix} \Rightarrow \nabla\cdot\vec{F} = 0 + \frac{\partial\phi}{\partial y} = \frac{\partial\phi}{\partial y}$$

$$\iint \frac{\partial\phi}{\partial x}\,dA = \underbrace{\iint \nabla\cdot\vec{F}\,dA = \oint \vec{F}\cdot\vec{n}\,dS}_{\text{Divergence Thm}} = \oint \begin{bmatrix} \phi \\ 0 \end{bmatrix}\cdot\begin{bmatrix} n_x \\ n_y \end{bmatrix} dS = \oint \phi\, n_x\, dS$$

$$\iint \frac{\partial\phi}{\partial y}\,dA = \underbrace{\iint \nabla\cdot\vec{F}\,dA = \oint \vec{F}\cdot\vec{n}\,dS}_{\text{Divergence Thm}} = \oint \begin{bmatrix} 0 \\ \phi \end{bmatrix}\cdot\begin{bmatrix} n_x \\ n_y \end{bmatrix} dS = \oint \phi\, n_y\, dS$$

So, the components of the average gradient in a control volume can be calculated as:

$$\overline{\frac{\partial \phi}{\partial x}} = \frac{1}{A}\iint \frac{\partial \phi}{\partial x} dA = \frac{1}{A}\oint \phi n_x \, dS$$

$$\overline{\frac{\partial \phi}{\partial y}} = \frac{1}{A}\iint \frac{\partial \phi}{\partial y} dA = \frac{1}{A}\oint \phi n_y \, dS$$

Dropping the overbar for simplicity, with the understanding that the gradient is treated as constant within the control volume, the equation for the gradient becomes:

$$\nabla \phi = \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{1}{A}\oint \phi n_x \, dS \\ \frac{1}{A}\oint \phi n_y \, dS \end{bmatrix} \tag{2.4}$$

The surface integrals can be discretized by examining and summing the scalar for which a gradient is desired on each of the triangle's 3 faces (edges in 2D). Using the x-component for example, the gradient is discretized as follows.

$$\frac{1}{A}\oint \phi n_x \, dS \approx \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i \hat{n}_{xi} \Gamma_i \tag{2.5}$$

Where: $\bar{\phi}_i$ = the average value of the scalar over the edge, calculated as the average of the value of the two nodes that makes up the face or as the average of the value at the two cell centers on either side of the face. It will be shown that using the nodal values adds simplicity to the final equations.

$\hat{n}_{xi}$ = x component of the surface unit-normal vector for face i. The unit-normal is normalized by the edge length, Γ_i

Γ_i = length of face i

A non-hatted n will indicate a non-unit normal vector component, which is found using equation (2.6).

$$\hat{n}_{xi} = \frac{n_{xi}}{\Gamma_i} \Rightarrow n_{xi} = \hat{n}_{xi}\Gamma_i$$
$$\hat{n}_{yi} = \frac{n_{yi}}{\Gamma_i} \Rightarrow n_{yi} = \hat{n}_{yi}\Gamma_i \tag{2.6}$$

The normal vectors for each of the three edges are calculated as follows:

$$\begin{aligned} n_{x1} &= y_3 - y_2 & n_{y1} &= -(x_3 - x_2) \\ n_{x2} &= y_1 - y_3 & n_{y2} &= -(x_1 - x_3) \\ n_{x3} &= y_2 - y_1 & n_{y3} &= -(x_2 - x_1) \end{aligned} \tag{2.7}$$

The value of the gradient on each edge will be treated as the average of the value at the two nodes that define the edge (note from Figure 3 that the edge number corresponds to the node opposite the edge) and will be designated by an overbar.

$$\bar{\phi}_1 = \frac{1}{2}(\phi_2 + \phi_3)$$
$$\bar{\phi}_2 = \frac{1}{2}(\phi_1 + \phi_3)$$
$$\bar{\phi}_3 = \frac{1}{2}(\phi_1 + \phi_2)$$

Using the numbering convention from Figure 3, we get the following equation.

$$\frac{\partial \phi}{\partial x} = \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i \hat{n}_{xi} \Gamma_i = \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i n_{xi} = \frac{1}{A}\left(\frac{1}{2}(\phi_2 + \phi_3)n_{x1} + \frac{1}{2}(\phi_3 + \phi_1)n_{x2} + \frac{1}{2}(\phi_1 + \phi_2)n_{x3}\right)$$

Rearranging this equation generates equation (2.8):

$$\frac{\partial \phi}{\partial x} = \frac{1}{2A}\left((\phi_2 + \phi_3)n_{x1} + (\phi_3 + \phi_1)n_{x2} + (\phi_1 + \phi_2)n_{x3}\right)$$
$$\frac{\partial \phi}{\partial y} = \frac{1}{2A}\left((\phi_2 + \phi_3)n_{y1} + (\phi_3 + \phi_1)n_{y2} + (\phi_1 + \phi_2)n_{y3}\right) \tag{2.8}$$

Rearranging equation (2.8) gives the most common form for the gradient component equations.

$$\frac{\partial \phi}{\partial x} = \frac{1}{2A}\left((n_{x2} + n_{x3})\phi_1 + (n_{x3} + n_{x1})\phi_2 + (n_{x1} + n_{x2})\phi_3\right)$$
$$\frac{\partial \phi}{\partial y} = \frac{1}{2A}\left((n_{y2} + n_{y1})\phi_1 + (n_{y3} + n_{y1})\phi_2 + (n_{y1} + n_{y2})\phi_3\right) \tag{2.9}$$

Using the fact that the normals sum to zero (which can be easily seen from (2.7)) equation (2.9) simplifies to equation (2.10) and the derivatives from (2.10) are used to form the 2D gradient of the scalar, which is shown on equation (2.11).

$$\frac{\partial \phi}{\partial x} = \frac{1}{2A}\left(-n_{x1}\phi_1 - n_{x2}\phi_2 - n_{x3}\phi_3\right)$$
$$\frac{\partial \phi}{\partial y} = \frac{1}{2A}\left(-n_{y1}\phi_1 - n_{y2}\phi_2 - n_{y3}\phi_3\right) \tag{2.10}$$

$$\nabla\phi = \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \end{bmatrix} = -\frac{1}{2A}\begin{bmatrix} n_{x1}\phi_1 + n_{x2}\phi_2 + n_{x3}\phi_3 \\ n_{y1}\phi_1 + n_{y2}\phi_2 + n_{y3}\phi_3 \end{bmatrix} \tag{2.11}$$

2.3 – Laplace and Poisson's Equation

A large segment of the research presented herein deals with the Winslow elliptic smoothing equations, which are derived in detail in Appendix B. The Winslow equations are used to ensure smoothness in a computational mesh and are derived by applying the Laplacian (with respect to the physical mesh coordinates) to the computational

coordinates. Because this system of equations is based on Laplace's equation (for a homogenous system) or Poisson's equation (for a nonhomogeneous system) it is interesting to compare it with other systems that follow the same laws.

Some other systems where the Laplacian is used are the steady state heat equation $\left(\nabla^2 T = 0\right)$ and the velocity potential equation. When a flow is steady, irrotational and isentropic, the Euler equations can be simplified into the potential velocity equation and it can be observed that $\nabla^2\psi = 0$ and $\nabla^2\phi = 0$ where lines of constant ψ are known as streamlines and lines of constant ϕ are known as equipotential lines. In this situation, ϕ is the equipotential function and ψ is the so-called stream function, defined so as to satisfy continuity identically [8].

The equipotential function and stream function is developed by looking at the continuity equation for incompressible flow.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.12}$$

The equipotential function, ϕ, is defined such that $\frac{\partial \phi}{\partial x} = u$ and $\frac{\partial \phi}{\partial y} = v$. Plugging these relationships into equation (2.12) gives Laplace's equation for ϕ: $\nabla^2\phi = 0$. If the stream function, ψ, is defined such that $u \equiv \frac{\partial \psi}{\partial y}$ and $v \equiv -\frac{\partial \psi}{\partial x}$ then the continuity equation is identically satisfied:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial x \partial y} \equiv 0$$

This relationship can now be plugged into the equation for vorticity (ω_z) and, if the flow is irrotational, we again get Laplace's equation:

$$-\left(\frac{\partial u}{\partial y}-\frac{\partial v}{\partial x}\right)=\omega_z \tag{2.13}$$

$$\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial x^2}=-\omega_z$$

$$\nabla^2\psi=-\omega_z$$

$$\nabla^2\psi=0 \tag{2.14}$$

It is interesting to examine a flowfield generated using the velocity potential equations, such as the one shown on Figure 4, and note that the lines of constant ψ and constant ϕ form a system similar to what would be expected when generating a mesh that is smoothed using the elliptic smoothing equations (also based on Laplace's equations).

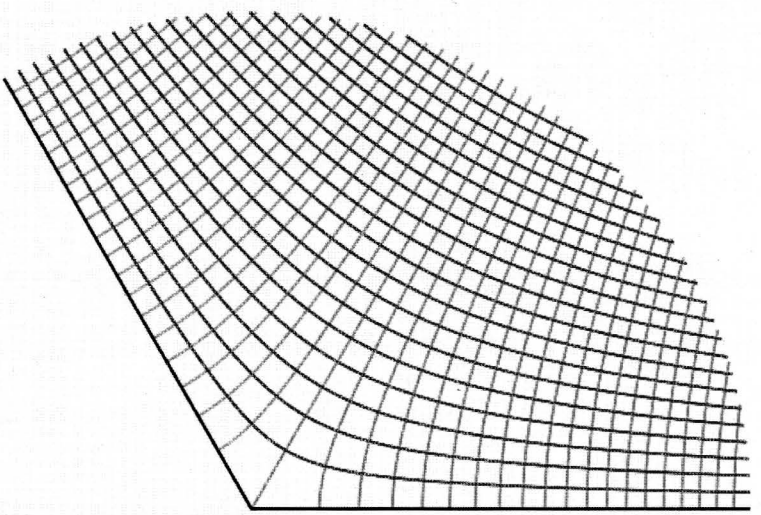


Figure 4 – Flowfield generated using the velocity potential equation.
In the equation above, the darker blue lines are streamlines, and the lighter blue lines are equipotential lines.

2.4 – Laplacian Smoothing vs. Winslow Smoothing

Prior to work done by Knupp, smoothing techniques on unstructured meshes were predominated by Laplacian smoothing because of its generality and ease of implementation. However, Knupp felt that the additional work of implementing the Winslow elliptic smoothing equations was worth the effort because of the robustness against grid folding that is achieved by using the Winslow equations [9].

In Laplacian smoothing, node positions are determined by solving the Laplacian of the physical coordinates with respect to the computational coordinates. In two dimensions, this becomes:

$$\begin{aligned} \nabla^2 x &= \frac{\partial^2 x}{\partial \xi^2} + \frac{\partial^2 x}{\partial \eta^2} = 0 \\ \nabla^2 y &= \frac{\partial^2 y}{\partial \xi^2} + \frac{\partial^2 y}{\partial \eta^2} = 0 \end{aligned} \tag{2.15}$$

This is easy to implement because the position of a node can be solved as the average of the positions of the N neighboring nodes:

$$\vec{x}_n = \frac{1}{N} \sum_{m=0}^{N-1} \vec{x}_m \tag{2.16}$$

Although Laplacian smoothing is easy to implement, its usefulness is limited by the fact that it sometimes results in mesh folding/spillover and no mathematical guarantee against such folding can be constructed [9].

The Winslow equations are obtained by requiring that the computational coordinate variables (ξ and η) be harmonic functions and then interchanging the dependent and independent variables in the corresponding Laplace equations. That is, we now deal with

Laplacians of the computational space with respect to physical space, as shown in equation (2.17).

$$\nabla^2\xi = \frac{\partial^2\xi}{\partial x^2} + \frac{\partial^2\xi}{\partial y^2} = 0$$
$$\nabla^2\eta = \frac{\partial^2\eta}{\partial x^2} + \frac{\partial^2\eta}{\partial y^2} = 0 \qquad (2.17)$$

The derivation of the Winslow equations from the above Laplace equations (equation (2.17)) is discussed in detail in Appendix B but it is worth noting that the technique that was used by Knupp was to let a local discrete uniform computational space for a node with a valence count of N be given by equation (2.18) and then letting the physical coordinates of the nodes be a function of the computational space coordinates: $x = x(\xi,\eta)$ and $y = y(\xi,\eta)$.

$$\xi_m = \cos\theta_m$$
$$\eta_m = \sin\theta_m \qquad (2.18)$$

where: $$\theta = \frac{2\pi m}{N}$$

By assuming that there exists smooth functions, $x = x(\xi,\eta)$ and $y = y(\xi,\eta)$, on the local computational space and that these functions can be approximated about the origin by a Taylor series expansion, the expansion can be used to approximate the first and second derivates of x an y that are needed to construct the necessary coefficients of the Winslow equations.

2.5 – Computational Mesh Derived from Connectivity

Early in the research process, a small amount of time was taken to examine the possibility of developing an algorithm in pursuit of a connectivity-based meshing code that could generate an unstructured global mesh using only the number of grid points and node connectivity information (i.e. no coordinate or boundary information would be provided). The reason why it was thought that this capability could prove valuable was that when a moving-body problem is being studied, the grid may become invalid once one of the boundaries (i.e. one of the moving bodies) has moved from its initial location. A connectivity-based meshing technique could have interesting implications for quickly creating an arbitrary but valid unstructured mesh that could then be used as the computational space in which to perform elliptic mesh smoothing. As mentioned in the introduction, elliptic smoothing has historically been used to construct high-quality structured grids but, because of the lack of an implicit unstructured computational space, it has traditionally been much less prevalent in the unstructured field. The ability to quickly generate a valid mesh that could be used as computational space, therefore, seemed worth investigating.

The code that was developed read in the connectivity information and then (using only connectivity information) determined which nodes were boundary nodes and how many levels of connectivity existed in the mesh (the boundary was set to level 0, all nodes connected to a boundary node were on level 1, etc.). The code then broke up the interior and used a space marching scheme to march inward until all of the node positions had been defined in the region. Initially, the marching algorithm made the assumption that there was only one boundary. The results for this were tested on a small mesh

(Figure 5) and the algorithm was then made more general to include the possibility of multiple boundaries. Different options were also added to handle cases where the positions of one or more boundaries are specified.

Additional cases with multiple boundaries were also tested and some of these results are shown on Figure 6. An interesting issue that arose stemmed from the fact that, for an unstructured grid with, say, a farfield at the outer boundary and an airfoil on the inner boundary, the number of connectivity levels between the outer boundary and the inner boundary can vary greatly at different locations on the boundary. One reason for this could be that the grid spacing on the inner boundary could vary by an order of magnitude or more from the leading edge, where high curvature requires fine spacing to capture the shape of the airfoil, to an area closer to the trailing edge where the airfoil has much less curvature and would, therefore, not require such a fine mesh.

For simple cases such as those that are shown on Figure 5 and Figure 6, variable connectivity levels do not cause significant problems. However, for larger cases, variable connectivity levels will cause grid crossing if a constant radius is used for each of the connectivity levels. To alleviate this problem, experiments were performed with ways to make the radius of the connectivity level be a function of position and thus morph the level to better fit the local behavior of the mesh.

After exploring the connectivity-based mesh generation scheme and dealing with all of the difficulties that arose, it was determined that a better way to deal with the computational space for an unstructured mesh was to use the decoupled computational space shown on Figure 2. By decoupling the nodes in computational space, the only

information that is needed to construct the computational space is connectivity information, which was essentially the goal of the connectivity based global algorithm. Using a decoupled computational space, coordinate information is implied based on the way the computational space is implemented and the valence count of each node. The computational space can then be modified based on the requirements of the physical mesh, as is discussed in the following chapters.

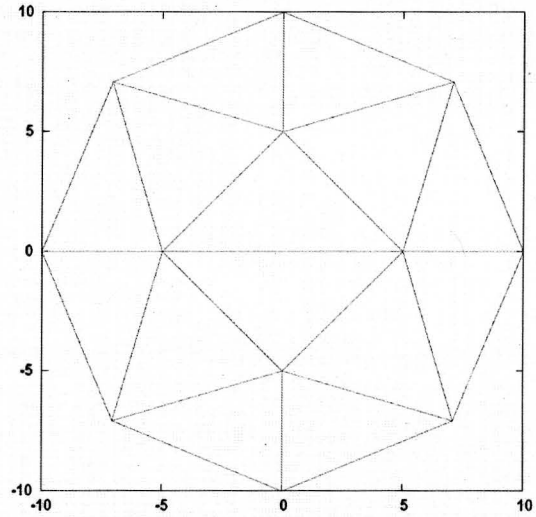


Figure 5 – Single boundary simple connectivity-based mesh.

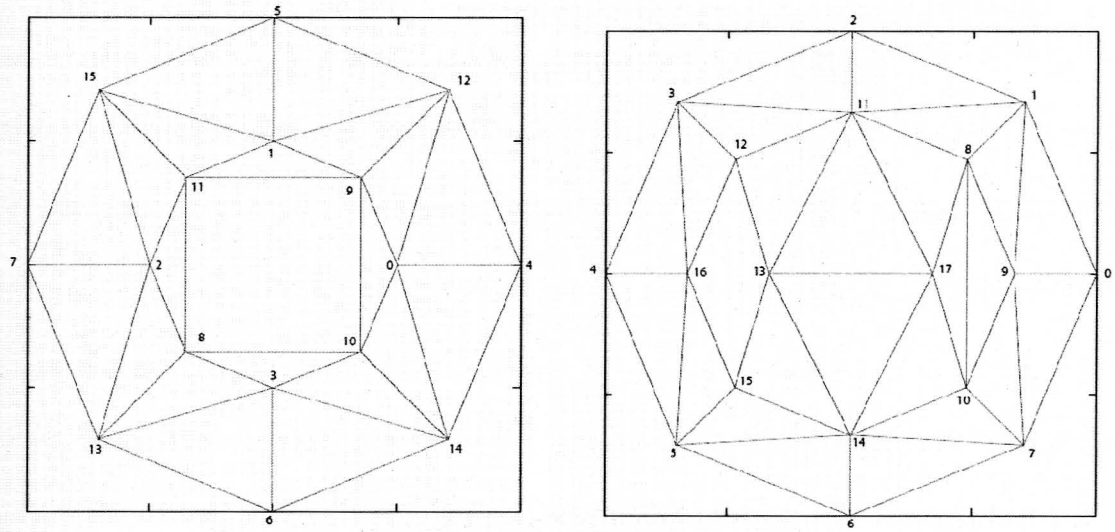


Figure 6 – Multiple boundary cases.

Chapter 3 – Isotropic Winslow Smoothing

3.1 – Elliptic Smoothing

Winslow smoothing is the term generally used to describe the method of elliptic mesh smoothing based on manipulating a mesh (structured or unstructured) using the Winslow elliptic smoothing equations. The Winslow equations, which are derived in detail in Appendix B, are found by taking the Laplacian of the computational space coordinates with respect to physical space.

$$\begin{aligned} \nabla^2 \xi &= 0 \\ \nabla^2 \eta &= 0 \end{aligned} \tag{3.1}$$

The Laplacian follows the Min/Max principle, which says that given a scalar field over a region R that satisfies Laplace's equation $\nabla^2 \phi = 0$, the value of a non-constant scalar, ϕ, cannot attain its maximum or minimum in the interior. This can be deduced from the fact that Laplace's equation, which is said to be harmonic, has the property that the average value over a spherical surface is equal to the value at the center of the sphere (Gauss's harmonic function theorem) [10]. Suppose that we wish to solve Laplace's equation in any region R. Consider any point p inside R and a circle of any radius r_0 (such that the circle is inside R). Let the value on the circle be $f(\theta)$. If Laplace's equation holds, the value at any point is the average of the values along any circle of radius r (lying inside R) centered at that point. The proof is by contradiction. Suppose the maximum or minimum was at point p as illustrated in Figure 7. The value at point p should be the average of all points on any surrounding circle, such as the one with radius

r_0; it is impossible for the value at p to be larger or smaller than the maximum or minimum surrounding point.

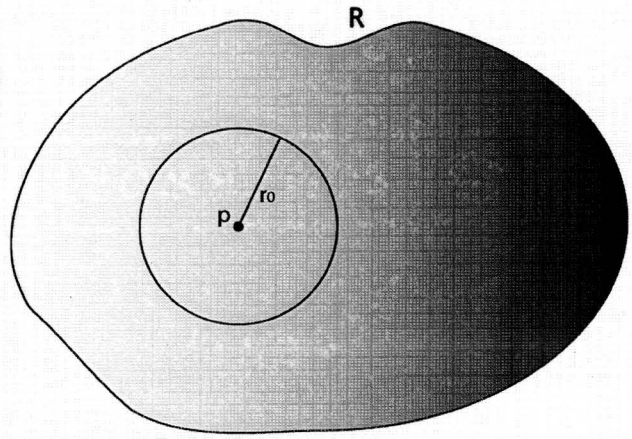


Figure 7 – Region surrounding a control volume.

The Winslow Equations, which are at the heart of the elliptic mesh smoothing that is examined throughout the remainder of this document, are derived by transforming the Laplacian of the computational space coordinates with respect to physical coordinates such that the equations are now with respect to computational coordinates and the physical coordinates are now the dependent variables. This transformation is necessary because it is generally the physical coordinates that will be modified using elliptic smoothing. The coordinates of the computational space need to be defined (either implicitly or explicitly) and can then be further manipulated to affect the physical mesh as required. The Winslow equations, in the form most relevant to the work performed herein, are shown below as equation (3.2).

$$\alpha\frac{\partial^2 x}{\partial\xi^2}-2\beta\frac{\partial^2 x}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 x}{\partial\eta^2}=0$$
$$\alpha\frac{\partial^2 y}{\partial\xi^2}-2\beta\frac{\partial^2 y}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 y}{\partial\eta^2}=0 \qquad (3.2)$$

where:

$$\alpha = x_\eta^{\,2} + y_\eta^{\,2}$$
$$\beta = x_\xi x_\eta + y_\xi y_\eta \qquad (3.3)$$
$$\gamma = x_\xi^{\,2} + y_\xi^{\,2}$$

3.2 – Virtual Control Volumes

In order to explore the computational space of a two dimensional unstructured mesh, consider a small subsection of a mesh comprised of triangles such as that shown on Figure 8. There are three internal nodes (marked as: ●●●) in the mesh subsection shown on Figure 8. Unlike structured grid methods, unstructured methods have not traditionally had a well-defined computational space associated with the physical space. While a global unstructured computational space remains elusive, nodes in an unstructured mesh can be mapped to an individualized computational space based on virtual control volumes decoupled from one another in computational space (they remain loosely coupled through their relationship in physical space). The computational space that is suggested by Karman et al [11] is one in which the node in question lies at the center of a virtual control volume based on a regular (equiangular) polygon whose vertices intersect a unit circle. For each of the internal nodes shown in physical space on Figure 8, the corresponding computational space can be seen on Figure 9. Defining the computational space in this way lends itself well to solving the Winslow equations (especially in inviscid regions where an isotropic mesh is desired) and allows the equations to be solved even if the physical grid is inextricably tangled.

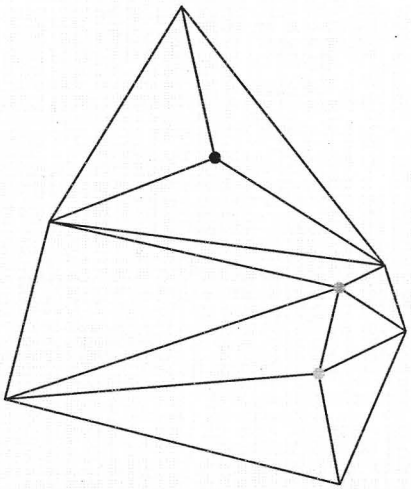


Figure 8 – Subsection of an unstructured mesh.

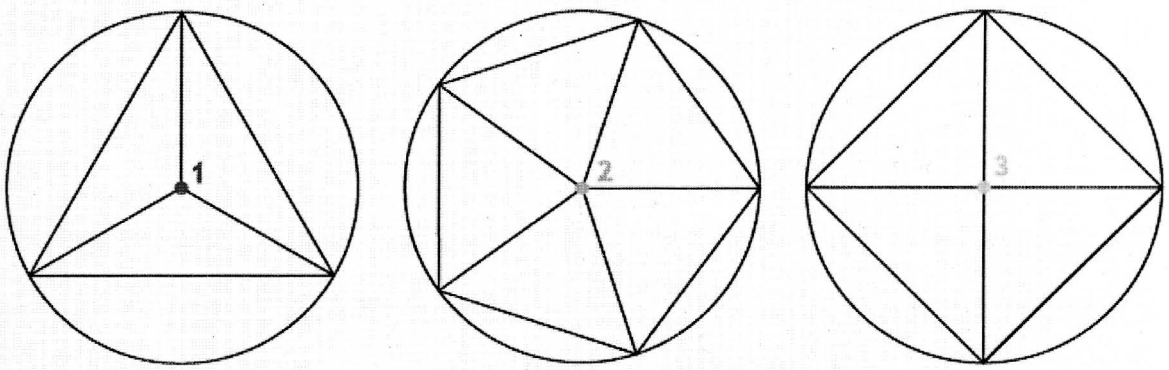


Figure 9 – Virtual control volumes in computational space.
The Computational space (ξ–η space) is based on the valence count of the node in physical space, which is mapped to the center of a regular polygon whose vertices intersect a unit circle. The nodes in this figure are color-coordinated to the nodes in physical space in Figure 8.

3.3 – Discrete Form of the Winslow Equations

The discretization of the Winslow equations will focus on the Laplace form of the equations (i.e. the homogeneous equations without any forcing functions). The justification for this is that a desired grid spacing that might incline a user to use the Poisson form of the equations can also be generated by manipulating the computational space so forcing functions become superfluous. The computational space manipulation that allows the homogeneous form of the Winslow equations to be used exclusively is

described in detail in many places in subsequent chapters but a particularly illuminating description is in section 6.6.4. The Laplace form of the Winslow equations for x and y (using condensed notation for the derivatives) are $\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = 0$ and $\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = 0$. Because the Winslow Equations are cast in computational space, the gradient operator (∇) will be defined such that when operating (through the dot product) on a vector in computational space such as $\vec{F} = \begin{bmatrix} f_\xi \\ f_\eta \end{bmatrix}$, the results are as follows:

$$\nabla \cdot \vec{F} = \frac{\partial}{\partial \xi} f_\xi + \frac{\partial}{\partial \eta} f_\eta \tag{3.4}$$

The following logic is used to get the Winslow equations in a form that can be used to solve the overall global system consisting of all nodes in a region that are to be smoothed. (Note that the physical coordinate in the x direction is used in the following derivation but that the derivation, of course, also applies to the gradients of y with respect to ξ and η as well.)

$$\begin{aligned}
\vec{F} = \begin{bmatrix} x_\xi \\ 0 \end{bmatrix} &\Rightarrow \nabla \cdot \vec{F} = \frac{\partial}{\partial \xi}\left(x_\xi\right) + 0 = x_{\xi\xi} \Rightarrow \vec{F} \cdot \hat{n} = x_\xi \hat{n}_\xi \\
\vec{F} = \begin{bmatrix} x_\eta \\ 0 \end{bmatrix} &\Rightarrow \nabla \cdot \vec{F} = \frac{\partial}{\partial \xi}\left(x_\eta\right) + 0 = x_{\xi\eta} \Rightarrow \vec{F} \cdot \hat{n} = x_\eta \hat{n}_\xi \\
\vec{F} = \begin{bmatrix} 0 \\ x_\eta \end{bmatrix} &\Rightarrow \nabla \cdot \vec{F} = 0 + \frac{\partial}{\partial \eta}\left(x_\eta\right) = x_{\eta\eta} \Rightarrow \vec{F} \cdot \hat{n} = x_\eta \hat{n}_\eta
\end{aligned} \tag{3.5}$$

Consider the Winslow equations in a control volume:

$$\iint_A \left(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta}\right) dA = 0$$
$$\iint_A \left(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right) dA = 0$$

The logic illustrated in (3.5) combined with the divergence theorem (2.2) generates the homogeneous Winslow equations in terms of the surface integral over the surface of the control volume.

$$\oint \left(\alpha x_\xi \hat{n}_\xi - 2\beta x_\eta \hat{n}_\xi + \gamma x_\eta \hat{n}_\eta\right) dS = 0$$
$$\oint \left(\alpha y_\xi \hat{n}_\xi - 2\beta y_\eta \hat{n}_\xi + \gamma y_\eta \hat{n}_\eta\right) dS = 0 \qquad (3.6)$$

Consider a node with a valence count of four, such as the node represented as ● in Figure 8. The virtual control volume for this node is shown on Figure 10 where t_1 through t_4 are the non-unit normal vectors for the surface of the overall virtual control volume. As will become clear when the equations are further expanded, t is used instead of the more typical n when describing the overall surface normal vectors because these vectors need to be distinguished from the normal vectors of the individual triangular elements that comprise the control volume (which are used to calculate the gradients in the control volume).

On Figure 11, the control volume from Figure 10 is shown exploded into the triangular sectors that make up the overall control volume. The non-unit normal vectors for each of the triangles comprising a control volume are illustrated in Figure 11, which elucidates the distinction between the normal vectors of the overall control volume (the t values) and the normal vectors of each of the triangular sectors of the control volume (the n values).

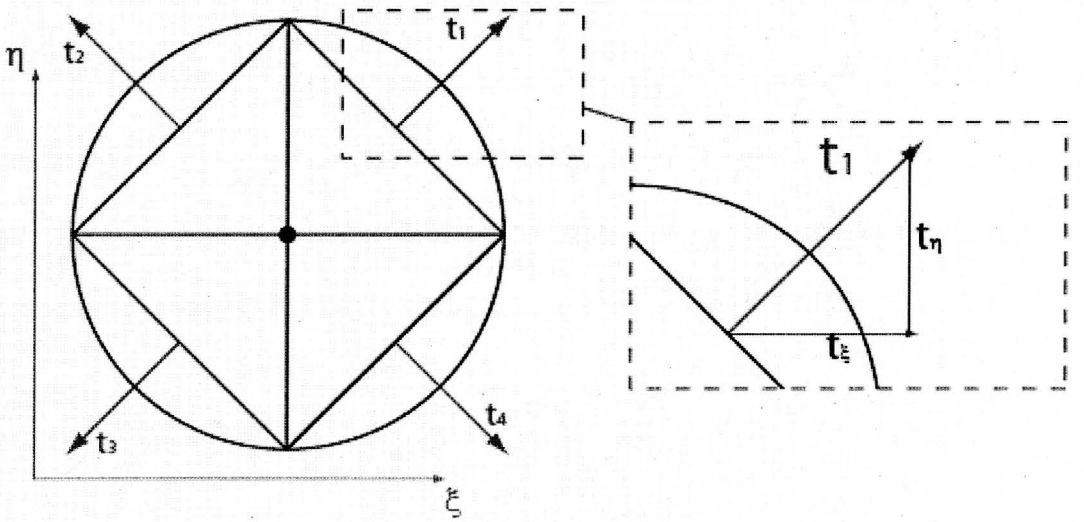


Figure 10 – Virtual control volume for a valence-count 4 node.

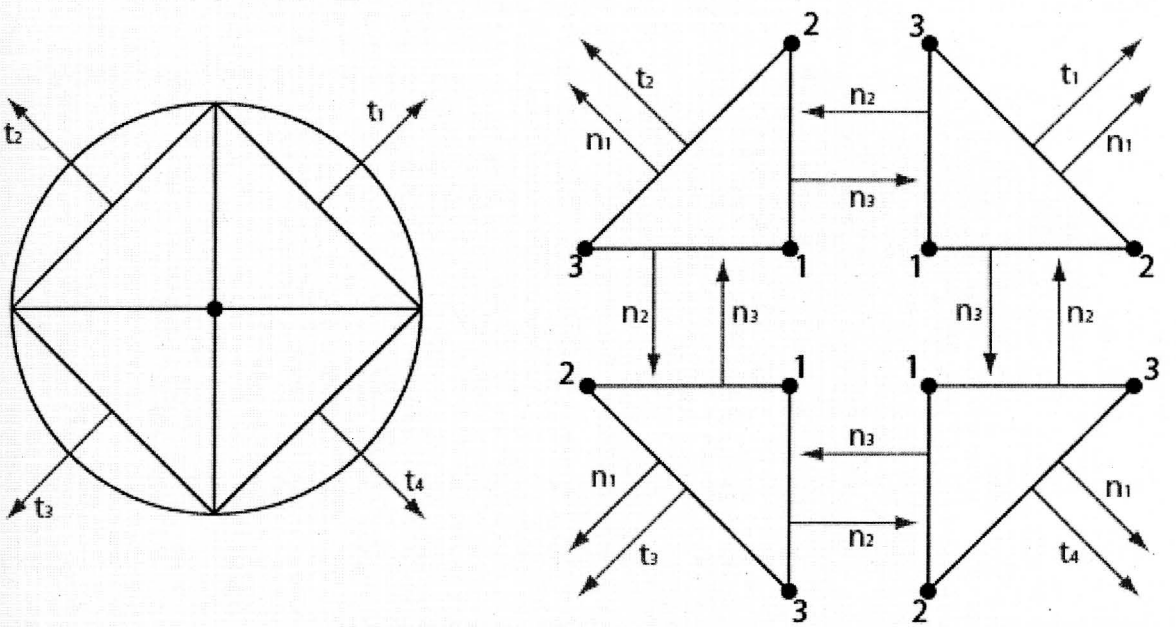


Figure 11 – The control volume for a single valence-4 node exploded into its individual triangular components.

The variable t is the non-unit surface normal vector of the overall control volume (i.e. $t_i = \hat{t}_i \, d\Gamma$ where $\hat{t}_i$ is the unit normal vector and $d\Gamma$ is the length of face i). The discretized Winslow equation (for x), defined for a given control volume with a valence count of N (i.e., N is the number of neighboring nodes and thus the number of triangles comprising the control volume) is shown below as equation (3.7).

$$\oint \left(\alpha x_\xi \hat{n}_\xi - 2\beta x_\eta \hat{n}_\xi + \gamma x_\eta \hat{n}_\eta \right) dS = \sum_{i=1}^{N} \left[\alpha x_\xi \hat{t}_\xi - 2\beta x_\eta \hat{t}_\xi + \gamma x_\eta \hat{t}_\eta \right]_i d\Gamma = 0$$

$$\sum_{i=1}^{N} \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_i = 0 \qquad (3.7)$$

Expanding this summation for a node with a valence count of 4, such as the central node shown on Figure 10, gives:

$$\sum_{i=1}^{4} \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_i = 0$$

$$\left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_1 + \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_2$$
$$+ \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_3 + \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_4 = 0$$

The discretized integral expanded above is the integral for the central node's entire control volume, whose surface has been discretized into 4 line-segments. However, the elliptic smoothing equations apply to an arbitrary control volume so the argument can be made that the only way for this integral to be zero in general is if the integrand is zero. Another way to think about this is that the discrete form of the Winslow equations have to apply everywhere in the domain, both globally and locally. Thus, for each of the

sectors (the individual triangles in discretized space) of the control volume, the following equations apply.

$$\begin{aligned} \alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta = 0 \\ \alpha y_\xi t_\xi - 2\beta y_\eta t_\xi + \gamma y_\eta t_\eta = 0 \end{aligned} \tag{3.8}$$

From the Gradient Section in chapter 2 we can apply equation (2.11) to get the values of the gradients (with respect to the computational coordinates) in each triangle.

$$\begin{aligned} x_\xi &= -\frac{1}{2A}\left(n_{\xi 1}x_1 + n_{\xi 2}x_2 + n_{\xi 3}x_3\right) \\ x_\eta &= -\frac{1}{2A}\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right) \end{aligned} \tag{3.9}$$

$$\begin{aligned} y_\xi &= -\frac{1}{2A}\left(n_{\xi 1}y_1 + n_{\xi 2}y_2 + n_{\xi 3}y_3\right) \\ y_\eta &= -\frac{1}{2A}\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right) \end{aligned} \tag{3.10}$$

In the equations for the gradients, $n_{\xi 1}$ $n_{\xi 2}$ and $n_{\xi 3}$ represent the ξ component of the non-unit normal vector of the three sides of the triangle and likewise for η.

Using the gradient relationships from equation (3.9) and (3.10) in equation (3.8) and multiplying by -2A (which can be done because no forcing functions are being used and the equations are homogeneous) results in equation (3.11).

$$\begin{aligned} &\alpha\left(n_{\xi 1}x_1 + n_{\xi 2}x_2 + n_{\xi 3}x_3\right)t_\xi - 2\beta\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right)t_\xi \\ &+\gamma\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right)t_\eta = 0 \\ &\alpha\left(n_{\xi 1}y_1 + n_{\xi 2}y_2 + n_{\xi 3}y_3\right)t_\xi - 2\beta\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right)t_\xi \\ &+\gamma\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right)t_\eta = 0 \end{aligned} \tag{3.11}$$

Rearranging equations (3.11) gives the Winslow equations in a form that allows them to be solved for the central node (x_1,y_1) using an iterative method such as the Point Implicit method. This final form is shown as equation (3.12).

$$\begin{aligned} &x_1\left(\alpha n_{\xi 1}t_\xi - 2\beta n_{\eta 1}t_\xi + \gamma n_{\eta 1}t_\eta\right) + x_2\left(\alpha n_{\xi 2}t_\xi - 2\beta n_{\eta 2}t_\xi + \gamma n_{\eta 2}t_\eta\right) \\ &+x_3\left(\alpha n_{\xi 3}t_\xi - 2\beta n_{\eta 3} + \gamma n_{\eta 3}t_\eta\right) = 0 \\ &y_1\left(\alpha n_{\xi 1}t_\xi - 2\beta n_{\eta 1}t_\xi + \gamma n_{\eta 1}t_\eta\right) + y_2\left(\alpha n_{\xi 2}t_\xi - 2\beta n_{\eta 2}t_\xi + \gamma n_{\eta 2}t_\eta\right) \\ &+y_3\left(\alpha n_{\xi 3}t_\xi - 2\beta n_{\eta 3}t_\xi + \gamma n_{\eta 3}t_\eta\right) = 0 \end{aligned} \tag{3.12}$$

The values in parentheses in equation (3.12) can be thought of as weights for the Winslow equations at each node and the two equations shown in equation (3.12) can be rewritten in matrix form as follows.

$$\begin{aligned} x_1w_{11} + x_2w_{12} + x_3w_{13} &= 0 \\ y_1w_{21} + y_2w_{22} + y_3w_{23} &= 0 \end{aligned}$$

$$\begin{bmatrix} w_{11} & 0 \\ 0 & w_{21} \end{bmatrix}\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} w_{12} & 0 \\ 0 & w_{22} \end{bmatrix}\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} + \begin{bmatrix} w_{13} & 0 \\ 0 & w_{23} \end{bmatrix}\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = 0 \tag{3.13}$$

To solve the overall system, a global matrix is constructed where each element of the global matrix is a 2×2 matrix and each row of the global matrix represents a single node in the system. The values of the elements of the row associated with a given node are constructed by using the values of w_{11} and w_{21} as contributions to the diagonal element of the row (the central node) and the values of w_{12}, w_{22}, w_{13} and w_{23} as contributions to the off-diagonal elements. Refer to Figure 11 and it becomes clear that the contributions for each triangle making up the control volume for a node are being summed to satisfy the integrals first shown as equation (3.6) and then discretized in equation (3.7).

Consider a node with a valence count of 4, such as the central node shown in Figure 11. This node will have a control volume comprised of 4 triangles, each of which will have a vertex that represents the central node and two vertices that represent two of the central node's neighbors. The local numbering convention used for each of the triangles is that the central node is node number 1 and the two neighboring nodes are numbered node 2 and node 3 based on the right-hand rule. The weights for each of the nodes in each of the triangles are calculated based on the normal vectors and the Winslow coefficients for each triangle (the gradients, and thus the coefficients, are treated as constant within each triangle). When the contributions for each of the triangles have been summed up, the off diagonal contributions (w_{12}, w_{22}, w_{13} and w_{23}) can be moved to the right hand side of the global equation and central node values (w_{11} and w_{21}), which are on the diagonal, can be solved in an iterative fashion.

3.4 – Effects of Computational Space on the Winslow Equations

When employing structured techniques for mesh generation, the mapping between the physical mesh and the computational mesh is well established with the mesh in computational space generally represented by a Cartesian grid. The spacing of the nodes in computational space may be manipulated to affect the spacing in physical space but the computational space is generally always defined as a Cartesian mesh with the directions orthogonal to each other.

In unstructured meshing, there is no such universally accepted way for defining the computational space. There has been a significant amount of success in unstructured

mesh smoothing by using a virtual control volume paradigm for the computational space where each node is effectively decoupled from the computational spaces of the other nodes. Because this mapping, while successful, is still relatively immature, alternate paradigms for the unstructured computational space were also explored. This was done to better understand the characteristics of the Winslow equations as applied through a virtual control volume and also to investigate whether modifying the virtual control volume construction might result in improved grids in certain instances. Three computational space paradigms were initially investigated and compared. They were:

1. The Virtual Control Volume Paradigm
2. A Direct Mapping Paradigm that directly mapped the physical space to the computational space
3. A Hybrid Mapping Paradigm that used the mesh from the original physical mesh but then scaled it onto a unit circle

For the simple physical space shown on Figure 12, the three computational spaces generated using the three paradigms described above are shown on Figure 13. In these figures, the edges in physical space are color-coordinated with the normal vectors in computational space for ease of comparison.

The three paradigms for the unstructured computational space (illustrated on Figure 13) were applied to a coarse NACA0012 mesh and the results that occurred are shown on Figure 14. The first two approaches (using virtual control volumes and direct mapping for the computational space) seem to generate similar grids while the hybrid computational

space blows the mesh away from the surface. However, even though mesh A and mesh B on Figure 14 look similar, the computational spaces for each of the cases are significantly different and are affecting the physical mesh in different ways.

A simpler case was also examined with similar results. A simple box mesh was investigated that had the same characteristic geometric progression as that seen on the NACA0012 mesh (i.e., the mesh goes from a comparatively small mesh element size at the solid surface (bottom) to a comparatively large mesh element size at the farfield (top)). Figure 15 shows that the new grids have the same characteristics as seen before. The virtual control volume approach and direct mapping result in similar grids while the hybrid approach, where the original physical mesh is scaled to a unit circle, stretches the mesh away from the surface. For additional clarity, look also at a superimposed view of the original mesh with the hybrid mesh, which is shown on Figure 16.

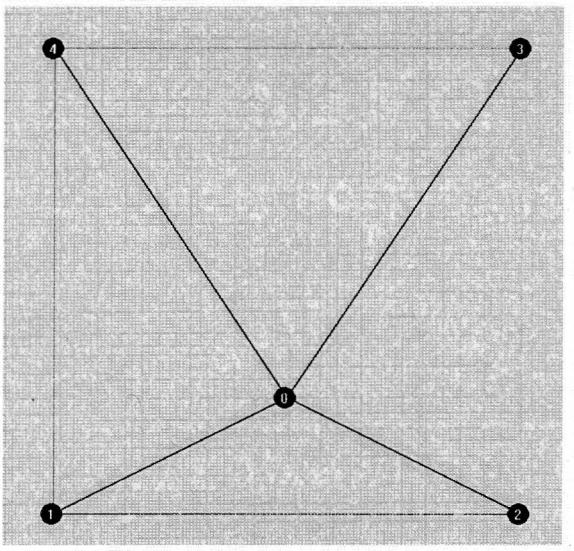


Figure 12 – Physical domain of a 5-point mesh (box5).

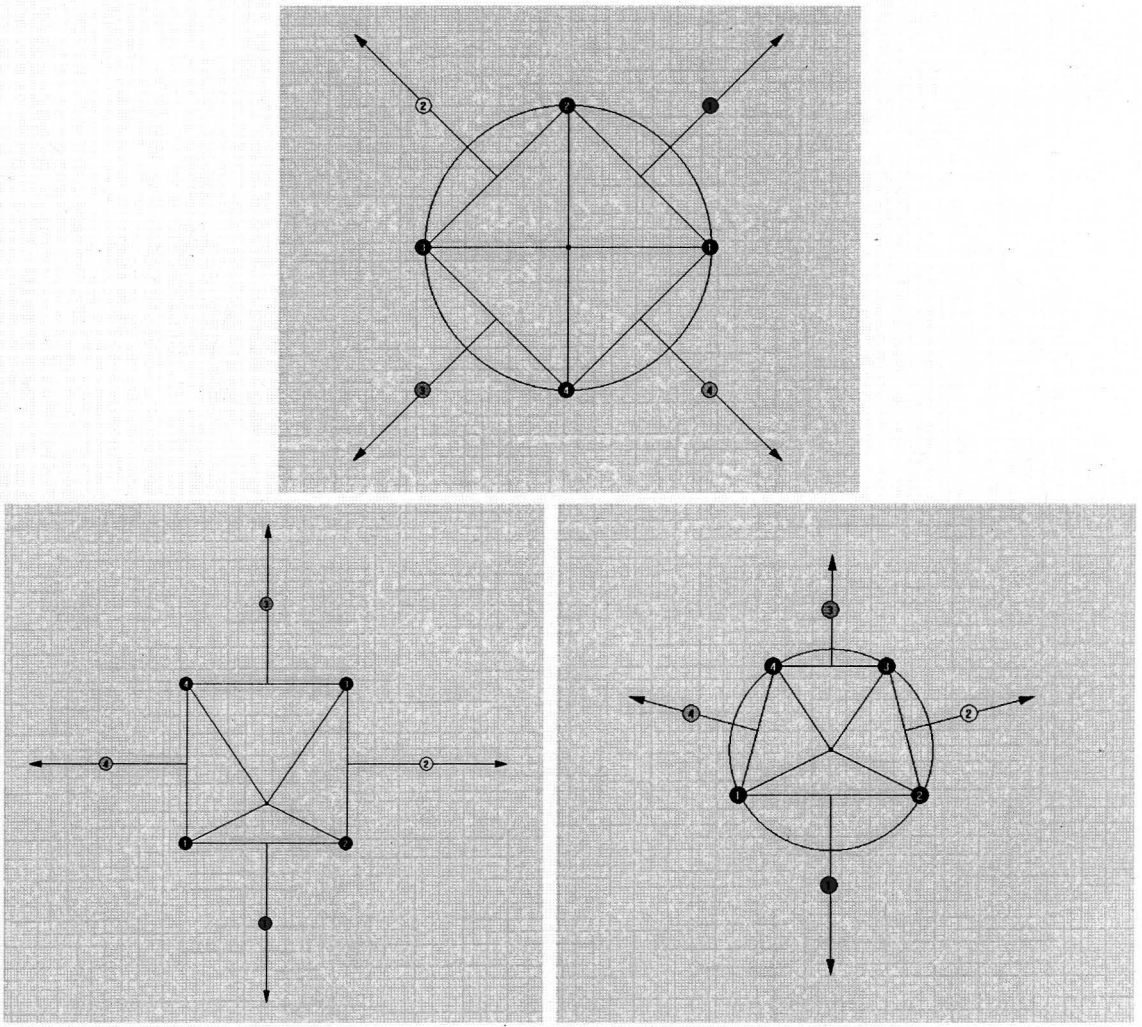


Figure 13 – Computational space based on a virtual control volume (top) direct mapping (bottom left) and hybrid (or scaled) mapping (bottom right).

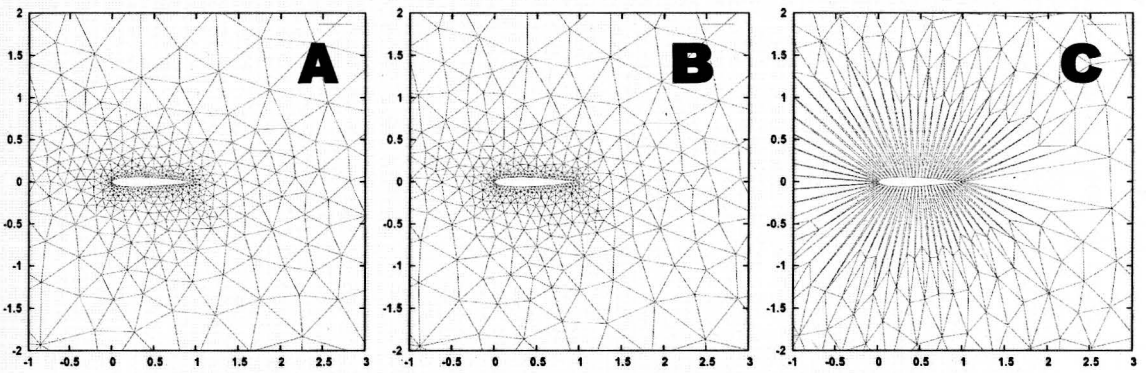


Figure 14 – Effect of computational space on a NACA0012 airfoil mesh.
A – Virtual Control Volume, B – Direct Mapping, C – Hybrid Mapping.

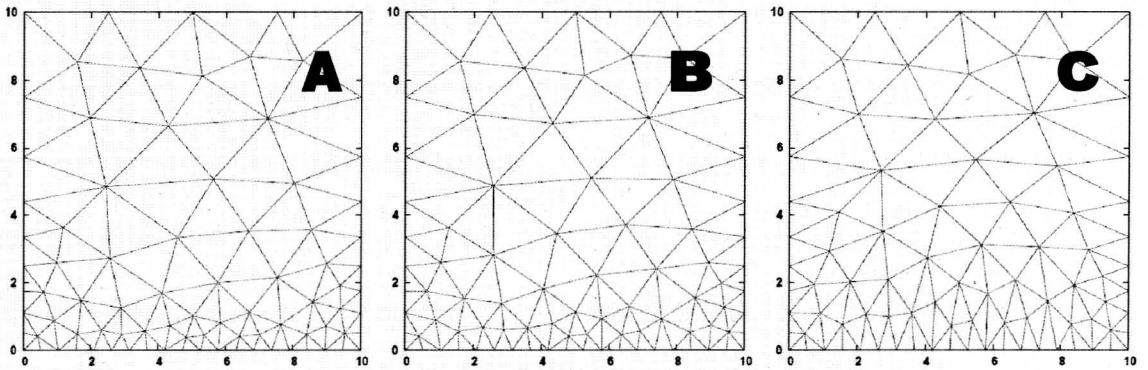


Figure 15 – Effect of computational space on a simple box mesh.
A – Virtual Control Volume, B – Direct Mapping, C – Hybrid Mapping.

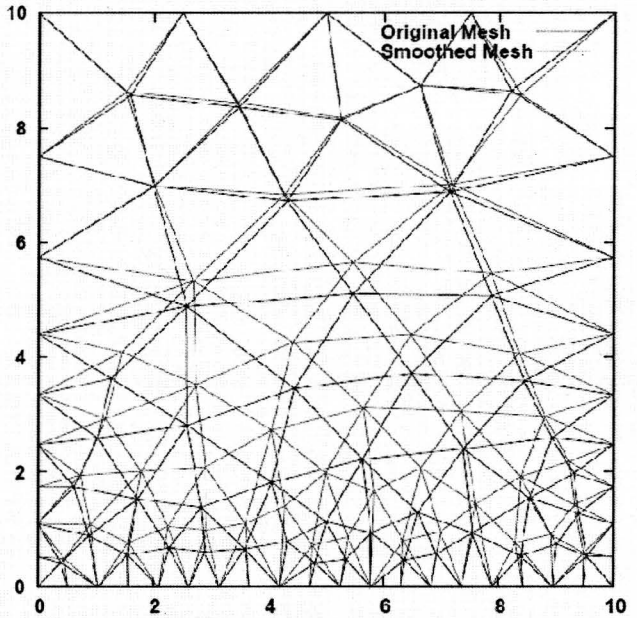


Figure 16 – Original mesh compared with a mesh generated using a hybrid computational space paradigm.

To explore exactly what is happening, the smallest possible non-trivial mesh was examined. This was a mesh with a total of 5 points, 4 of which are boundary points that will remain static. The physical domain for this mesh was shown on Figure 12. It is instructive to look at this simple mesh because the Winslow equations are a system of equations that work on the entire system of mesh points in an iterative and collective fashion. This 5-point mesh (referred to as box5) allows an observer to isolate a single point for analysis.

While examining the box5 test mesh, it is important to keep focused on the fact that at each node (in this case only one node) the Winslow equations, which are derived by taking the Laplacian of the computational coordinates with respect to physical space ($\nabla^2\xi = 0$ and $\nabla^2\eta = 0$) are being solved. By focusing on the Winslow equations, which are shown on equation (3.2), and keeping in mind the characteristics of Laplace's equation, much insight can be gained from examining the elliptic smoothing of the single interior point.

The first of the three computational space paradigms used to smooth the box5 mesh is the virtual control volume paradigm. This computational space is shown on as the top illustration on Figure 13 with the edge normals color coordinated to the edge colors of the physical mesh shown in Figure 12. The second paradigm uses the original mesh as the computational space (shown on the bottom left of Figure 13) and the third paradigm uses a hybrid approach where the original mesh is used for the computational space but the space is scaled such that each of the surrounding points are on a unit circle surrounding the node of interest (shown on the bottom right of Figure 13).

The virtual control volume paradigm results in the interior mesh point being driven to a location equidistant from the surrounding nodes. The direct mapping paradigm results in the interior mesh point remaining exactly where it started out. The hybrid paradigm causes the interior node to be driven away from the surface that it was located closest to initially. The final smoothed positions of the interior node and the accompanying meshes for all three smoothing approaches are shown on Figure 17. By examining each of the resulting meshes, which are generated using each of the three computational space paradigms, it can be seen that the choice of computational space does, indeed, have a profound effect on the final position of the elliptically smoothed nodes.

To capture the effect that each of the three computational spaces is having when driving the physical mesh to its new orientation, note that each of the computational space meshes is driving the physical mesh to mimic the characteristics of the given computational mesh. This can be clearly seen (as on Figure 18) when looking at the ratios of where the mesh falls in its domain in computational space compared with the ratios of where it falls in physical space; they are exactly the same.

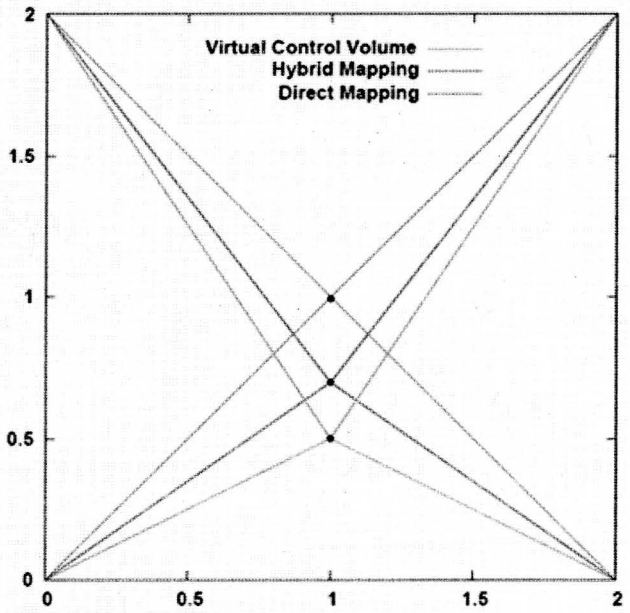


Figure 17 – Physical Box5 mesh after smoothing.

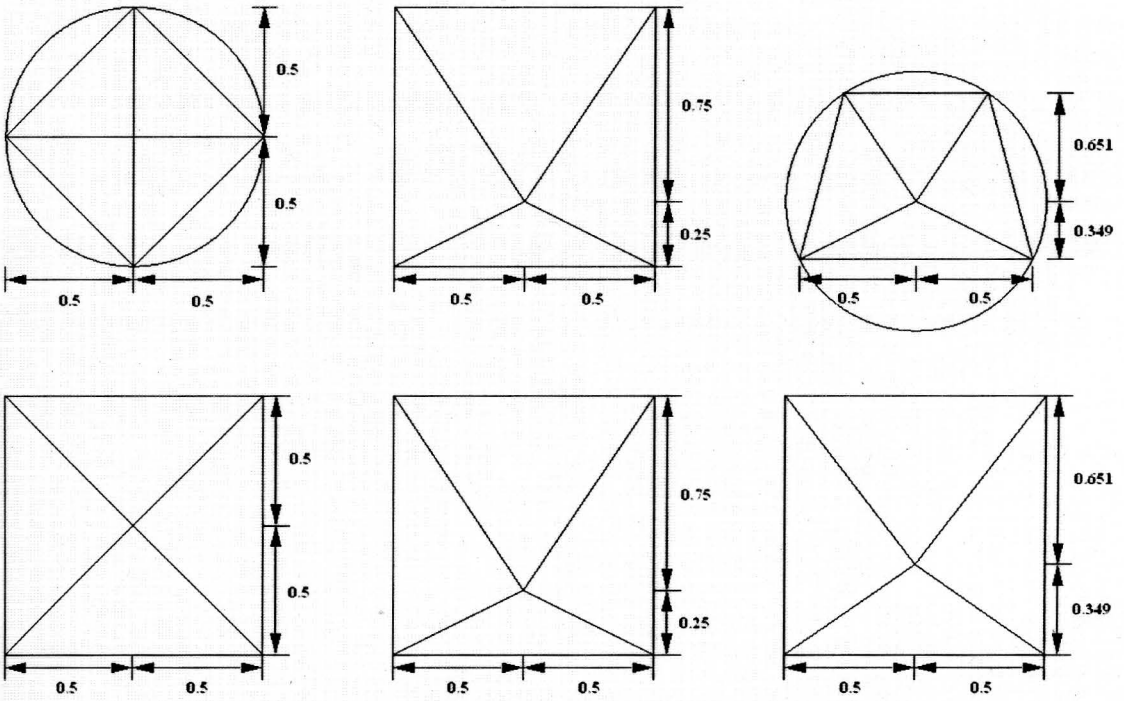


Figure 18 – Comparison of an interior point in computational (top) and physical (bottom) space. Results on the left are for the virtual control volume case; results in the middle are for the direct mapping case; results on the right are for the hybrid (scaled) mapping case.

In all three of the box5 cases that were examined, the x coordinates of the nodes are distributed evenly and will not have any effect on the interior node. Thus, the y coordinate can be focused on in order to understand what is happening mathematically. The Winslow equation for y, first shown on equation (3.2) and restated here is:

$$\alpha \frac{\partial^2 y}{\partial \xi^2} - 2\beta \frac{\partial^2 y}{\partial \xi \partial \eta} + \gamma \frac{\partial^2 y}{\partial \eta^2} = 0$$

Recall that the coefficients for the Winslow Equations, which are made up of derivatives of the physical coordinates with respect to the computational coordinates, are:

$$\alpha = \left(\frac{\partial x}{\partial \eta}\right)^2 + \left(\frac{\partial y}{\partial \eta}\right)^2 , \quad \beta = \frac{\partial x}{\partial \xi}\frac{\partial x}{\partial \eta} + \frac{\partial y}{\partial \xi}\frac{\partial y}{\partial \eta} , \quad \gamma = \left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \xi}\right)^2$$

The Winslow equations originated with structured meshes, where it is easy to think of the derivatives from a finite difference perspective. To illustrate the concept of the derivatives of physical space with respect to the computational space coordinates from a finite difference perspective, consider the structured mesh surrounding a NACA0012 airfoil shown on Figure 19 where the ξ direction wraps around the airfoil and the η direction emanates out from the airfoil.

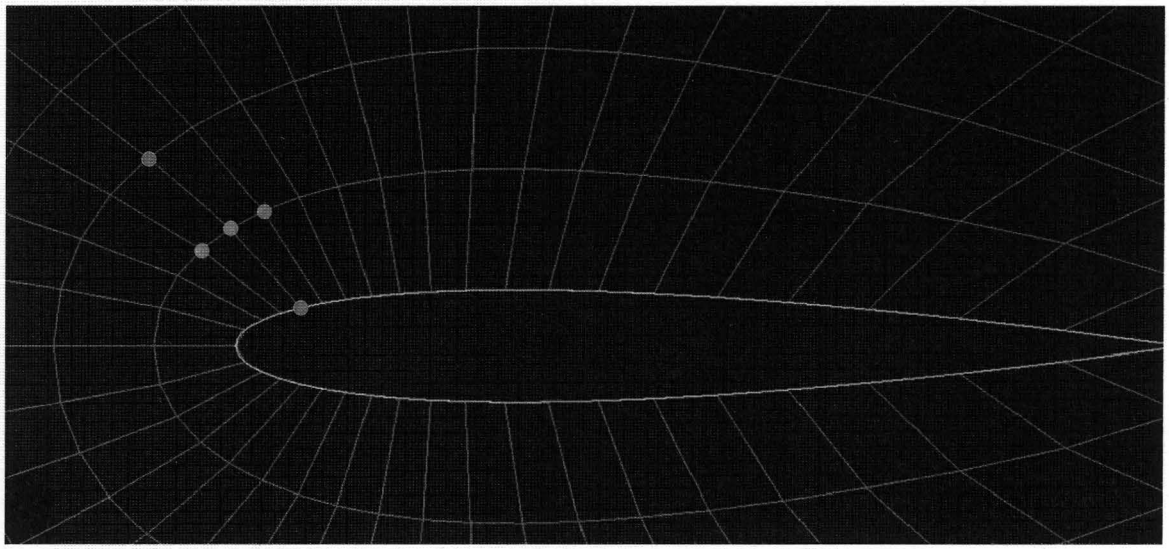


Figure 19 – Structured mesh around an airfoil.

Consider the green dot on the structured grid surrounding the airfoil that is shown on Figure 19. If the ξ direction wraps around the airfoil and the η direction emanates out from the airfoil, the derivatives can be found as:

$\frac{\partial \xi}{\partial x} = \frac{\Delta \xi}{\Delta x}$ where Δx is the x distance between the pink dots and $\Delta \xi$ is 2.

$\frac{\partial \xi}{\partial y} = \frac{\Delta \xi}{\Delta y}$ where Δy is the y distance between the pink dots and $\Delta \xi$ is 2.

$\frac{\partial \eta}{\partial x} = \frac{\Delta \xi}{\Delta x}$ where Δx is the x distance between the orange dots and $\Delta \eta$ is 2.

$\frac{\partial \eta}{\partial y} = \frac{\Delta \eta}{\Delta y}$ where Δy is the y distance between the orange dots and $\Delta \eta$ is 2.

There is, however, no equivalent finite difference way to do this using unstructured methods and, instead, the derivatives are examined from a finite volume perspective, which is derived from the divergence theorem and relies on surface normals. The partial derivatives can be found using the non-unit normal vectors of each of the edges and the

average value of some parameter over an edge of a control volume. Specifically, the gradients of y with respect to the computational coordinates (ξ,η), can be determined as:

$$\frac{\partial y}{\partial \xi} = \frac{1}{A} \sum_{i=1}^{\#\text{of edges}} \bar{y}_i n_{\xi i}$$

$$\frac{\partial y}{\partial \eta} = \frac{1}{A} \sum_{i=1}^{\#\text{of edges}} \bar{y}_i n_{\eta i}$$

The non-unit surface normal vectors for the three computational space paradigms shown on Figure 13 and described above have the values tabulated in Table 1.

	Virtual Control Volume		Direct Mapping		Scaled Mapping	
Edge 1 ($\bar{y}=0$)	$n_\xi = 1.0$	$n_\eta = 1.0$	$n_\xi = 0.0$	$n_\eta = -2.0$	$n_\xi = 0.0$	$n_\eta = -1.79$
Edge 2 ($\bar{y}=1$)	$n_\xi = -1.0$	$n_\eta = 1.0$	$n_\xi = 2.0$	$n_\eta = 0.0$	$n_\xi = 1.28$	$n_\eta = 0.340$
Edge 3 ($\bar{y}=2$)	$n_\xi = -1.0$	$n_\eta = -1.0$	$n_\xi = 0.0$	$n_\eta = 2.0$	$n_\xi = 0.0$	$n_\eta = -1.11$
Edge 4 ($\bar{y}=1$)	$n_\xi = 1.0$	$n_\eta = -1.0$	$n_\xi = -2.0$	$n_\eta = 0.0$	$n_\xi = -1.28$	$n_\eta = 0.340$

Table 1 – Non-unit normal values.

Each of the edges in computational space is color coordinated with the edge in physical space. When the average values of y (i.e., $\bar{y}$) over each of the edges in computational space are used in the equations for calculating gradients, the coefficients for the Winslow equations can be calculated. Coefficients were generated for all three paradigms and the results are compiled in Table 2.

Virtual Control Volume	Direct Mapping	Scaled Mapping
$\alpha = 1.0$	$\alpha = 2.0$	$\alpha = 2.44$
$\beta = 0.0$	$\beta = 0.0$	$\beta = 0.0$
$\gamma = 1.0$	$\gamma = 2.0$	$\gamma = 1.90$

Table 2 – Winslow equation coefficients.

For the virtual control volume paradigm and the direct mapping paradigm, the Winslow equation for y, when the coefficients from Table 2 are used, reverts to the simple homogeneous Laplace's Equation for y in computational space (shown below).

$$\alpha \frac{\partial^2 y}{\partial \xi^2} - \beta \frac{\partial^2 y}{\partial \xi \partial \eta} + \gamma \frac{\partial^2 y}{\partial \eta^2} = 0$$

$$\frac{\partial^2 y}{\partial \xi^2} + \frac{\partial^2 y}{\partial \eta^2} = 0$$

From inspection of the computational space domains for the virtual control volume and direct mapping paradigms (Figure 13), and knowledge of the behavior of Laplace's Equation, it becomes obvious that the y value for the interior points would become what they did (the values became what would be expected given regions bounded by Dirichlet boundary conditions such as those shown on Figure 20).

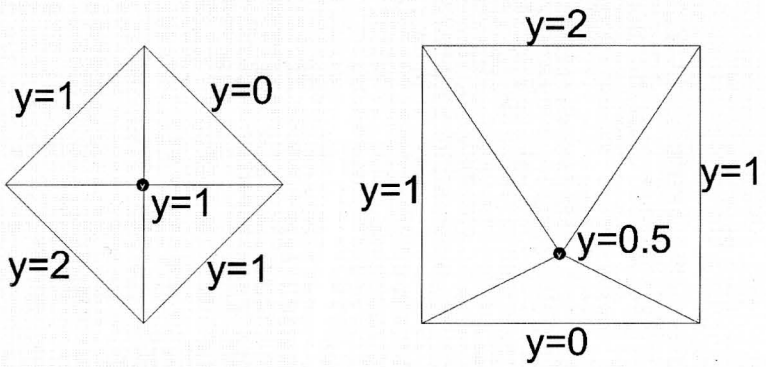


Figure 20 – Interior value for y given Dirichlet boundary conditions and regions driven by Laplace's equation.

The case where a hybrid computational space is used is more interesting because it does not revert to a simple homogeneous Laplace equation but instead ends up in a more general second order partial differential equation (shown below) where the coefficients are not unity. These non-unity coefficients are affecting the solution in such a way that that they are driving the node away from the nearest surface in physical space.

$$\alpha \frac{\partial^2 y}{\partial \xi^2} - \beta \frac{\partial^2 y}{\partial \xi \partial \eta} + \gamma \frac{\partial^2 y}{\partial \eta^2} = 0$$

$$2.44 \frac{\partial^2 y}{\partial \xi^2} + 1.90 \frac{\partial^2 y}{\partial \eta^2} = 0$$

Consider once more a generic mesh for analyzing external flow on a body which progresses from a smaller element size at the body surface to a larger element size at the farfield. Consider a mesh node with a valence count of 6 (a node connected to 6 other neighboring nodes). If this node were in a perfectly uniform mesh, it could be modeled by a point in the center of a circle. However, if there is some effective geometric progression in the mesh, the node could be modeled as a point offset in a circle. So, a point above a surface could be modeled as shown on Figure 21. If this point was then mapped to the center of a circle (as is done with the Hybrid Computational Space Paradigm) the results would be as seen on Figure 22. Note the effect that this has on the normals. The normals have all shifted away from the surface.

As an example from an actual (though still simple) mesh, the mesh shown originally in Figure 15 is reexamined here with the focus on the region surrounded by the green box and the node highlighted by the green dot in Figure 23.

If the node shown in green in Figure 23 is mapped to a unit circle, the results can be seen on the image on the right of Figure 23. The same effect as seen on Figure 22 is also seen on these normal vectors, which is that the normal vectors are again shifted away from the surface. The normals from the original mesh are shown in black on Figure 23 while the normals of the shifted space are shown in red. Both normals are shifted to the surface of the unit circle for easier comparison. Because the normals are driving the Winslow equations, is not surprising that, as the normals are shifted away from the surface, the mesh is driven away from the surface as it is smoothed.

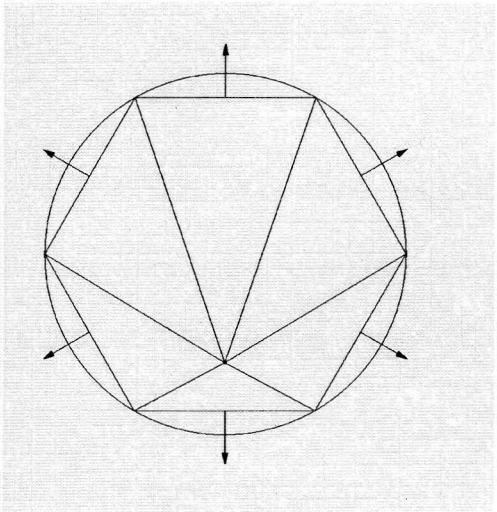


Figure 21 – Offset node.

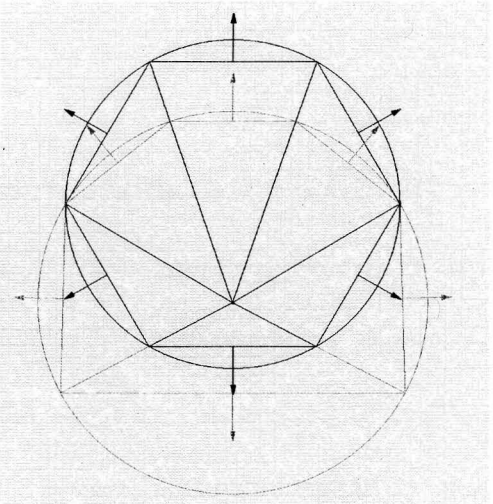


Figure 22 – Offset node and computational space using hybrid scaling.

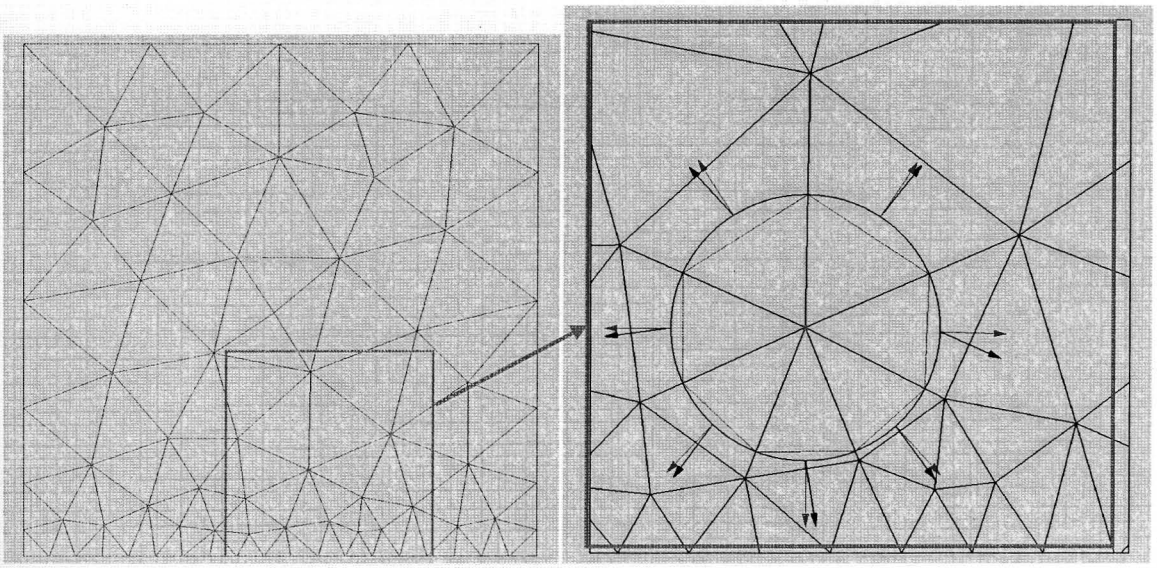


Figure 23 – Effect of hybrid scaling on a node near a surface.
Note that manipulating the computational space has a direct effect on the physical space. It is this fact that allows the homogeneous form of the Winslow equations to be used. If no direct effect was realized, forcing functions would be required to manipulate the physical mesh.

3.5 – Isotropic Winslow Equations Applied to a Flat Plate

Prior to the research described herein, the Winslow equations have been explored and utilized on many meshes suitable for analyzing inviscid flow fields. However, very little research has been performed in which the equations are applied to unstructured meshes suitable for analyzing viscous flow fields. The logical place to start the exploration into this area was with a flat plate since viscous flow on a flat plate is well defined and understood. In addition to the well-established characteristics of a flat plate, a flat plate also has the benefit of being able to be manipulated within a mesh in ways (such as bending and warping) that could cause problems with more complex geometries.

A two dimensional flat plate was created using the Gridgen software package [12] and converted to the format required by the mesh manipulation code. The initial flat plate

mesh that was used is shown on Figure 24. This initial flat plate mesh was very coarse and there was no symmetry in the mesh surrounding the flat plate surface.

The flat plate surface was rotated 45 degrees within the surrounding mesh domain and the mesh smoothing code utilizing the Winslow equations was run on the surrounding mesh. It was found that, due to the coarseness of the mesh and the proximity of the modified inner surface to the stationary outer surface, the resulting mesh, seen on Figure 25, resulted in considerable grid crossing. This was an interesting test because it illustrated the limits of the Winslow elliptic smoothing equations, especially if boundary nodes are held static.

The unsatisfactory result shown on Figure 25 leads to two conclusions: It is necessary to use a finer mesh around the flat plate and, also, there is a need to have the ability to move mesh points on a boundary surface. It can be seen from Figure 25 that there is an insufficient number of points between the endpoints of the flat plate and the outer surface. Because of this, combined with the fact that the points on the outer surface are not permitted to move, the Winslow code was not able generate (at least not with the relaxation factor that was utilized) a viable mesh for the modified (rotated) flat plate given the original connectivity.

Moving boundary points will be discussed in subsequent sections but consider here the effects of the Winslow equations on the same flat plate with a finer interior mesh. The same basic mesh setup was used again but with the exception that the boundary-decay was changed from 0.5 to 0.9 within the Gridgen grid generation package. This resulted in the refined mesh shown on Figure 26. The flat plate surface was again rotated 45 degrees

and the surrounding unstructured mesh was again smoothed using the Winslow equations. Unlike the results for the coarser mesh, the smoothing algorithm now generated a viable smoothed mesh around the rotated flat plate, which is shown on Figure 26.

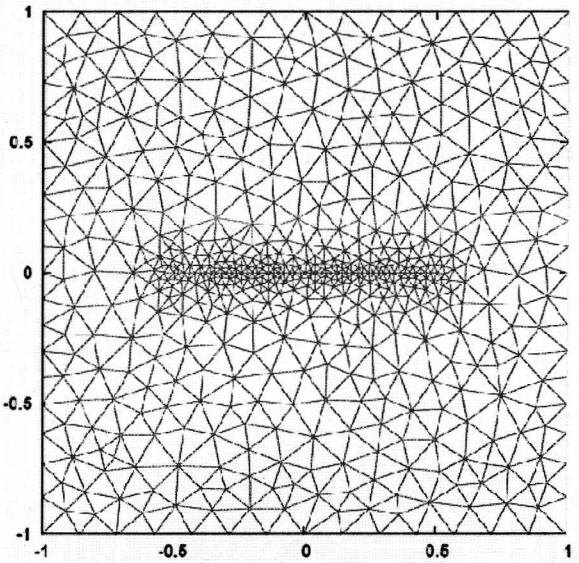


Figure 24 – Initial coarse mesh surrounding a zero-thickness flat plate.

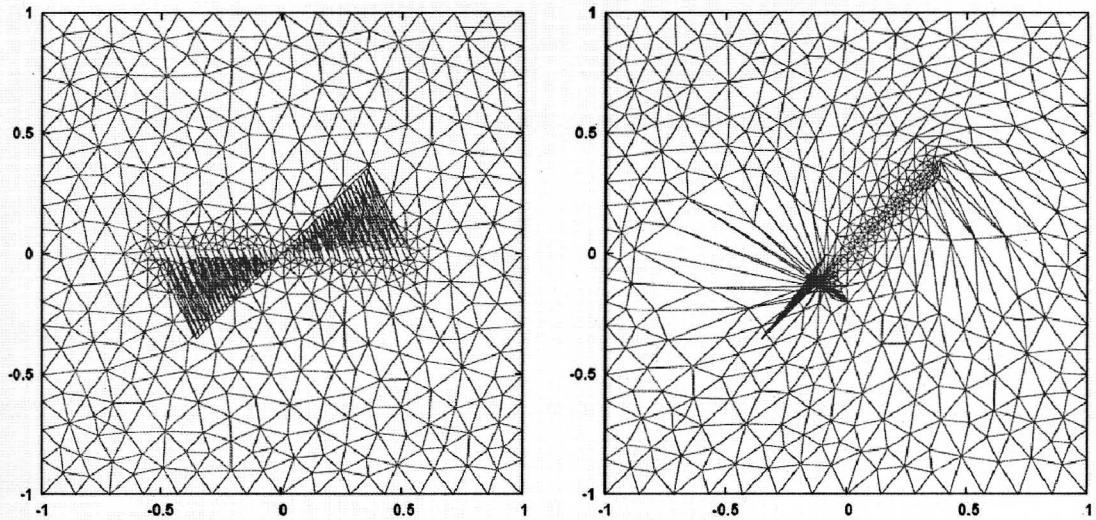


Figure 25 – Results of Winslow smoothing on a rotated coarse flat plate mesh.

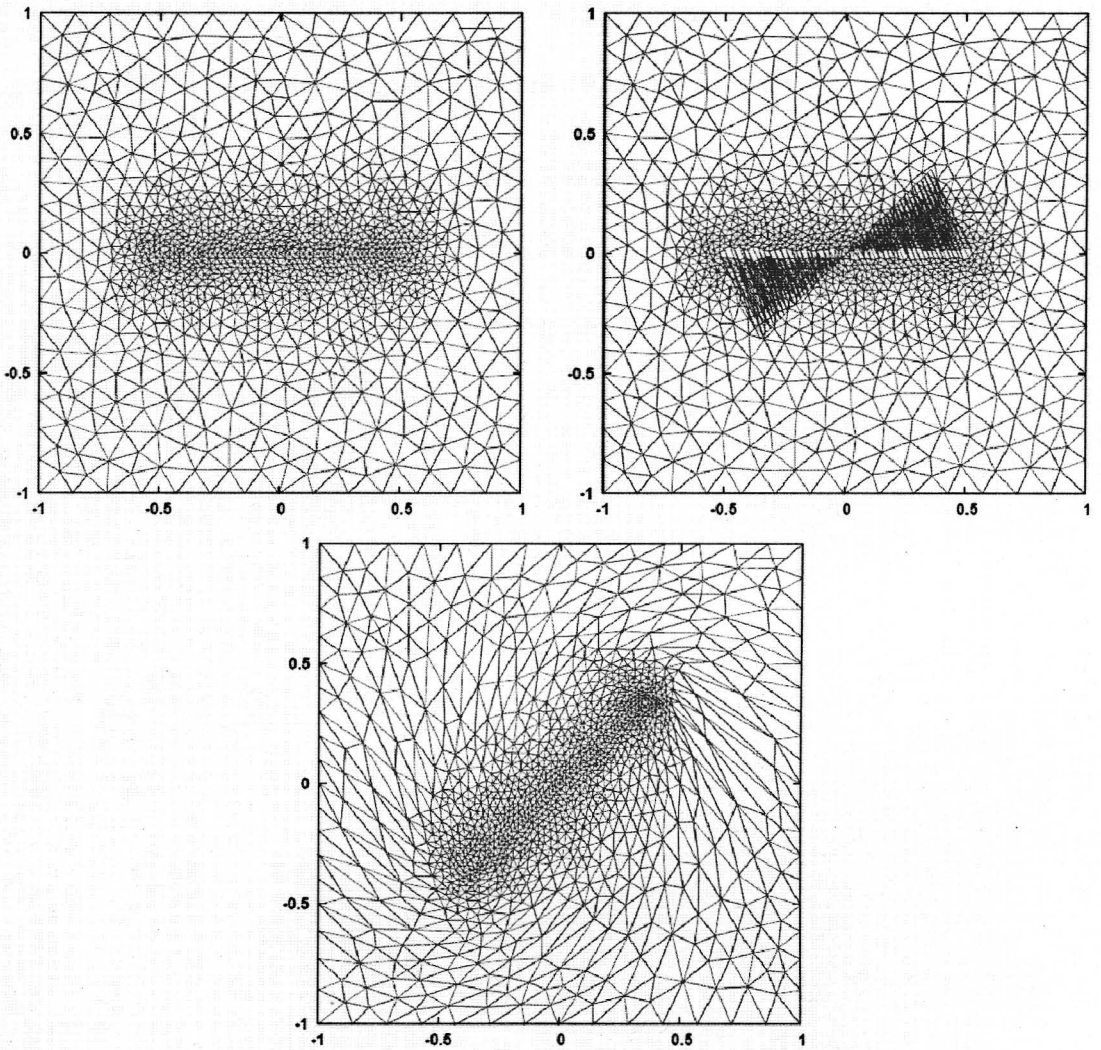


Figure 26 – Original and rotated flat plate using a boundary decay of 0.9.

Although the last mesh shown on Figure 26 is a viable mesh (i.e., there is no edge crossing or negative volumes) it is obvious that without allowing the boundary points to move, the Winslow equations will generate some unreasonably high skewness in some of the cells of the final mesh.

To further explore the Winslow Equations on a flat plate. A mesh was generated that closely resembled the mesh described in Reference [13]. This paper explored the area of

mesh movement but used entirely different techniques, which were based on the Linear Elastic equations. The new flat plate mesh is shown on Figure 27.

Three cases were examined using the mesh from Figure 27: rotation, translation and a warping of the flat plate surface. The results from these three cases are shown in Figure 28. For each of the cases, the mesh is shown after the inner surface is moved before and after smoothing has been performed.

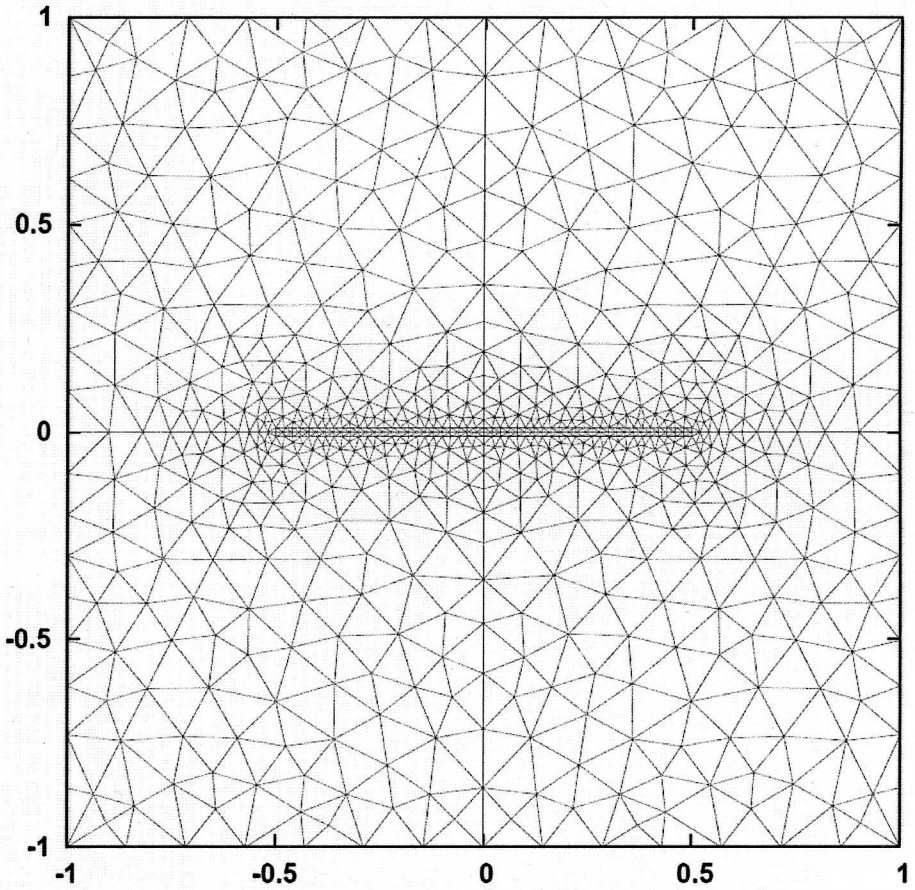


Figure 27 – Refined symmetric flat plate mesh.

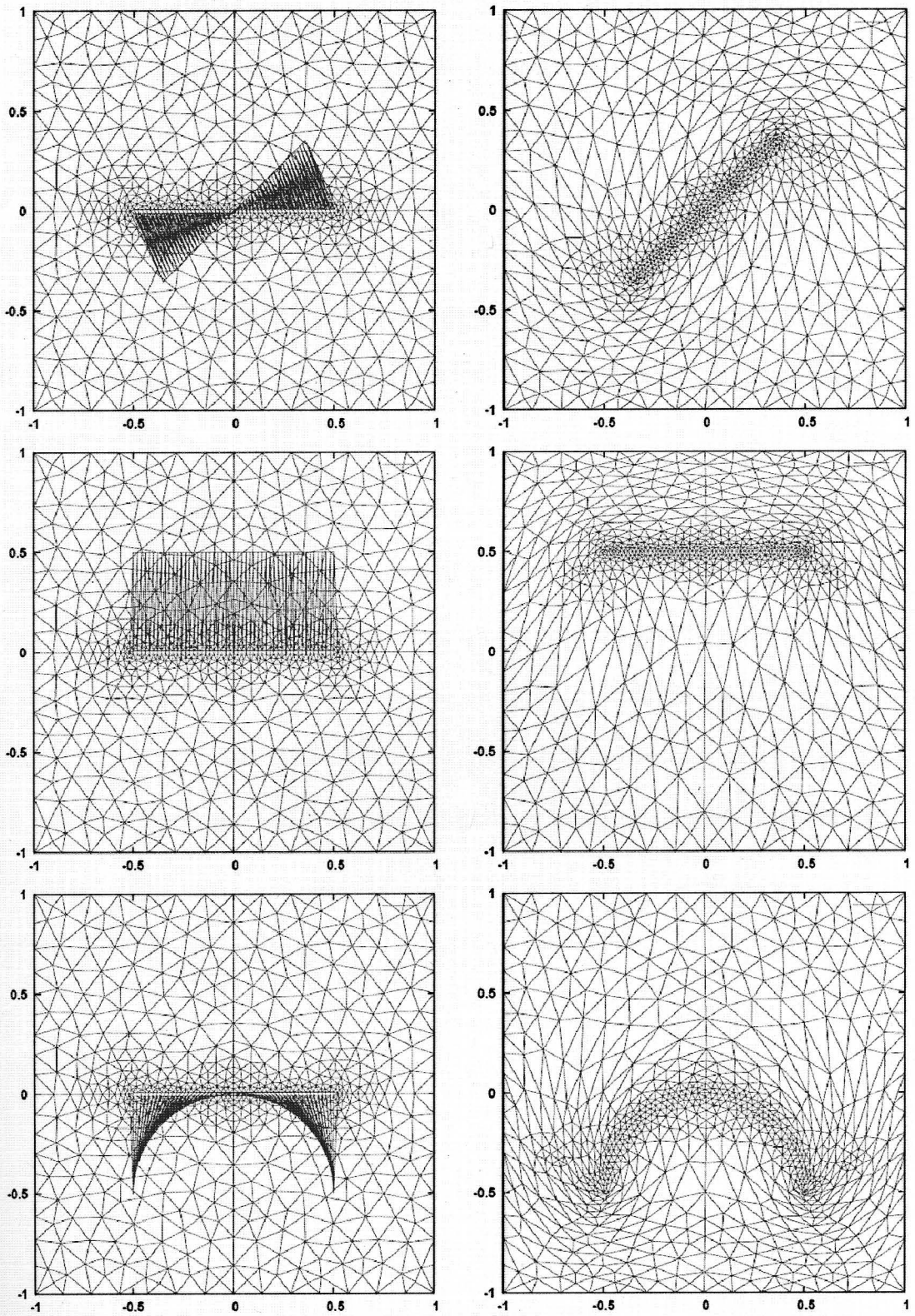


Figure 28 – Results from applying Winslow equations to a flat plate mesh that has been rotated, translated and warped into a semi-circle.

Although all of the grids on Figure 28 look relatively good, the mesh where the flat plate has been warped into a semi-circle again illustrates the need to have the ability to move boundary points if a surface near another boundary is going to be modified. A close-up of the mesh near the right endpoint of the flat plate is shown on Figure 29 and Figure 30. This close-up follows two sets of points (marked as black and blue dots) before and after mesh smoothing is performed and shows the difficulty in trying to wrap the mesh around the new geometry while retaining the original connectivity and outer surface grid spacing.

Examining what is happening on Figure 30, it can be seen that the smoothing equations are trying to wrap the two lines of connectivity around the warped flat plate but in some areas the spacing and limited connectivity between the two lines in not adequate to generate a high-quality mesh. Note the connectivity between the two sets of points near the flat plate. Before the flat plate is warped, there is adequate space in between the two sets of nodes to accommodate the connectivity between the sets. However, after the flat plate is warped and the mesh has been smoothed, the two sets of nodes are driven together in a way that makes it difficult for valid connectivity to be maintained.

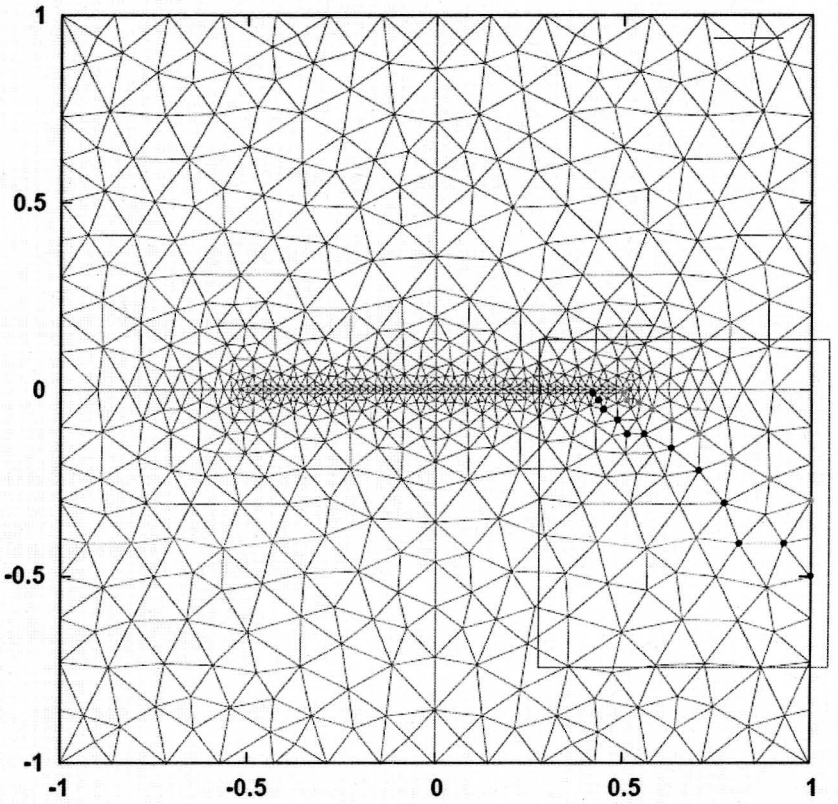


Figure 29 – Lines of connectivity between an inner boundary and an outer boundary.

The area surrounded by the blue box in this figure is examined in greater detail in Figure 30.

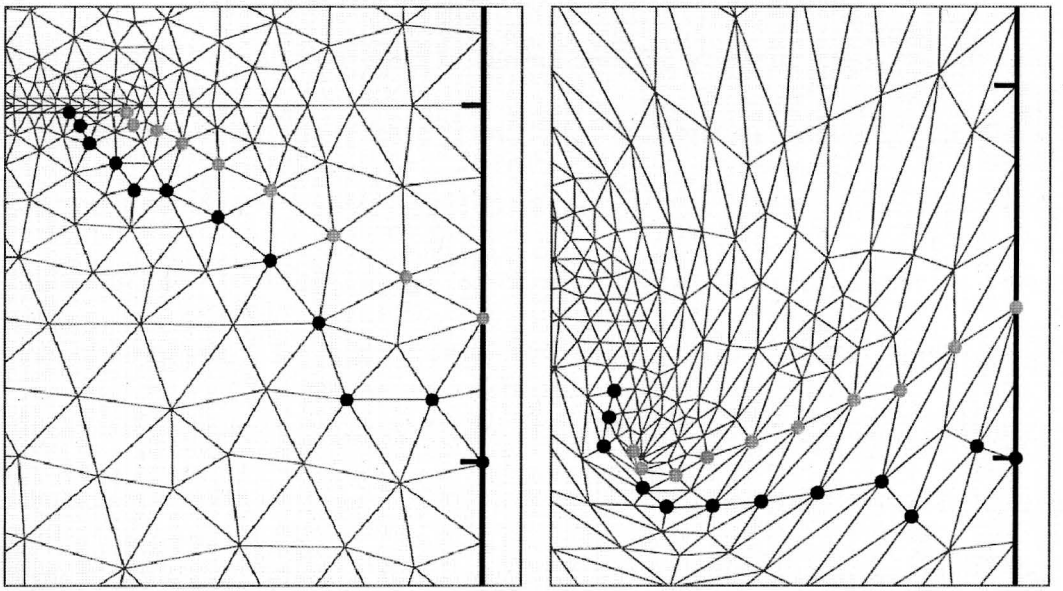


Figure 30 – Connectivity path between surface and outer boundary before and after surface modification.

3.6 – Isotropic Winslow Equations Applied to a NACA0012 Airfoil

It should be noted that although isotropic Winslow Equations have significant limitations, there are also many situations in which they are extremely valuable tools for mesh manipulation. One of these situations is to smooth the inviscid mesh of a body that has been moved or deformed in some manner. Another situation where the isotropic Winslow equations can add significant value is in Adaptive Mesh Refinement where h-refinement is employed.

H-refinement schemes determine regions of the grid where the solution is under-resolved and add elements locally to improve the resolution. However, if no smoothing is employed, the mesh can become extremely skewed as more points are added to a given area. Winslow elliptic smoothing can alleviate this tendency toward skewed cells as additional nodes are added.

A NACA0012 airfoil was used to further test the implementation of the isotropic Winslow grid smoothing algorithms. The airfoil, which had an initial angle of attack of 0 degrees, was rotated to a nose-down position of 60 degrees and the mesh was smoothed. The grid before and after the rotation can be seen on Figure 31 and Figure 32. Note that after the initial rotation, the mesh is completely invalid. The boundary points associated with the airfoil surface were moved but all of the interior points remained stationary and this resulted in a mesh containing extreme grid crossing and skewness. This is not an issue when structured grids are utilized because structured grids have an implicit computational space that is not invalidated by an invalid physical mesh as long as the

grid connectivity does not change. Likewise, by using equal angle equal edge-length virtual control volume as the computational space for the unstructured mesh, the invalid physical space was not an issue and a high-quality inviscid mesh was generated for the transformed airfoil mesh.

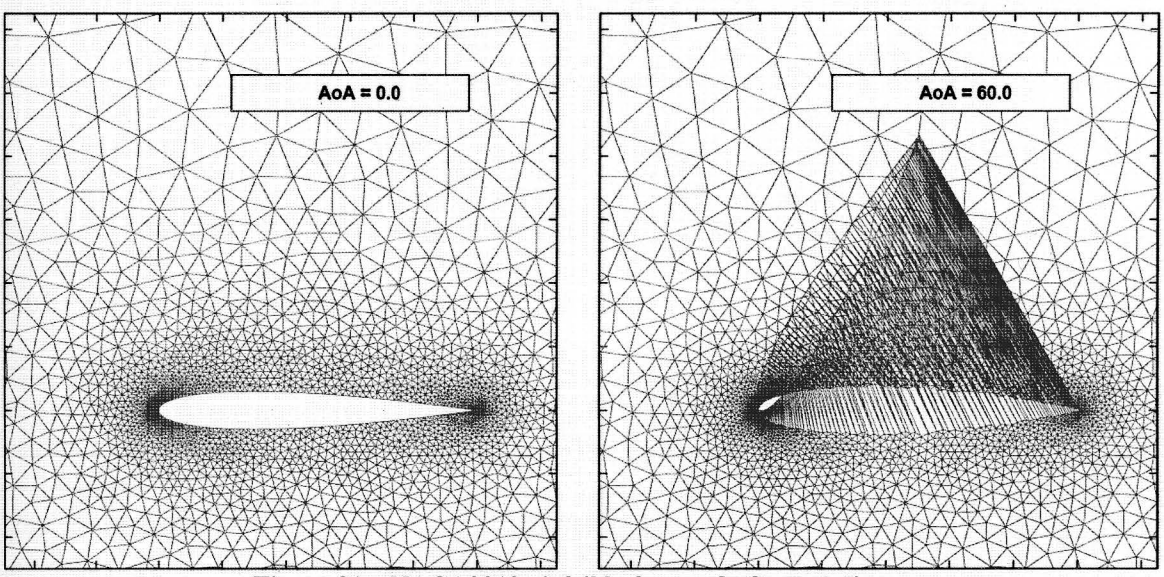


Figure 31 – NACA0012 airfoil before and after rotation.

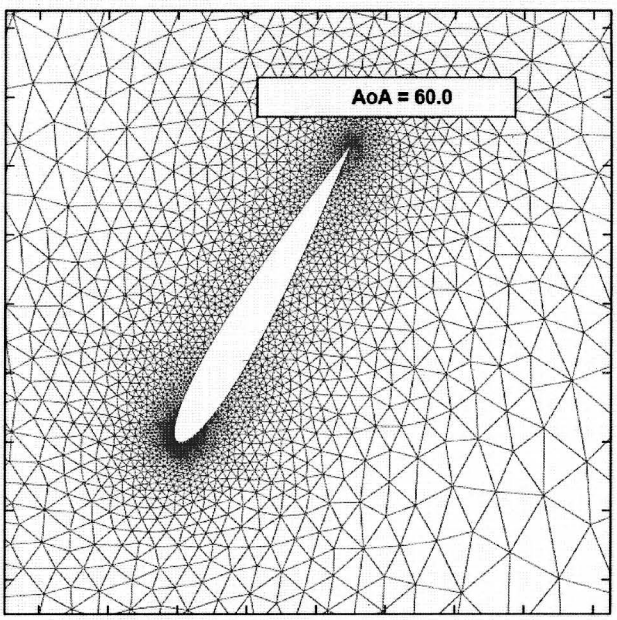


Figure 32 – NACA0012 mesh after Winslow smoothing has been performed.

The next case that was examined was a transonic case involving a NACA0012 airfoil in a Mach 0.95 flow. At Mach 0.95 and zero angle of attack, the flow speed increases to about Mach 1.3 as the flow traverses the airfoil and then shocks down at the tail. However, the flow does not quite shock down to subsonic at that point and remains slightly supersonic for several chord lengths behind the airfoil. The flow then shocks down below the speed of sound, causing the airfoil to exhibits a distinct sonic line or "fishtail shock" in its wake. This is illustrated on Figure 33. The objective of examining this case was to use solution adaptation to adapt the original grid (shown on Figure 33) with the hope of capturing the shockwaves coming off the airfoil and also the secondary shock behind it. This was an excellent case for employing the inviscid Winslow equations because the areas of interest (the sonic lines) were in the inviscid region of the mesh and thus an isotropic mesh in these regions was adequate.

Three different flow parameters (which were generated using an in-house 2D Euler-based CFD code) were examined: Mach number, pressure, and velocity magnitude. Successful solution adaptation was performed with all three parameters but Mach number was focused on because it captured some elements of both pressure and velocity. Shown below in the images comprising Figure 34 is a progression of the grid over 6 refinement passes. Figure 34 shows that the adaptation algorithm, coupled with the Winslow elliptic smoothing algorithm, successfully captured the shock coming off of the airfoil and also the secondary shock behind it. As the solution was repeatedly refined, the mesh elements in the area of the shock would have become increasingly skewed if no smoothing was employed. It is likely that eventually the skewness of the cells would have negatively affected the CFD solution. An enlarged image of the final mesh is also shown on Figure

35 and it can be seen that even after many refinement iterations, the mesh elements in the region of the shock still have good isotropic aspect ratios.

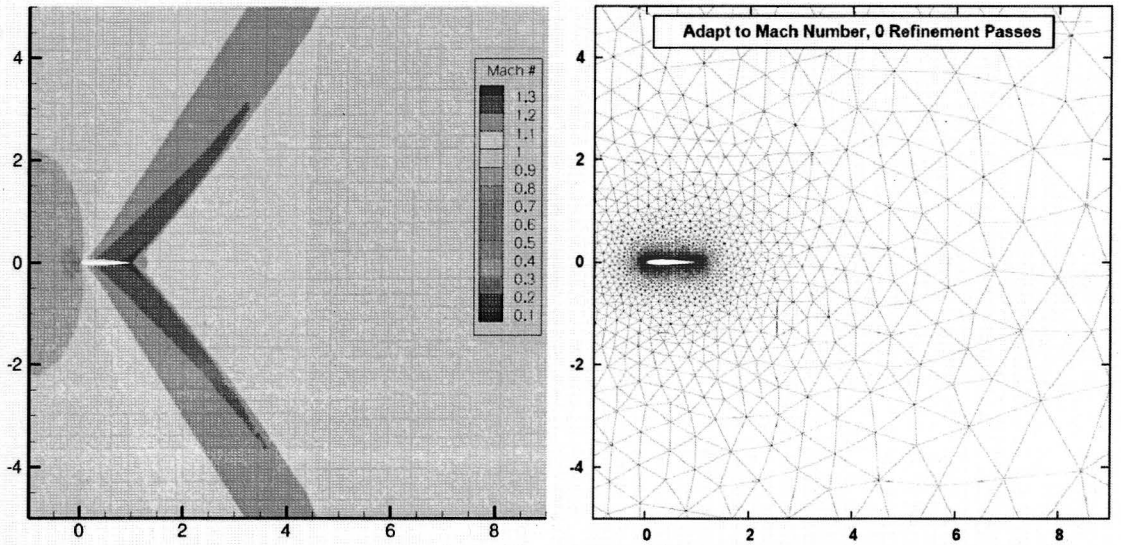


Figure 33 – Flowfield and original mesh around a NACA0012 airfoil at Mach 0.95.

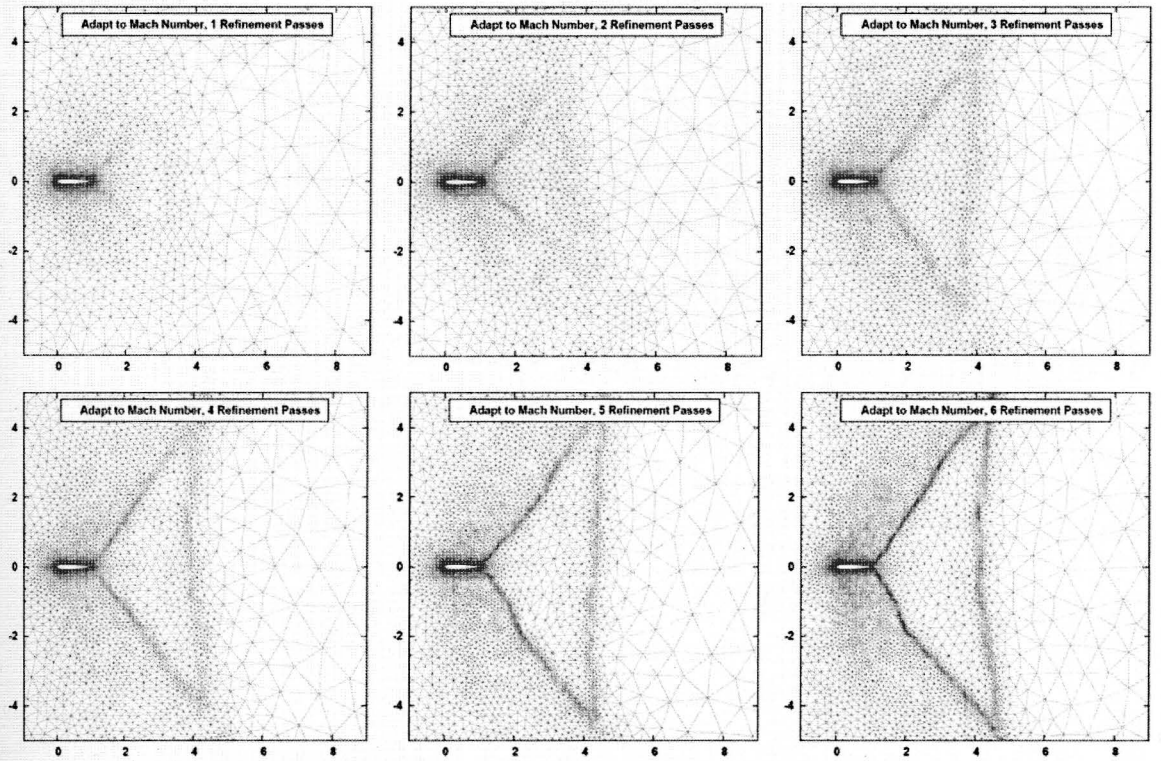


Figure 34 – Progression of grid refinement.

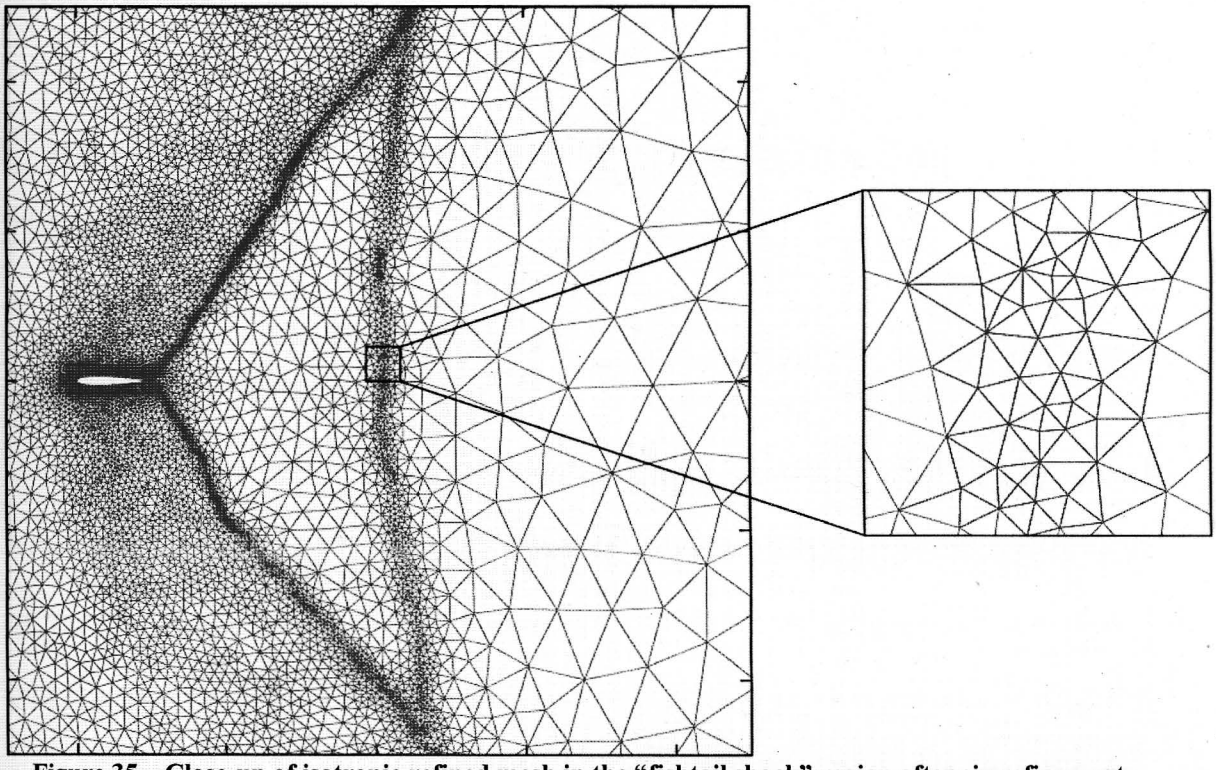


Figure 35 – Close-up of isotropic refined mesh in the "fishtail shock" region after six refinement iterations.

Chapter 4 – Winslow Equations on Boundaries

4.1 – Background

As noted in previous sections, the Winslow elliptic smoothing equations are derived from Laplace's Equations for a parameter distribution over a region. When applied to an unstructured mesh, the equations allow interior mesh points to be moved and smoothed to conform to a moving surface. Prior to the work outlined in this section, the Winslow equations have been applied only to interior mesh points on unstructured meshes. The following methodology allows for the implementation of the Winslow equations on boundary points as well.

The Winslow equations deal with derivatives of physical space (x,y) with respect to computational space (ξ,η). The equations, which were first defined in equation (3.2) are restated below:

$$\alpha\frac{\partial^2 \mathrm{x}}{\partial\xi^2}-2\beta\frac{\partial^2 \mathrm{x}}{\partial\xi\,\partial\eta}+\gamma\frac{\partial^2 \mathrm{x}}{\partial\eta^2}=0$$

$$\alpha\frac{\partial^2 \mathrm{y}}{\partial\xi^2}-2\beta\frac{\partial^2 \mathrm{y}}{\partial\xi\,\partial\eta}+\gamma\frac{\partial^2 \mathrm{y}}{\partial\eta^2}=0$$

The Winslow equations are applied to a control volume (in computational space) around each point in the mesh. By employing the Divergence Theorem, a set of surface

integrals, shown below, is generated on which the Winslow equations can be solved to smooth the physical coordinates of the mesh points.

$$\iint\left(\alpha\frac{\partial^2 x}{\partial\xi^2}-2\beta\frac{\partial^2 x}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 x}{\partial\eta^2}\right)dA=\oint\left(\alpha\frac{\partial x}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial x}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial x}{\partial\eta}\hat{n}_\eta\right)dS=0$$
$$\iint\left(\alpha\frac{\partial^2 y}{\partial\xi^2}-2\beta\frac{\partial^2 y}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 y}{\partial\eta^2}\right)dA=\oint\left(\alpha\frac{\partial y}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial y}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial y}{\partial\eta}\hat{n}_\eta\right)dS=0$$

In the above equations, $\hat{n}_\xi$ is the ξ component of the unit normal vector on the surface of the control volume in computational space and $\hat{n}_\eta$ is the η component of the unit normal vector on the surface. The surface integrals can be discretized as follows:

$$\begin{aligned}\sum\left(\alpha\frac{\partial x}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial x}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial x}{\partial\eta}\hat{n}_\eta\right)d\Gamma=0\\ \sum\left(\alpha\frac{\partial y}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial y}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial y}{\partial\eta}\hat{n}_\eta\right)d\Gamma=0\end{aligned}\qquad(4.1)$$

In the discretized equations, $d\Gamma$ is the length of a discrete surface segment. Equation (4.1) can be simplified by applying the relationship between the unit normal vector $(\hat{n})$ and a non-unit normal vector (n) on a given surface segment. Applying this relationship, which is $\hat{n}\Gamma = n$, simplifies equation (4.1) to (4.2).

$$\begin{aligned}\sum\left(\alpha\frac{\partial x}{\partial\xi}n_\xi-2\beta\frac{\partial x}{\partial\eta}n_\xi+\gamma\frac{\partial x}{\partial\eta}n_\eta\right)=0\\ \sum\left(\alpha\frac{\partial y}{\partial\xi}n_\xi-2\beta\frac{\partial y}{\partial\eta}n_\xi+\gamma\frac{\partial y}{\partial\eta}n_\eta\right)=0\end{aligned}\qquad(4.2)$$

To see an application of a control volume on which this would be applied, consider the real (though simple) 9-point unstructured mesh shown in Figure 36. In this figure, the

nodes are numbered in order to illustrate the mapping from physical space to computational space, where the Winslow equations are applied.

The only interior node in Figure 36 is node 8 and the computational space for node 8 is shown on the right hand side of Figure 36.The computational stencil for the interior point (node 8) is relatively straight-forward but no intrinsically obvious stencil exists for a point on a boundary, such as node 5. Recall from Figure 2 in Chapter 1 that the computational stencil of a boundary point is incomplete. In order to complete the computational stencil for a boundary point, the concept of ghost nodes is introduced. If, for example, it was desired that node 5 on Figure 36 be allowed to float, a ghost point could be placed as shown on Figure 37.

The ghost point, which in this case is node 9, allows the control volume in computational space to be closed. The closed virtual control volume making up the computational space for the surface node (node 5) can be seen on Figure 37.

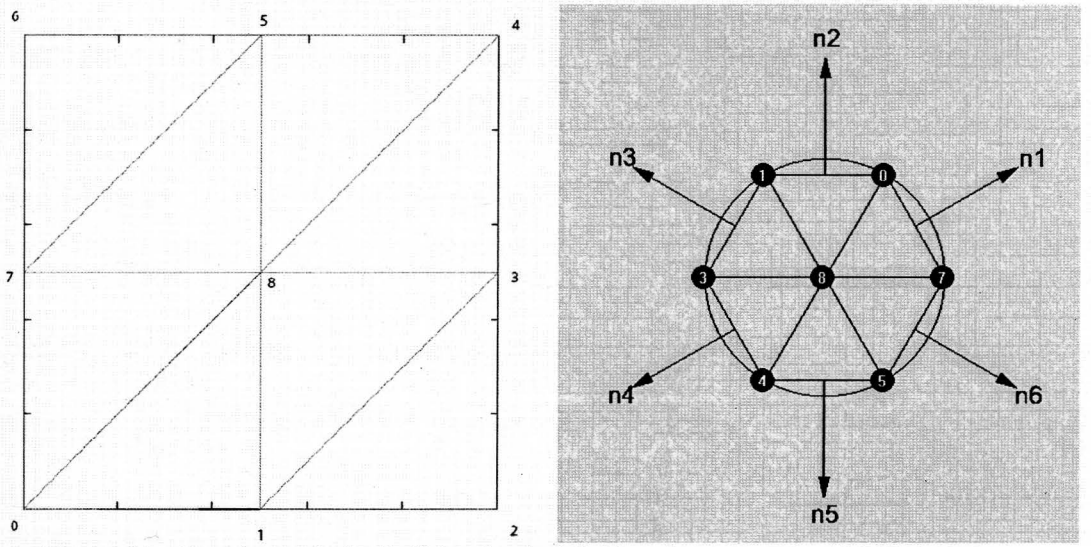


Figure 36 – Interior node (node 8) in physical (left) and computational (right) space.

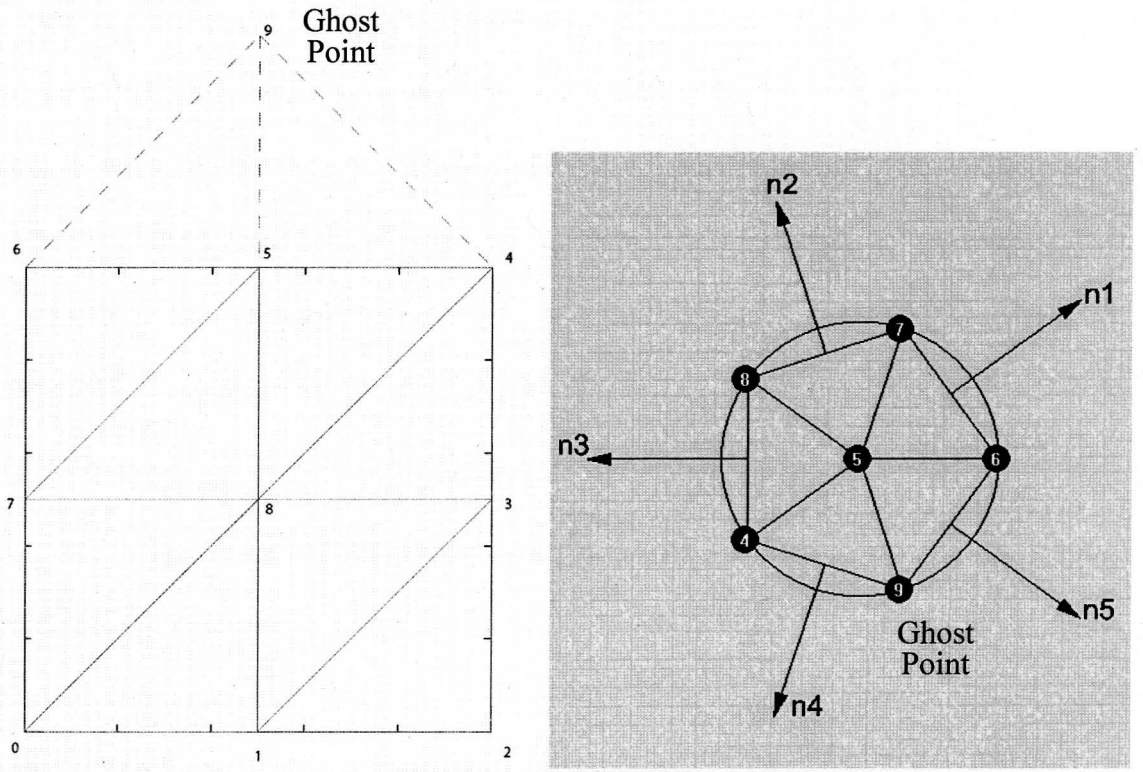


Figure 37 – Surface node (node 5) and ghost node (node 9) in physical (left) and computational (right) space.

4.2 – Placing Ghost Points

Consider a surface node such as node 5 in Figure 37. If a node on a surface is to be allowed to float, it will need an associated ghost point (such as node 9 in the previous figures) in order to close the virtual control volume to which the node is mapped in computational space. The Winslow equations are solved in computational space and the parameters they are solving for are the physical x and y coordinates of node 5 based on the physical x and y values of the connected neighboring nodes (nodes 6, 7, 8, 4 and 9). Because the value of x and y for node 5 is a function of the values of x and y for the connected nodes, it is necessary to place the ghost point at some reasonable location in physical space.

The first attempt at placing the ghost point was to place it at the same location as the associated boundary node. However, this had a fatal effect on the boundary points. For example, when the top outer boundary of a mesh surrounding a flat plate was allowed to float and the ghost point values were set to the values of their associated boundary nodes, the surface points were all driven either to the corners of the boundary or, because of the symmetry characteristics of the mesh, to the symmetry plane. This effect can be seen on Figure 38.

Upon further consideration, it makes sense that this method of placing ghost points would cause problems. To illustrate this, consider the control volume shown on Figure 37, which is in computational space. It is apparent from Figure 37 that node 5 will never be able to reach equilibrium with the surrounding points if one of the surrounding points is always being driven to the same value as the point at the center. This is what is happening as the attempt is made to generate a solution on the flat plate mesh. The central nodes in computational space that represent floating-node boundary points will never be able to reach a converged solution and nodes in physical space will be driven in some direction until the solver runs out of iterations.

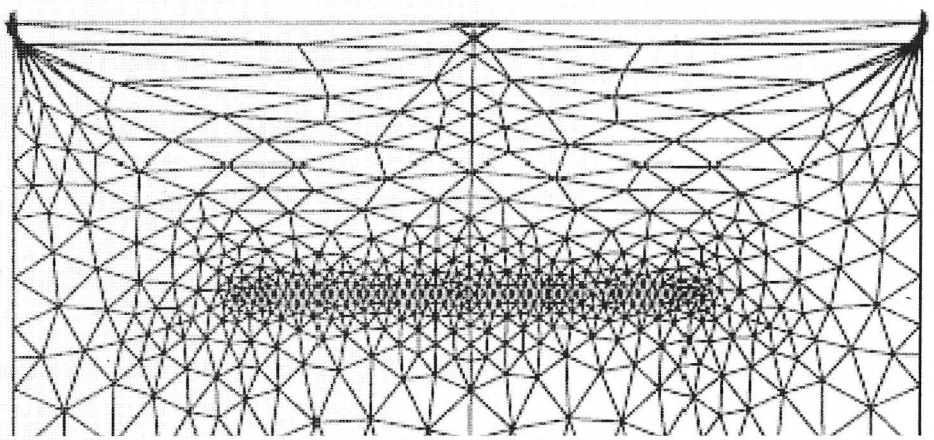


Figure 38 – Effect of co-locating ghost points with their associated surface points.

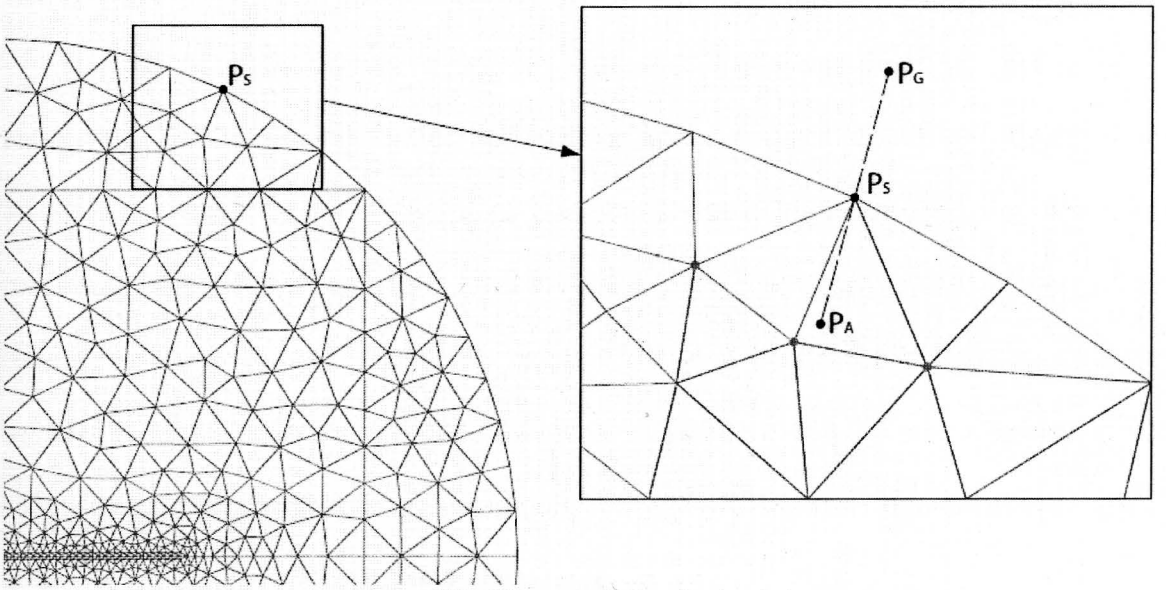


Figure 39 – Placement of a ghost point.

The next attempt at defining the coordinates of the ghost point involved calculating the average location of all interior points (P_A) connected to a surface point (P_S) and then extending the line connecting P_A and P_S. Once the line connecting P_A and P_S was extended, a ghost point (P_G) was then placed on this extended line at a distance equal to the distance between P_A and P_S.

To graphically illustrate the placement of the ghost points, consider the surface point shown as P_S on Figure 39. P_S has 3 interior nodes to which it is connected; these points are shown in green on Figure 39. The average interior point (P_A) is placed at the average coordinates of the three attached points. A line is then formed between P_A and P_S using the point slope equation of a line $\left(y-y_1=m\left(x-x_1\right)\right)$ and a ghost point, P_G, is placed on that same line at a distance outside the boundary equal to the distance between P_A and P_S.

4.2.1 – Reflection vs. Extension

Placing the ghost points using the extension technique described above gave reasonable results but, in order to most efficiently place the ghost points, the methodology used for the ghost point placement was explored further. As described previously, the first viable method for placing the ghost points involved extending a line connecting P_A to P_S and placing P_G on that line at a distance equal to the distance between P_A and P_S. Another possible technique would be to use a direct reflection of P_A about the boundary surface at P_S. The contrast in the placement of the ghost point using these two techniques is illustrated on Figure 40.

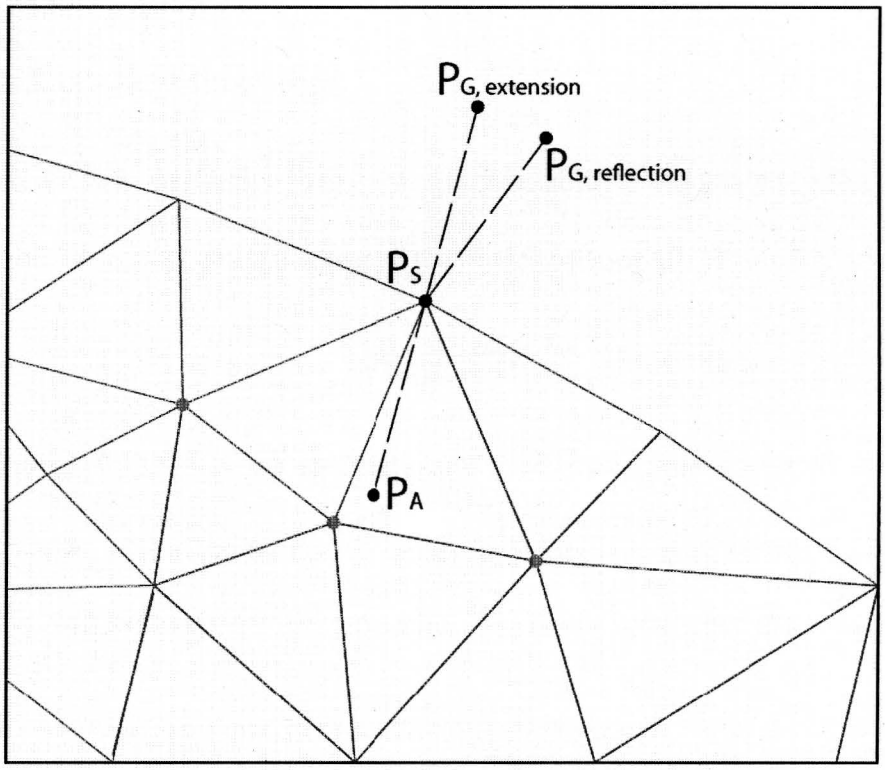


Figure 40 – Ghost point placement: reflection vs. extension

Intuitively, from examining Figure 39 and Figure 40, it appears that placing the ghost point using reflection would tend to be a better convention. Looking at the mesh in these figures, which has a circular outer boundary, it can be seen that P_A is at a location which is generally clockwise from P_S. However, placing the ghost point using extension will put the ghost point at a position which is slightly counterclockwise to P_S, thus diminishing some of the force that is driving the surface point in the desired direction (toward the average of the connected interior points). By using reflection to place the ghost point, the ghost point complements the interior points in driving the surface point to a new location rather than counterbalancing them.

4.2.2 – Reflection Methodology

If reflection is to be used, a matrix of transformation is used to place the ghost point by mapping P_A onto a reflected position. This transformation, $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, has the form:

$$T(\vec{x}) = [R]\vec{x} \tag{4.3}$$

In equation (4.3), $[R]$ is a standard matrix of linear transformation and $\vec{x}$ is the coordinate vector of the point that is being mapped to a new position. In order to calculate the standard matrix of reflection, one must construct the tangent vector, $\vec{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$, about which the point will be reflected. This tangent vector will represent a line tangent to the boundary surface at the point P_S. For a discretized boundary surface, the tangent can be approximated as $P_{s+} - P_{s-}$ where P_{s-} is the point on the surface before P_S

and P_{s+} is the point on the surface after P_S. Using the tangent vector of the line tangent to the surface at P_S, the standard matrix of reflection can be found as:

$$[R] = \frac{1}{d_1^2 + d_2^2}\begin{bmatrix} d_1^2 - d_2^2 & 2d_1^2 d_2^2 \\ 2d_1^2 d_2^2 & -d_1^2 + d_2^2 \end{bmatrix} \tag{4.4}$$

Once the matrix of reflection is known, the ghost point can be calculated as follows:

$$P_G = \begin{bmatrix} P_{Gx} \\ P_{Gy} \end{bmatrix} = [R]P_A \tag{4.5}$$

As expected, using the reflection technique to place the ghost points did improve convergence. However, the effect was not drastic. The meshes in Figure 41, Figure 42 and Figure 43 show a comparison that illustrates the difference in using the two techniques after 1000, 3000 and 5000 iterations respectively. It can be seen that the mesh that is generated using the reflection technique (shown in magenta) is "leading" the mesh that is generated using the extension technique (shown in blue) as the two meshes advance toward a converged solution. The effect is best observed by looking at lines coming off the ends and center of the flat plate, which are seeking a state such that they are normal to both the inner surface (the flat plate) and the outer surface.

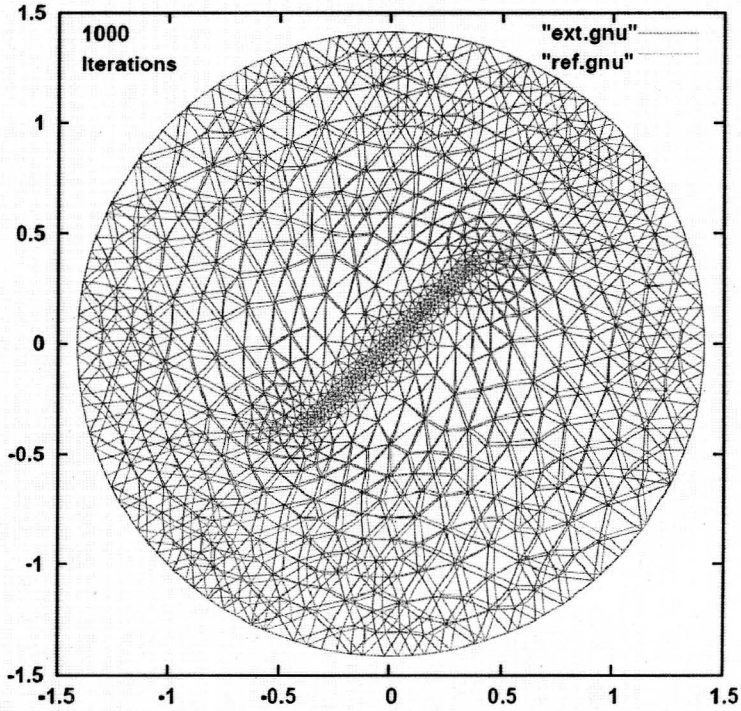


Figure 41 – Comparison of mesh using extension vs. reflection to place ghost points after 1,000 iterations.

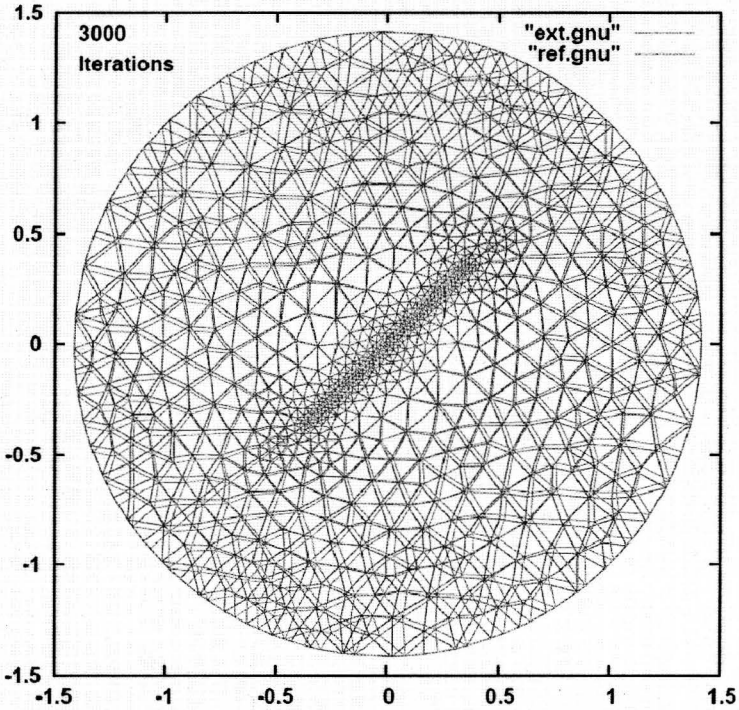


Figure 42 – Comparison of mesh using extension vs. reflection to place ghost points after 3,000 iterations.

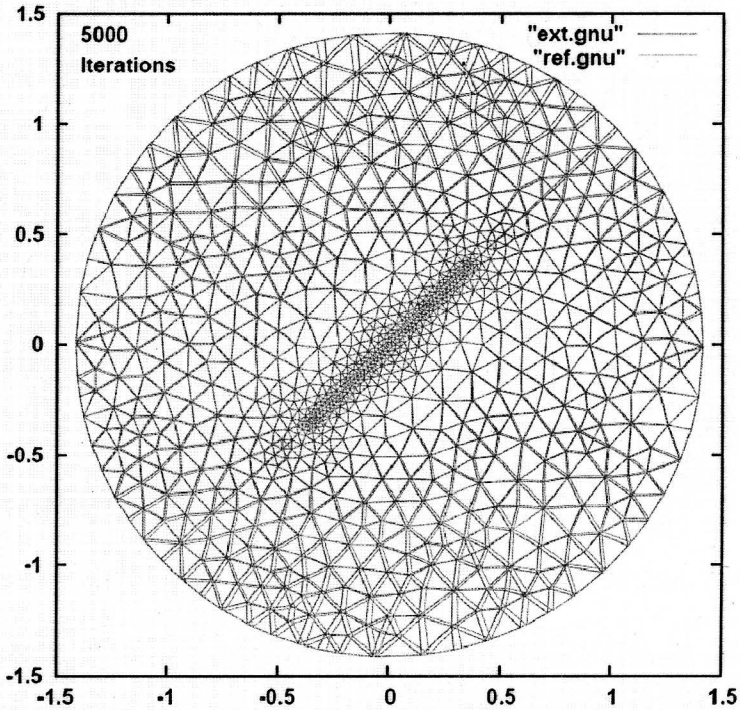


Figure 43 – Comparison of mesh using extension vs. reflection to place ghost points after 5,000 iterations.

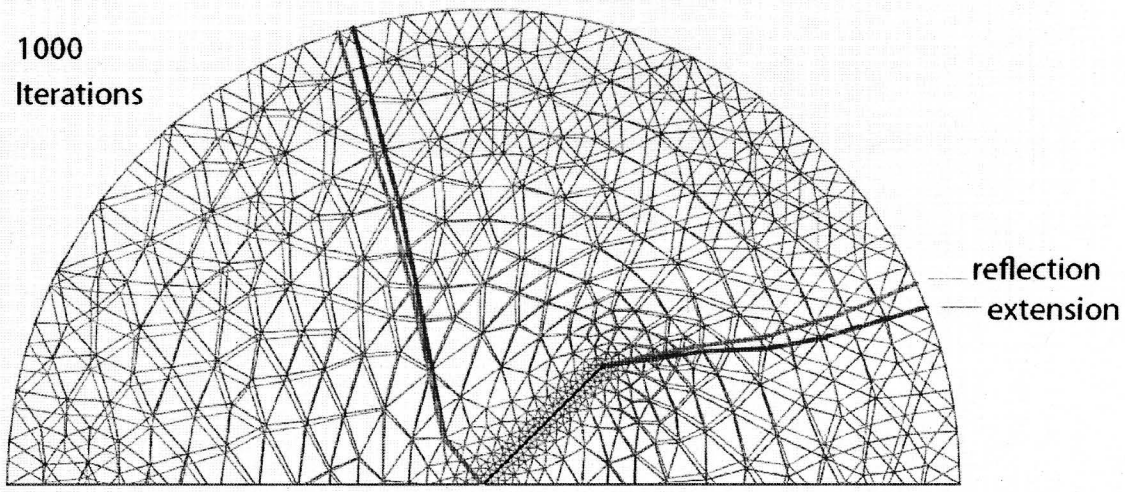


Figure 44 – Convergence comparison for a mesh using extension and reflection to place the ghost nodes.

In addition to the comparisons shown on Figure 41, Figure 42 and Figure 43, a more lucid view of the difference in the convergence using the two ghost-point placement methods (after 1,000 iterations) can be seen on Figure 44 above. The technique that is used to perform the coordinate reflection in two dimensions is extended to apply to three dimensions in Appendix C.

4.3 – Flat Boundaries

The first case that was used to test the methodology for manipulating surface points using the Winslow equations was a flat-plate surrounded by an unstructured mesh bounded by flat (constant in x or constant in y) boundary surfaces. This mesh can be seen in Figure 45. Recall that the Winslow equations are a set of 2 equations (for each node) that are solved for the x and y coordinates. Therefore, in the case where the boundary is flat (horizontal or vertical), only one of the equations will require a solution. For example, on the top boundary of Figure 45, the y-coordinate will not be changing so the second Winslow equation will not require a solution.

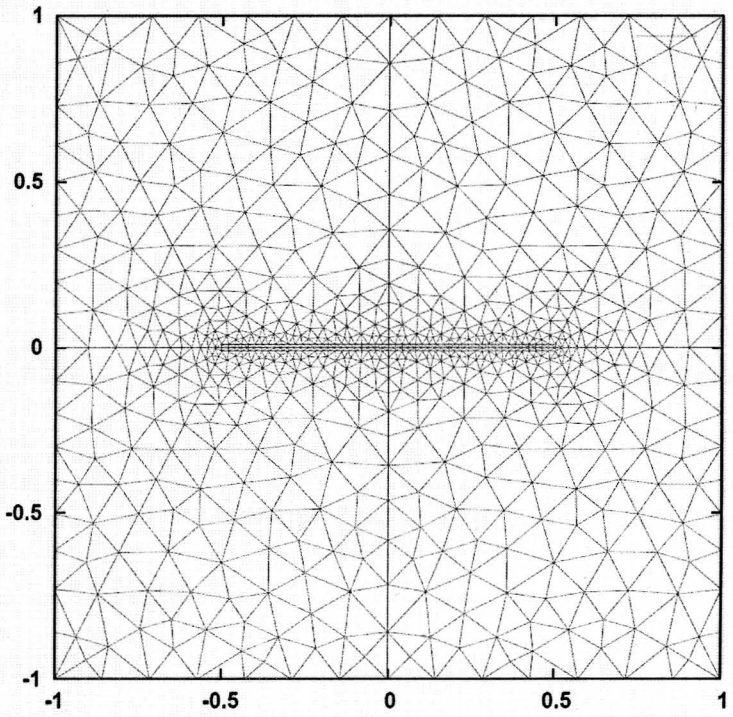


Figure 45 – Flat plate bounded by flat boundary surfaces.

The mesh shown on Figure 45 gave a good initial testbed for looking at the effects of a moving boundary. The flat plate surface embedded in the mesh in Figure 45 was translated, rotated and deformed into a semi-circle and the mesh was smoothed using the Winslow equations. The resulting meshes, with and without floating boundary points, are shown on Figure 46.

Figure 46 shows the benefit of allowing the nodes on the boundary to float. Without allowing the boundary nodes to float (i.e. when they are held static as they are on the left-hand side of Figure 46) the interior elements near boundaries exhibit large amounts of skew as the elements are forced to stretch to try to accommodate the moving inner boundary. When the nodes on the outer boundaries are allowed to float (as on the right-hand side of Figure 46) less skewing results because less element stretching is necessary.

The mesh shown on Figure 45 is of limited practical value because it has such a rigorous definition of what shape the outer boundaries must be (i.e., they must be constant in x or constant in y). For the cases with the flat outer boundaries, the implementation of the Winslow equations explicitly took this into account. The equations were implemented by only solving the equation for the coordinate that was moving. The nodes on the top boundary of Figure 45, for example, were not allowed to move in the y direction so the Winslow equation for movement in y did not even need to be solved. In general, this is not the case; in general, boundaries are described in one of two ways. They are defined analytically (circle, quadratic, spline etc.) or they are described by discrete segments. In either case, the Winslow equations for both x and y must be solved and the surface points must then be manipulated so that they conform to the original surface definition.

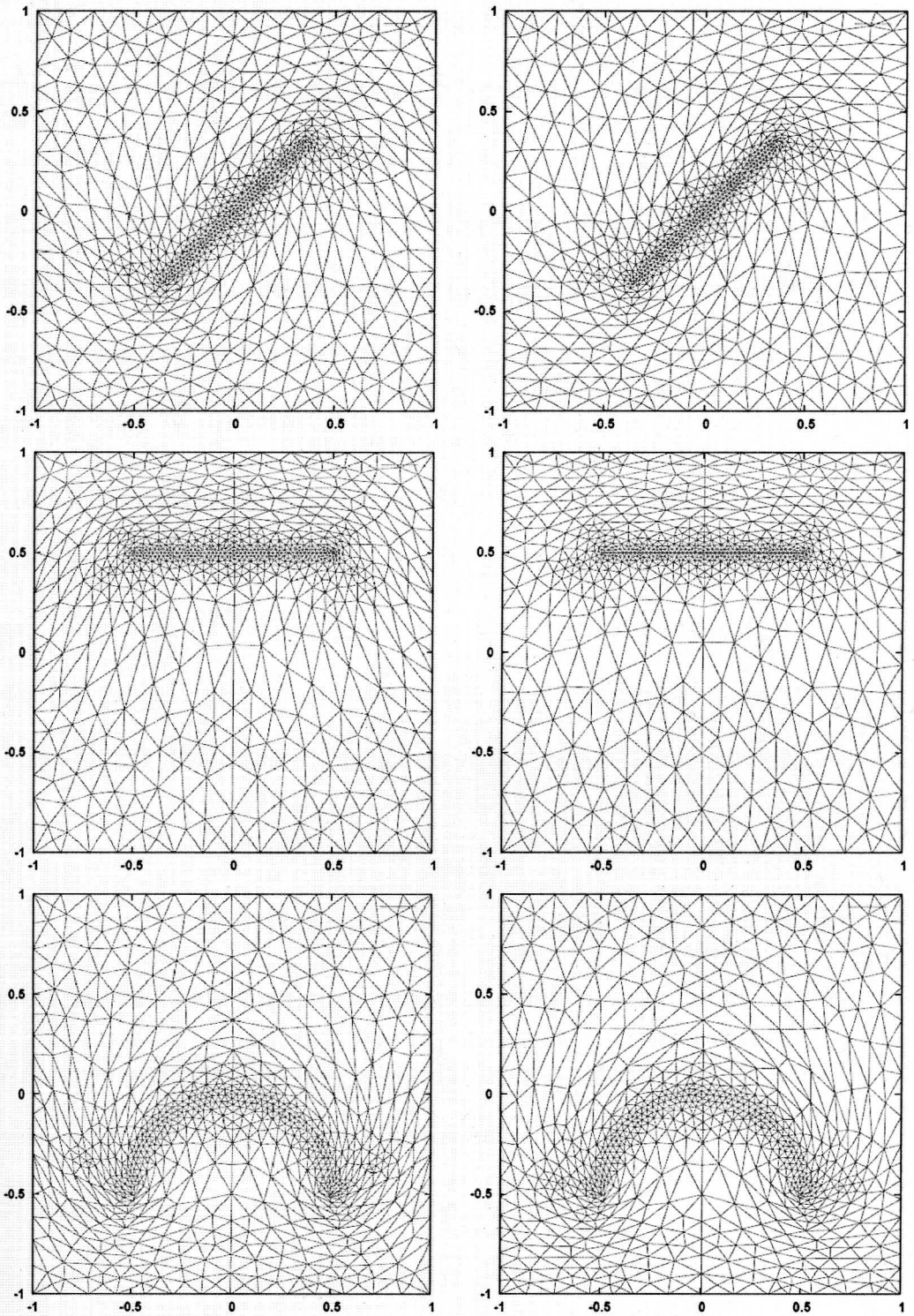


Figure 46 – Mesh smoothing with and without moving boundary nodes.

This figure shows the benefit of allowing the nodes on the outer boundaries to move. By allowing the boundary nodes to float (illustrated by the meshes on the right side) the skewness of the mesh has been reduced compared to meshes where the boundary nodes are fixed (illustrated by the meshes on the left side).

4.4 – Projecting a Node onto the Boundary

The algorithm that performs the mesh movement is a point iterative scheme where the coefficients in the Winslow equations, α, β, and γ, must be recomputed at each iteration. Each time an iteration is performed, all of the nodes will move to a point that the smoothing algorithm considers the optimal placement for that iteration. For interior points, this new position will be the starting place for the next iteration. However, for boundary points there is an additional step.

At each iteration, each boundary point that has moved must be projected back to the surface that defines the original boundary. If this is not done, the boundary will begin to wander away from its original position to a position that better satisfies the elliptic smoothing equations and not conform to the true boundary. Projection methods were implemented for two different boundary types: analytically and explicitly defined boundaries.

4.5 – Analytically Defined Boundaries

It has been the author's experience that the most common analytically defined outer boundary used in two-dimensional external-flow CFD is a circle. This boundary, for a region where the center is at the origin, can be defined by the expression $x^2 + y^2 = r^2$ where x and y are the coordinates of a boundary point and r is the radius of the outer boundary. Once Winslow smoothing has been performed, x and y will be modified and now $x^2 + y^2 \neq r^2$. The boundary point must then be moved to the closest point which is

on the original boundary surface definition. It can be shown (See Appendix D) that for a point, $p = (x_p, y_p)$, the closest point on a circle with radius r is the point of intersection between the circle and a ray originating from the origin and passing through point p. This ray can be defined by the equation for a line: $y = mx + b$ where the slope is $m = \frac{y_p}{x_p}$ and the y-intercept, b, is zero. There are now two equations and two unknowns for the closest point:

$$y = mx$$

$$x^2 + y^2 = r^2$$

Solving the equations for x gives:

$$x^2 + m^2x^2 = r^2$$
$$x^2(1 + m^2) = r^2$$

$$x = \pm\sqrt{\frac{r^2}{1 + m^2}} \tag{4.6}$$

In the "closest point" section of the Winslow smoothing code, the logic for determining the smoothed x and y coordinates of a boundary point is coded up as follows:

if(x_P = 0 and y_p = 0) → an error will be generated because if the point is at the origin, all points on the boundary are exactly the same distance away and the closest point is undefined.

If only $x_p = 0$ then the slope is zero and the closest point is found by inspection as:

$$\text{if}(x_P = 0 \text{ and } y_p < 0) \rightarrow \quad x = 0 \text{ and } y = -r$$

$$\text{if}(x_P = 0 \text{ and } y_p > 0) \rightarrow \quad x = 0 \text{ and } y = r$$

If neither x_p nor y_p are zero, then:

$$\text{if}(x_p<0) \rightarrow \quad x = -\sqrt{\frac{r^2}{1+m^2}} \text{ and } y = mx$$

$$\text{if}(x_p<0) \rightarrow \quad x = \sqrt{\frac{r^2}{1+m^2}} \text{ and } y = mx$$

4.5.1 – Flat Plate Results Using Analytically Defined Boundaries

The logic described above was applied to a mesh whose outer boundary was analytically defined as a circle. This mesh, prior to any smoothing, can be seen below on Figure 47.

The nodes on the inner flat-plate surface of Figure 47 are described in some position (either their original position or deformed in some way) and then the rest of the mesh, including all of the nodes on the outer surface, are allowed to float. The mesh is smoothed using the Winslow equations and, at each iteration, the nodes on the outer surface are projected to the nearest point on the analytically-defined surface so that the outer surface definition is not changed. The results for a mesh surrounding a flat plate

that has been translated, rotated, deformed and both rotated and deformed are shown on Figure 48 through Figure 51.

An interesting comparison can be made if the mesh is smoothed in its original position and then smoothed again after the flat plate has been rotated 45 degrees. The mesh, including the outer boundaries, should look the same, just rotated 45 degrees. Comparing the two meshes in Figure 52 shows that this is indeed what happens.

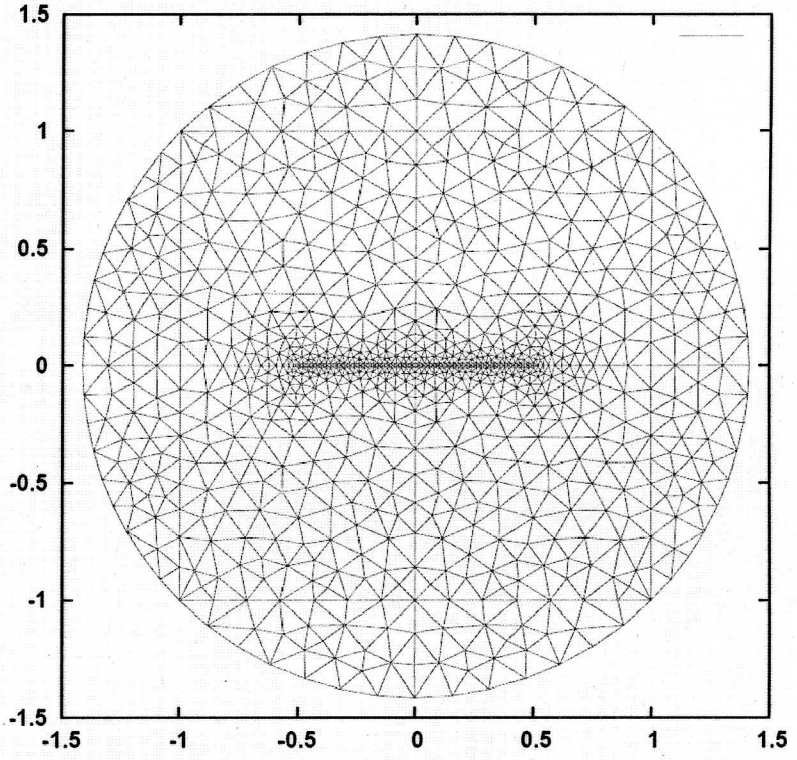


Figure 47 – Mesh region with outer boundary analytically defined as a circle.

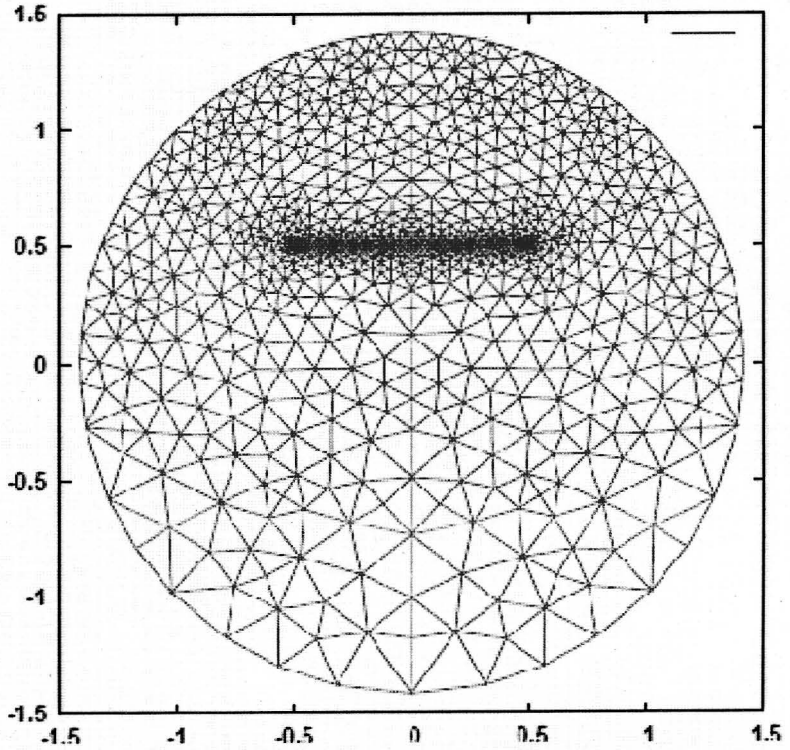


Figure 48 – Translated flat plate with floating-node outer boundary.

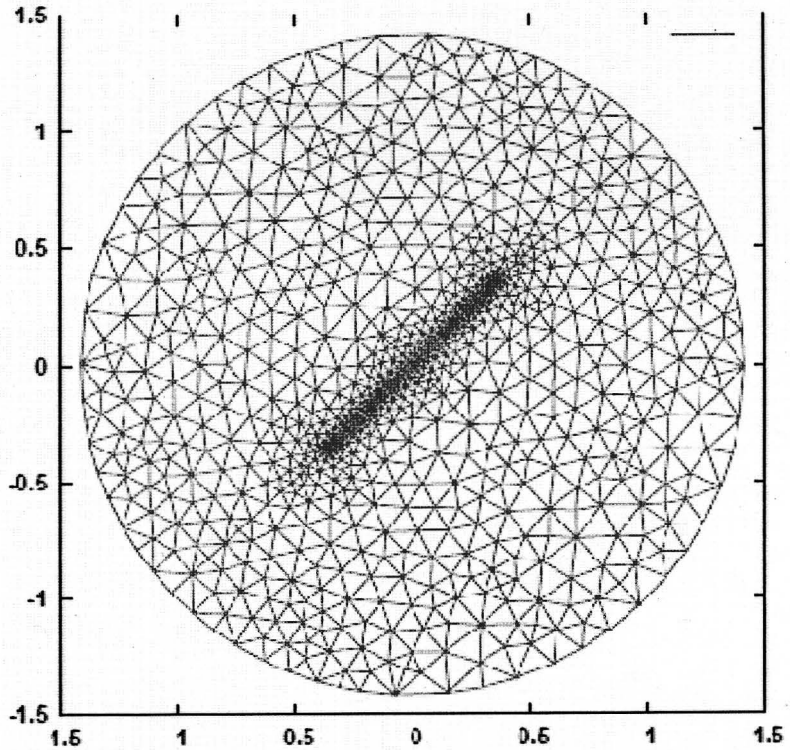


Figure 49 – Rotated flat plate with floating-node outer boundary.

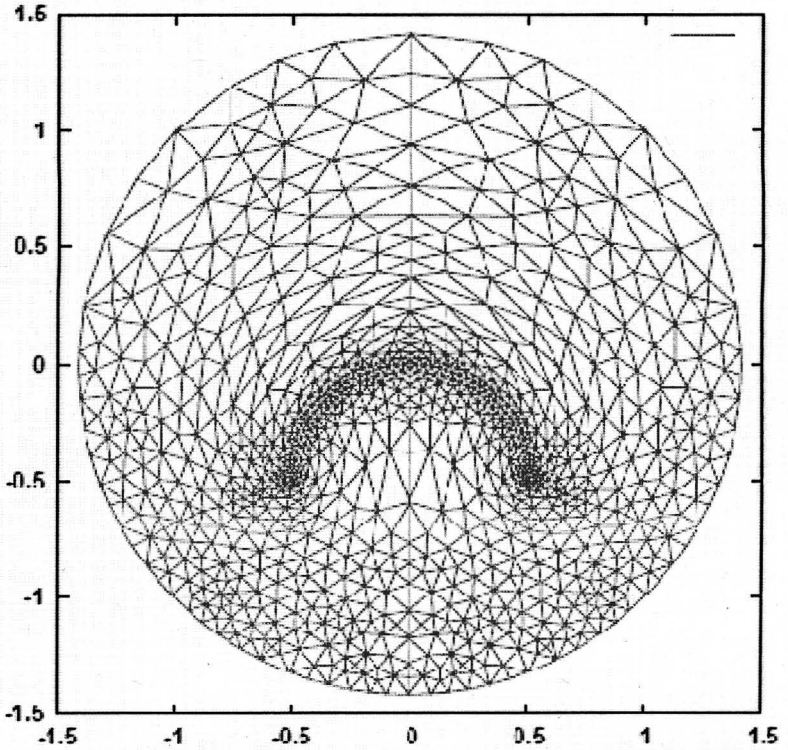


Figure 50 – Warped flat plate with floating-node outer boundary.

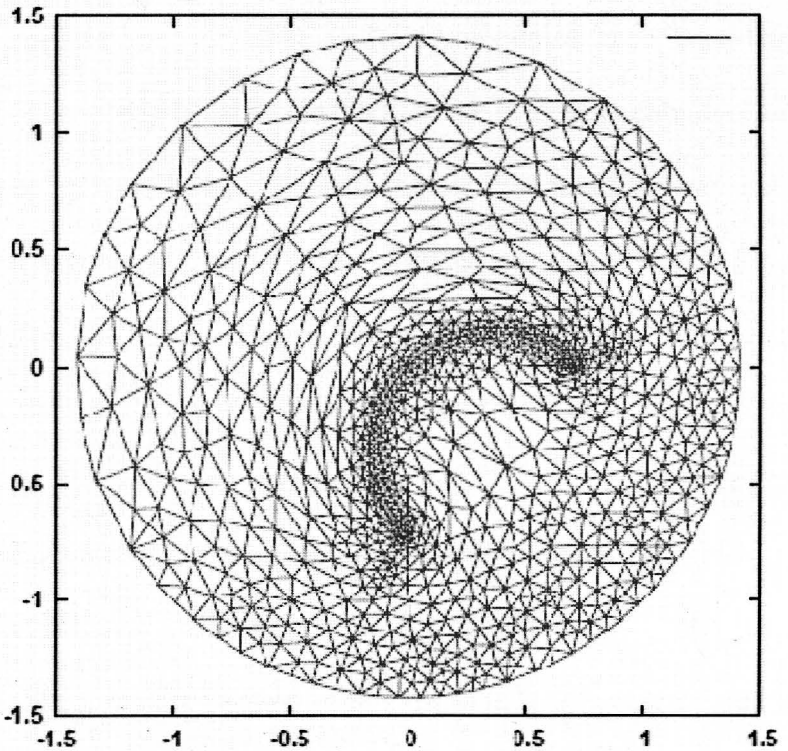


Figure 51 – Warped and rotated flat plate with floating-node outer boundary.

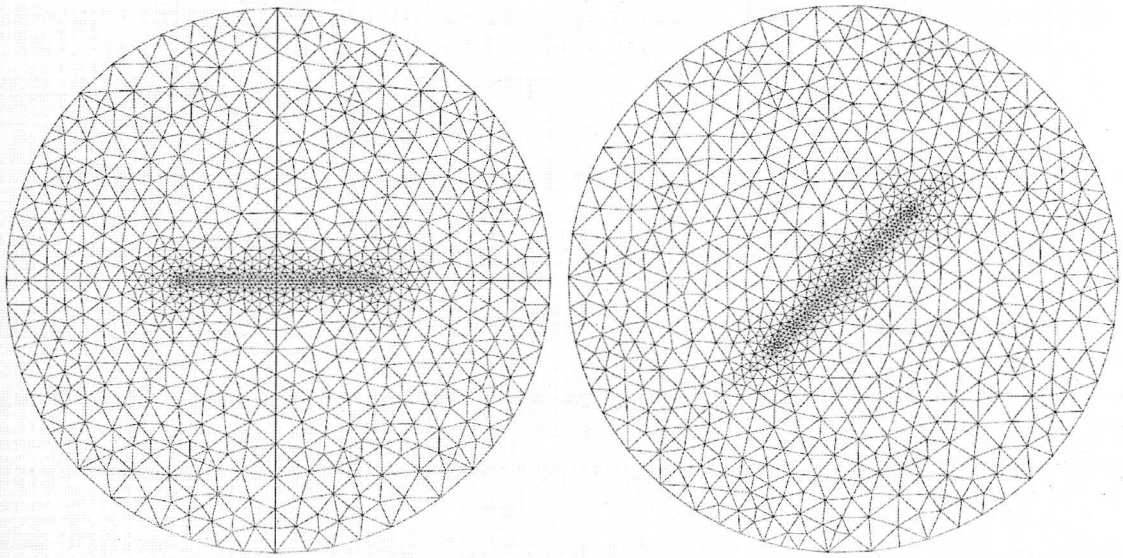


Figure 52 – Comparison of a smoothed mesh, before and after rotation.
Allowing the nodes on the outer boundary to float allows the rotated mesh to closely mimic the characteristics of the original mesh, just rotated 45 degrees.

4.6 – Explicitly Defined Boundaries

Although meshes used in CFD analysis sometimes have boundaries defined by analytical functions as described in the Section 4.5, it is more common for boundaries to be defined explicitly by a discrete shape that is defined in a geometry file. The nature of the geometry file will vary but, for two-dimensional analysis, the geometry file will generally take the form of a segment file where the shape is defined by discrete line segments. Any arbitrary level of accuracy in capturing the boundary shape or curvature can be achieved by increasing the number (and shortening the lengths) of the line segments in the geometry file. In three-dimensional analysis, the file will generally take the form of a tessellated surface file such as a VRML or STL file. Because boundary surfaces are most commonly defined in this manner, it is necessary that a floating-node

boundary on a mesh which is to be smoothed is able to adhere to a boundary explicitly defined in this way.

Projection onto an explicitly defined boundary is achieved by utilizing a C++ "geometry" class that was developed by Dr. Steve Karman at The University of Tennessee at Chattanooga (UTC). Using this class, the boundary is defined by a discrete geometry file, which is comprised of a user-defined number of line segments on the surface. The number of segments will vary depending on the desired tolerance of the projection onto the surface. The geometry file on which the geometry surface is based can be easily generated using Gridgen by exporting a curve (or number of curves) under the INPUT/OUTPUT commands in the Gridgen interface [12].

Once a geometry object is read in from the geometry file, a closest-point method (which is defined within the C++ geometry class) is used to project the surface node back onto the surface after it has been moved due to Winslow Smoothing. It was desired that the smoothing and projection methodology be tested on a shape other than just a simple circular surface so a discrete outer boundary resembling an ellipse was created and a flat-plate was manipulated within this new boundary. The smoothing of this mesh is demonstrated in the images shown on Figure 53 below.

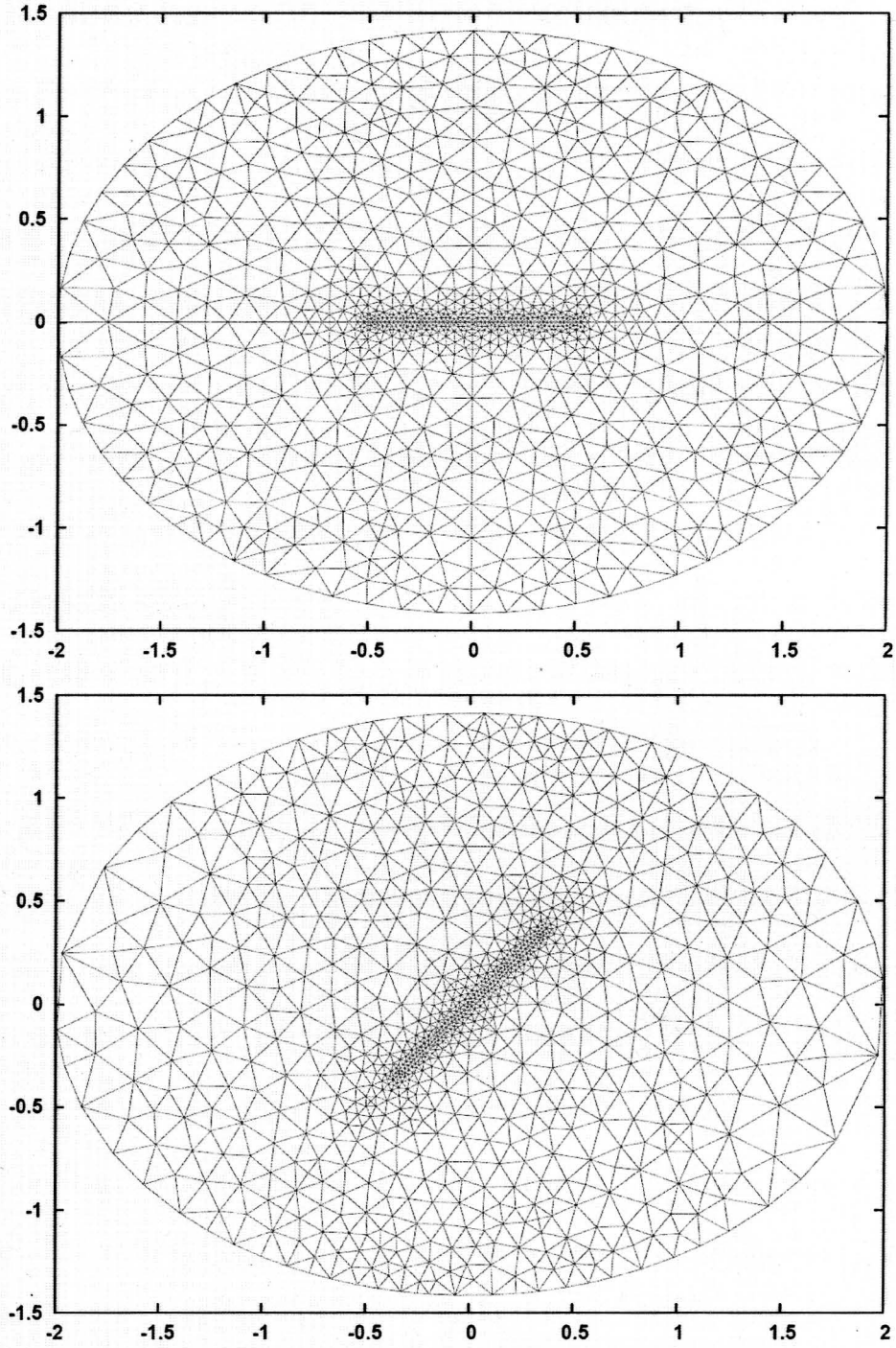


Figure 53 – Winslow smoothing on a mesh with an explicitly defined elliptically shaped outer boundary.

The top image shows the mesh in the original position and the bottom image shows the mesh after the embedded flat plate has been rotated and the mesh has been smoothed.

4.7 – Floating Points on Multiple Boundaries

Once the Winslow elliptic smoothing equations were working correctly on a single boundary, the code was extended to be able to handle moving points on multiple boundaries. To test this, a mesh was generated that had a structure similar to the flat-plate mesh but, instead of a flat plate, the inner surface was a NACA0012 airfoil.

When dealing with a shape such as an airfoil, which has two portions of the surface that are relatively close together, it is necessary to split the surface into an upper and lower portion and limit the surface-point projection to the original surface. Otherwise (e.g., at the sharp tail of an airfoil) a node from the upper surface can easily be projected onto the bottom surface if the node is moved in between the two surfaces when smoothing is performed. This undesirable effect is illustrated on Figure 55.

A separate boundary condition was added to the code to handle boundaries that were comprised of multiple sections. The boundary condition was implemented in such a way that when the mesh was smoothed, the endpoints remained in their original position while the remaining points on the boundary were allowed to float. This was different from the previous implementation where all surface nodes, even the nodes at endpoints, were allowed to float. When this new boundary condition was implemented, the mesh surrounding the interior (airfoil) surface acted as expected. The results can be seen below on Figure 56.

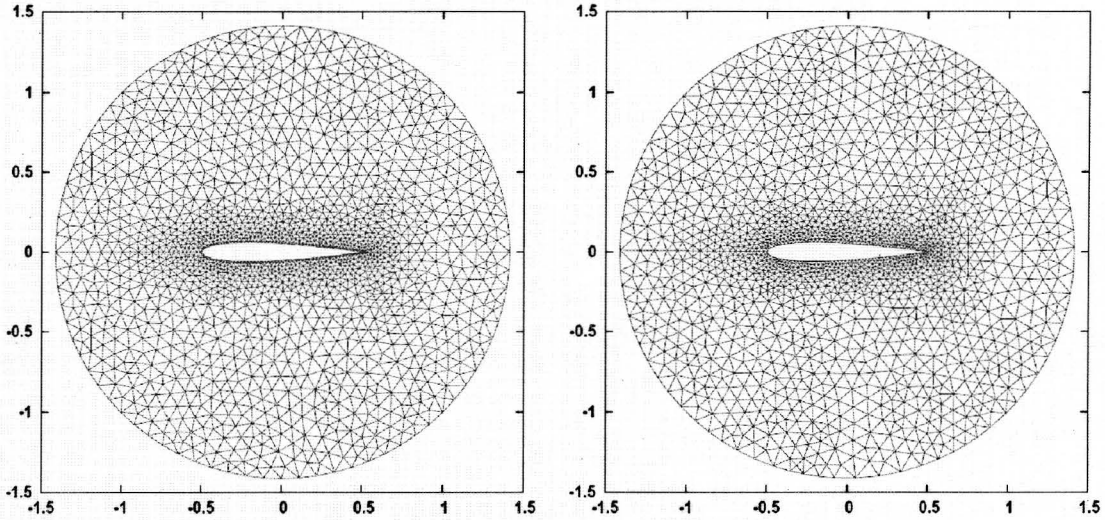


Figure 54 – Airfoil mesh before and after smoothing.

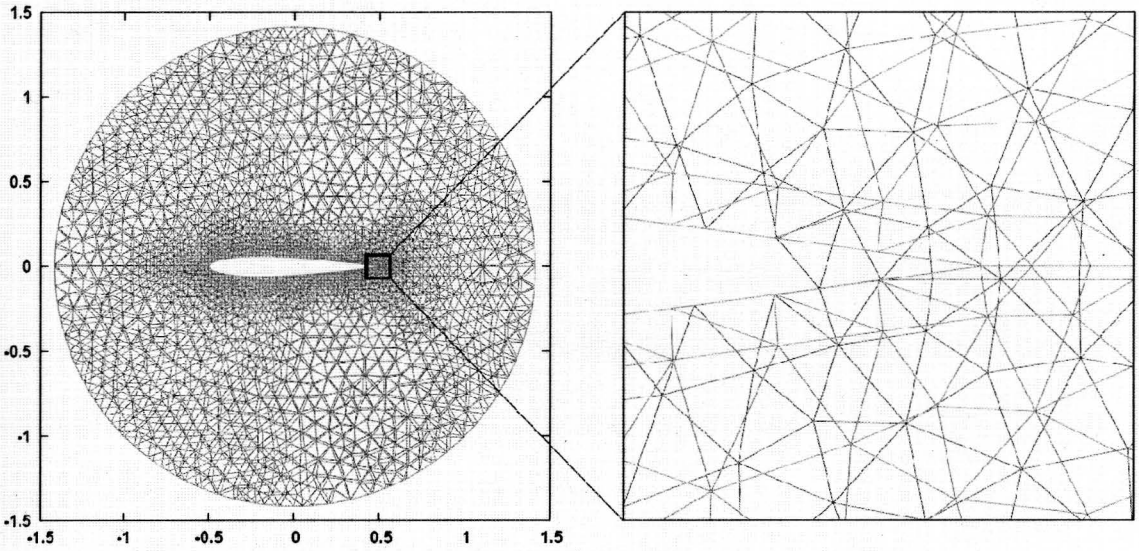


Figure 55 – Effect of projecting onto the wrong surface of a sharp airfoil tail.

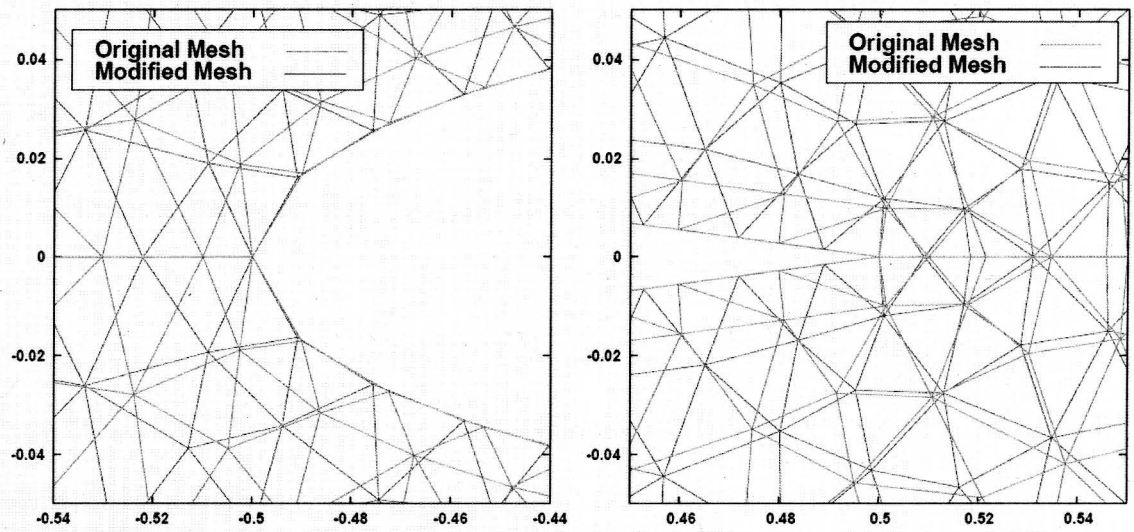


Figure 56 – Close-Up of mesh near interior surface at the nose and tail of an airfoil.
In this figure, the airfoil surface was defined as two distinct surfaces (a lower surface and an upper surface) and the surface mesh was confined to its original surface. This eliminated the possibility of a node from the upper surface being projected onto the lower surface if the smoothing equations placed the node in between the two original surface locations.

As seen on Figure 56, the modified mesh around the airfoil surface is comprised of what would normally be considered very "good" triangles in that they are close to equilateral. However, note from Figure 56 that in order to equalize the angles in the surface triangles, the mesh points were driven away from the airfoil surface. This might by a good mesh for an inviscid flow but, if it is desired to capture the viscous effects of a flow within a viscous boundary layer, this will have a negative impact on capturing the correct characteristics of the flow in this region. To alleviate this effect, the smoothing logic must be modified such that the computational space (which currently utilizes equal angles) can be utilized to try to better retain the characteristics of the original mesh. This is discussed in detail in chapter 5.

4.8 – Multiple Element Airfoil

4.8.1 – Airfoil Rotation

The final case that was examined to explore the effects of applying the Winslow equation to the boundaries was to apply Winslow elliptic smoothing to a multi-element airfoil. The airfoil that was employed for this purpose was the 30P30N multi-element high-lift airfoil. The 30P30N is a three-element airfoil consisting of a central airfoil section with a flap section in the rear and a slat section in the front. The mesh that was used was a relatively dense mesh with 23,012 mesh points and 44,437 triangular elements. The original configuration of the airfoil was with the flap and slat extended (the high-lift configuration); the entire mesh can be seen on Figure 57 and a close-up view near the airfoil surface can be seen on Figure 58.

The 30P30N airfoil was pitched 30 degrees counter clockwise and the surrounding mesh was smoothed using the Winslow equations to adapt the flowfield mesh to the new airfoil surface position. The mesh was smoothed using both a static outer boundary and a floating-node outer boundary, which can be seen on the left and right sides of Figure 59, respectively. Initially, because of the dense mesh and the abundance of levels of connectivity between the inner and outer surfaces, it was unclear if the outer boundary would be greatly affected by the treatment of the outer boundary nodes (static vs. floating). After the cases were examined, it was determined that, in fact, the outer boundary was significantly affected. As shown on Figure 59, when a floating-node boundary was employed, the nodes on the outer boundary floated in a counter-clockwise direction until the final mesh closely resembled the original mesh, just rotated 30 degrees.

Four representative outer-boundary mesh points are marked in black on Figure 59 so the rotation can be easily discerned.

A close up of the mesh near the airfoil (before and after the 30^{o} pitch) is seen on Figure 60 and Figure 61. In order to test the quality of the mesh, a CFD flow solution was generated using an in-house 2D Euler solver for the airfoil at both 0^{o} and 30^{o} angle of attack using a freestream Mach number of 0.95 and the results are shown on Figure 62 and Figure 63. The solver did not have any problems generating a flow solution on either the zero angle of attack or the rotated mesh.

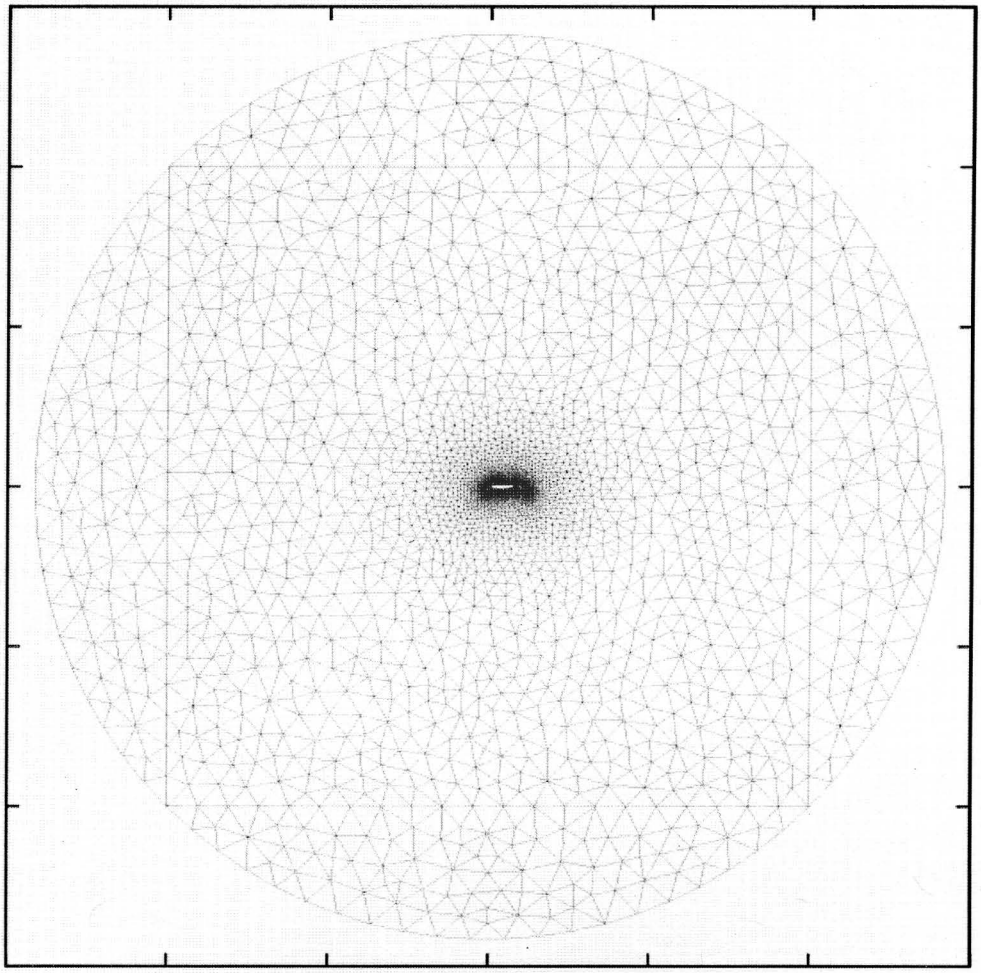


Figure 57 – 30P30N airfoil and mesh.

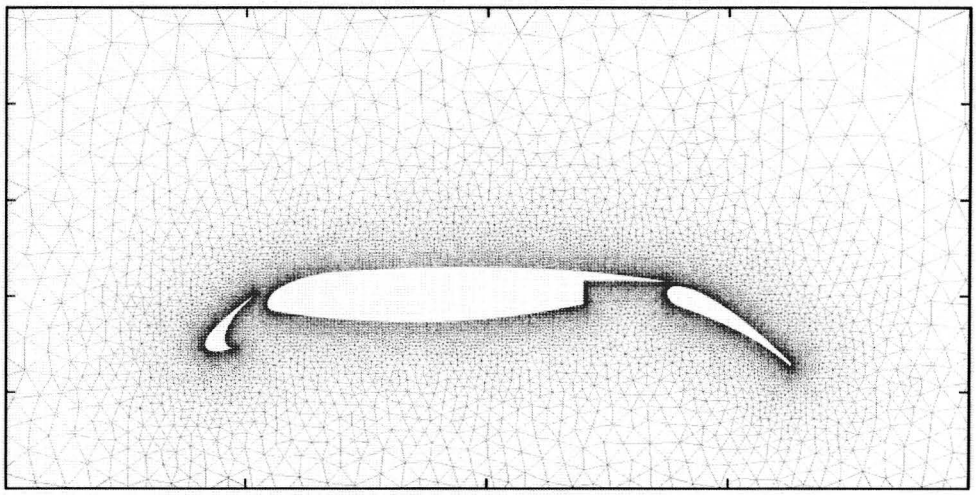


Figure 58 – Close-up of 30P30N airfoil.

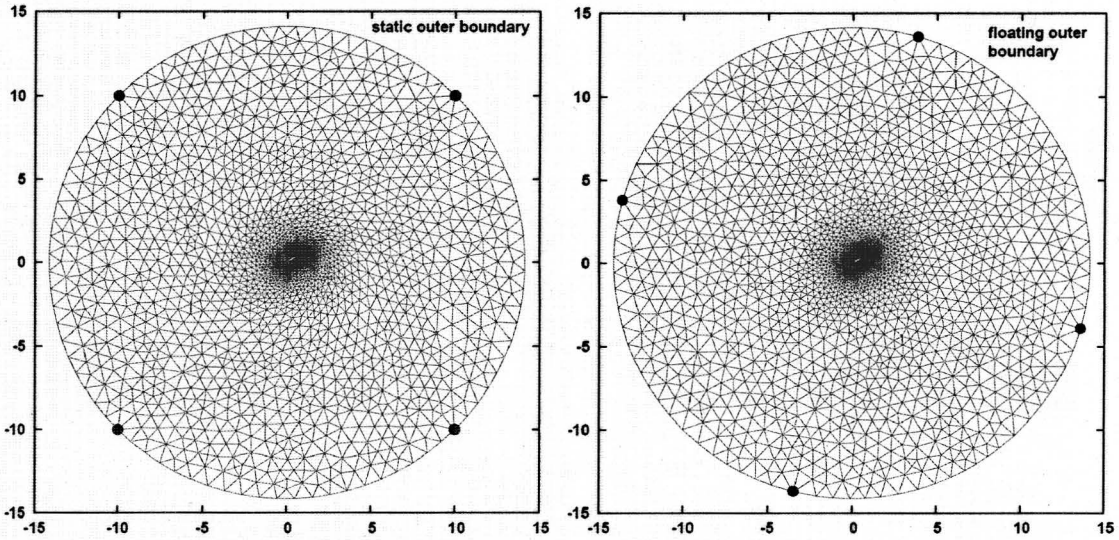


Figure 59 – 30P30N mesh, without (left) and with (right) a floating-node outer boundary.

When a floating-node boundary was employed, the nodes on the outer boundary floated in a counter-clockwise direction until the final mesh closely resembled the original mesh, just entirely rotated 30 degree. Four representative outer-boundary mesh points are marked in black so the rotation can be easily discerned.

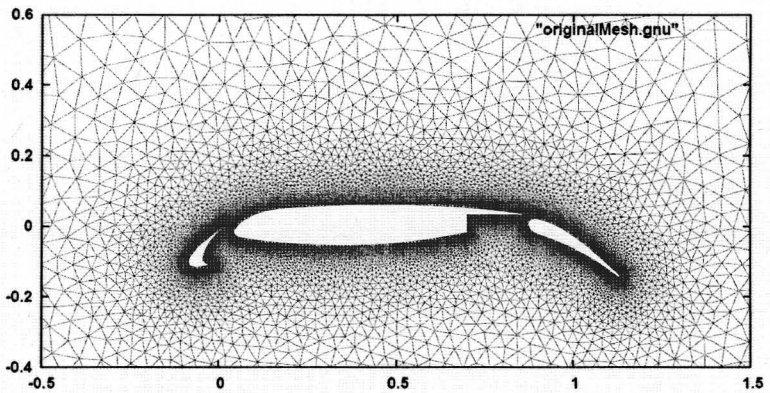


Figure 60 – 30P30N airfoil at zero degree angle of attack.

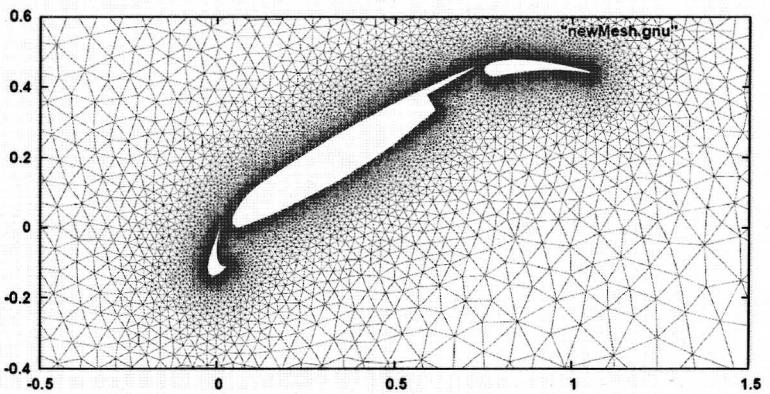


Figure 61 – 30P30N airfoil at thirty degrees angle of attack.

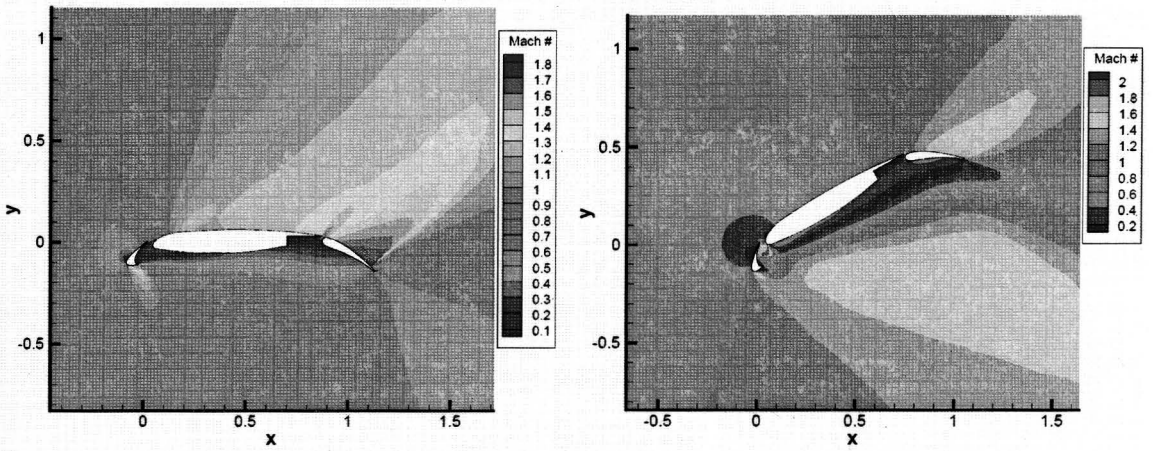


Figure 62 – CFD Solution on a 30P30N airfoil.

4.8.2 – Airfoil Slat Movement

Because the goal of analyzing the 30P30N airfoil was to demonstrate the power of the Winslow Equations applied to a boundary, it made sense to look at how two surfaces that were close together behaved if one of the surfaces was moved relative to the other. For this, the slat at the front of the airfoil was rotated from its original extended (high-lift) position to a retracted position. The slat was rotated over two steps and the change in position can be seen on Figure 63.

The mesh was smoothed at each of the two slat rotation steps. The first case that was examined was using static nodes on all of the airfoil boundaries. When this was done, the mesh between the slat and the central airfoil section became highly skewed. These results can be seen on Figure 65. The next case that was examined involved letting the boundary points on the surface of the central airfoil section float as the slat was rotated. This gave a much better mesh near the leading edge of the central airfoil section and these results can be seen on Figure 66.

The results from the exploration of the 30P30N multi-element airfoil show one practical "real world" case where utilizing the ghost node technique for applying the Winslow elliptic smoothing equations to the boundaries of an unstructured mesh resulted in a significant improvement compared to the case where static-node boundaries were used and only the interior mesh was smoothed.

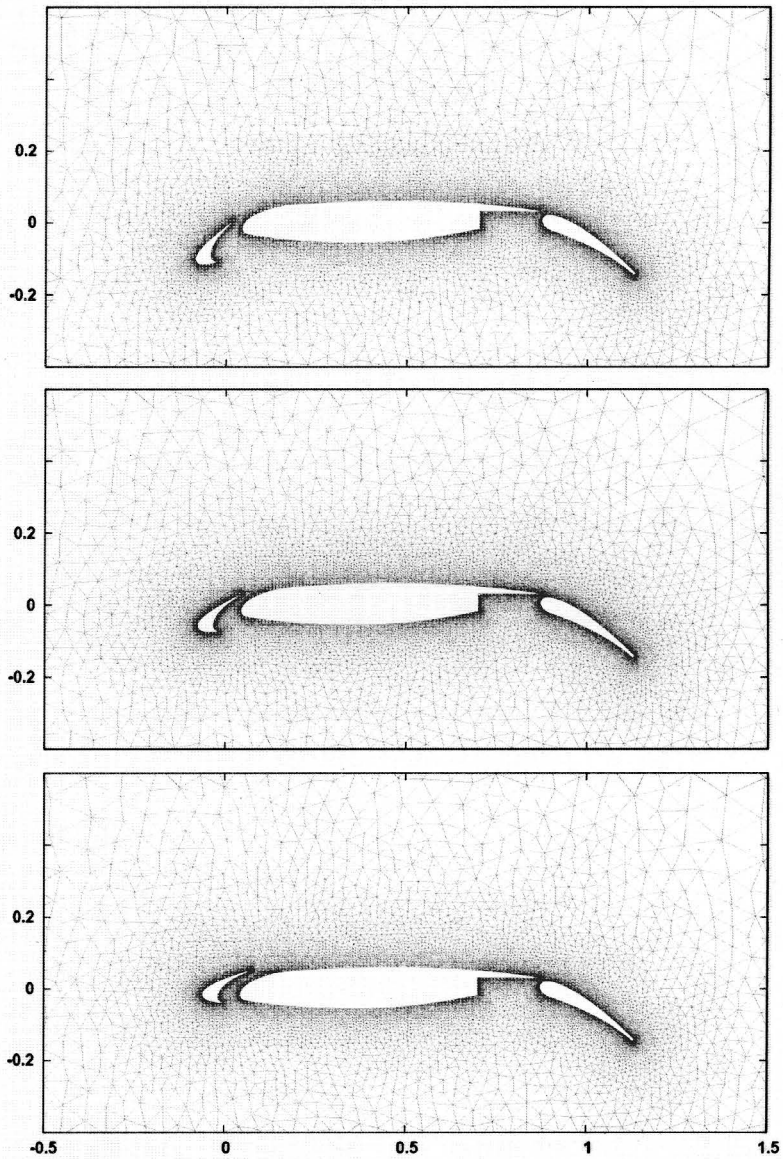


Figure 63 – 30P30N airfoil with front (slat) section moved from the extended to the retracted position.

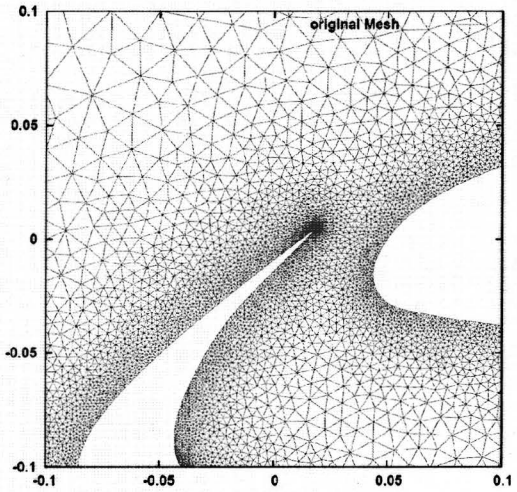


Figure 64 – Original slat position.

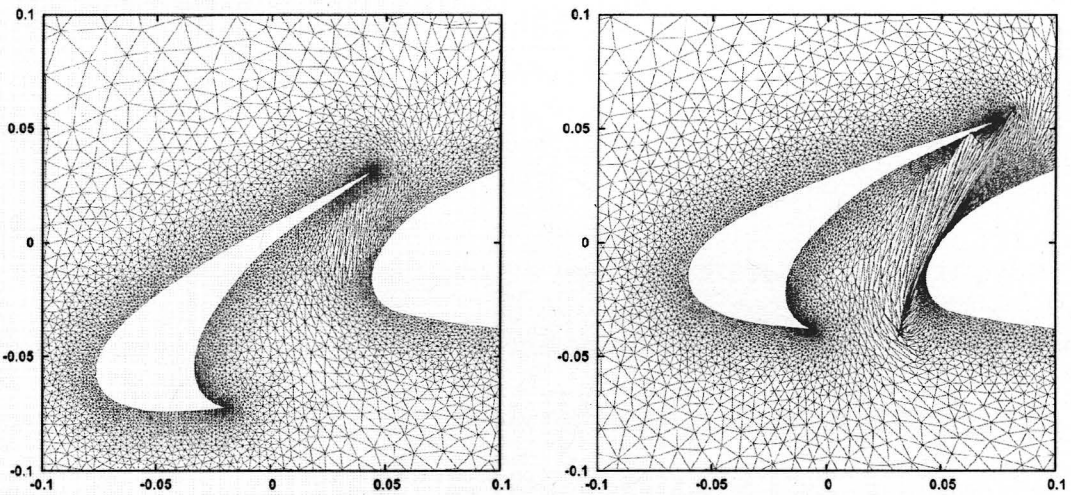


Figure 65 – Slat movement using static surface nodes.

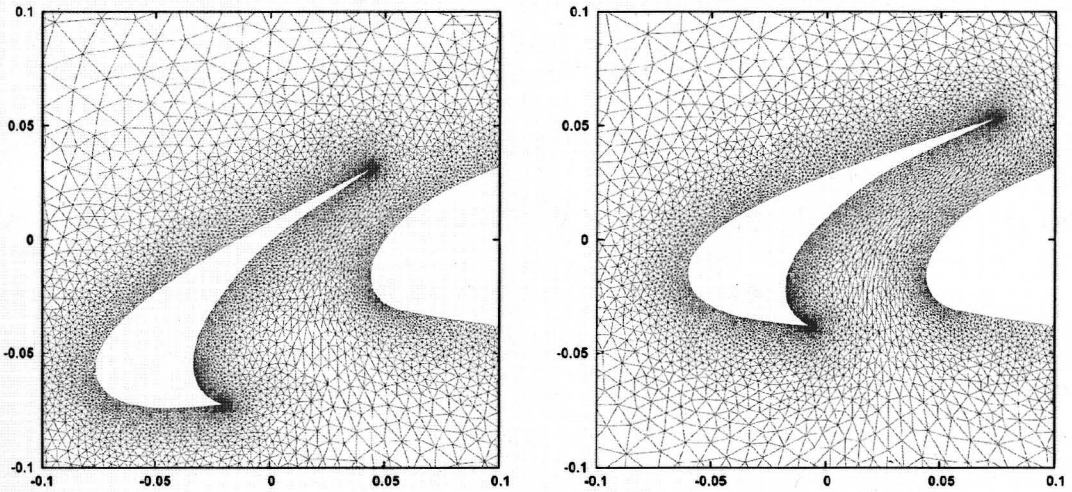


Figure 66 – Slat movement using a floating-node boundary on the central airfoil section.

Chapter 5 –Viscous Region Manipulation on a Flat Plate

5.1 – Equal Angle, Equal Edge-Length Computational Space

The Winslow elliptic smoothing equations, which are applied to a node in computational space, are implemented using the discretized form of the equations first shown on equation (4.2) and repeated below.

$$\sum\left(\alpha\frac{\partial x}{\partial\xi}n_{\xi}-2\beta\frac{\partial x}{\partial\eta}n_{\xi}+\gamma\frac{\partial x}{\partial\eta}n_{\eta}\right)=0$$

$$\sum\left(\alpha\frac{\partial y}{\partial\xi}n_{\xi}-2\beta\frac{\partial y}{\partial\eta}n_{\xi}+\gamma\frac{\partial y}{\partial\eta}n_{\eta}\right)=0$$

Recall that the computational space has been implemented in such a way that each node in the mesh has a unique computational space. Up to this point, the computational space for a given node, which is comprised of all of the connected neighboring nodes, has been constructed by placing each of the connected nodes on a unit circle such that both angles and edge-lengths are all equal for a given computational space control volume. If the Winslow equations are to be utilized on a mesh that has been generated to analyze a viscous flow, the treatment of the computational space will need to be modified. This equal angle, equal edge-length computational space is illustrated in Figure 67, which shows a node (node 8) in physical and computational space, respectively.

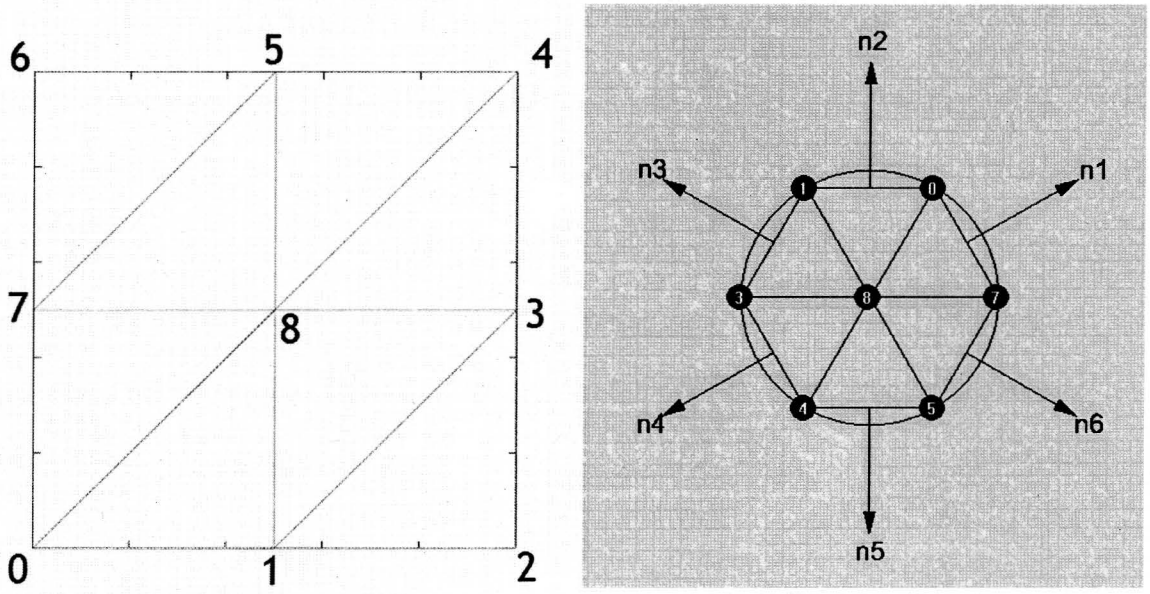


Figure 67 – Interior node (node 8) in physical and computational space.

By implementing the computational space in the way shown in Figure 67 (i.e., using equal angles and equal edge-lengths), the Winslow equations smooth the physical mesh in such a way that the modified (i.e., smoothed) mesh is ideally suited for an inviscid flow. The modified mesh will be comprised, to the greatest extent possible given the overall system, of near-equilateral triangles, which is what would be desired for an inviscid flow analysis. However, equilateral mesh elements are not ideally suited for the analysis of a viscous flow. The nature of a viscous mesh near a no-slip wall is in direct opposition to the goals of the unmodified Winslow equations utilizing equal angles and equal edge-lengths in computational space.

Consider the flat plat mesh that has been examined previously. This mesh, before and after smoothing has been performed, is shown on Figure 68. A cursory examination of Figure 68 shows that the spacing seems to be retained when smoothing is performed and the observer may infer that perhaps a viscous grid will also retain some of the tight

surface mesh spacing necessary to capture a viscous boundary layer. However, a closer view of the mesh near the flat-plate surface, such as on Figure 69 and Figure 70, shows that the spacing of the smoothed mesh is not actually determined by the original off-wall (i.e., normal) spacing of the mesh. Rather, the spacing near the surface remains tight because the surface nodes, which are not moving, are relatively close together so the nodes just off the surface remain close together because the elements containing them are being driven to an equilateral shape.

When a mesh actually has a near-body spacing that would be characteristic of a mesh designed to capture a viscous flow, the fact that the spacing near the wall is not being retained is much more pronounced. This is illustrated in Figure 71 through Figure 74.

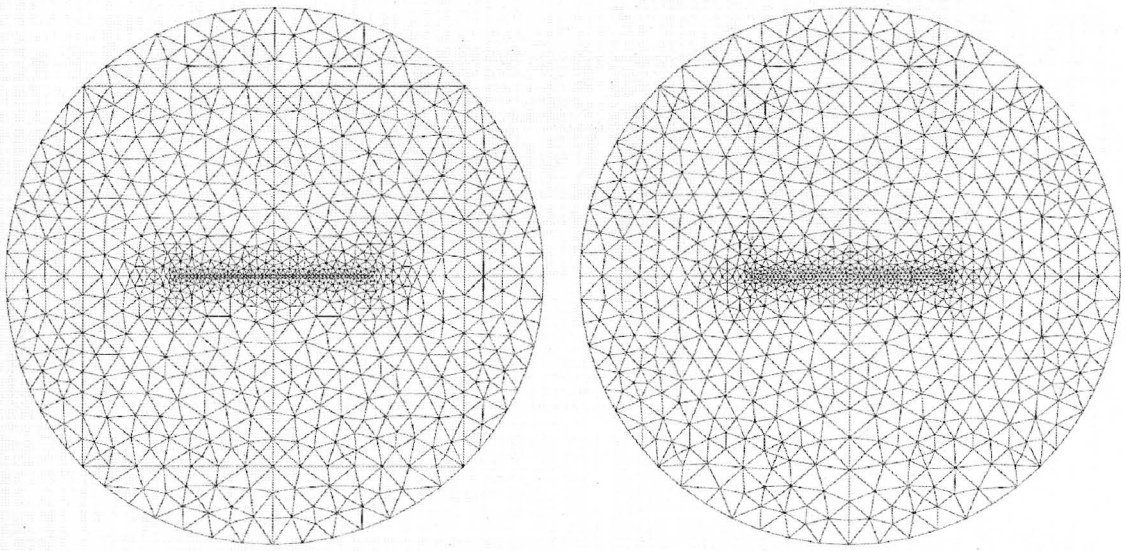


Figure 68 – Flat plate mesh, with inviscid spacing near the solid surface, before and after smoothing has been performed.

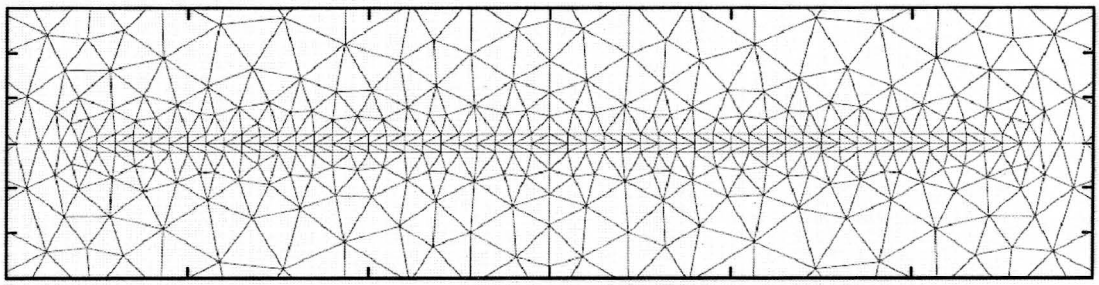


Figure 69 – Close-up of the flat plate surface before smoothing has been performed.

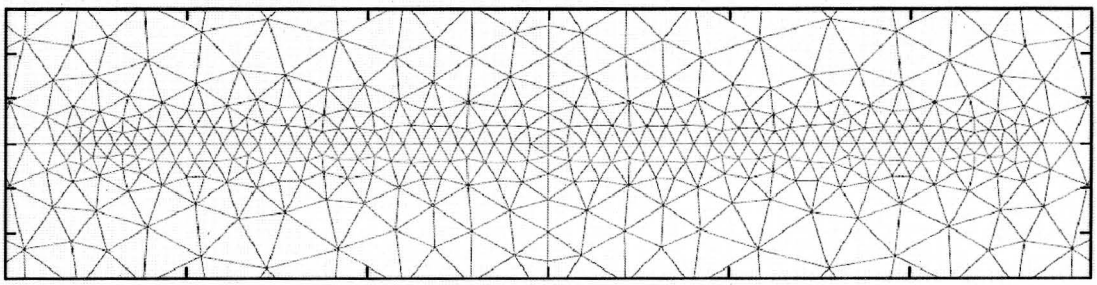


Figure 70 – Close-up of the flat plate surface after smoothing has been performed.

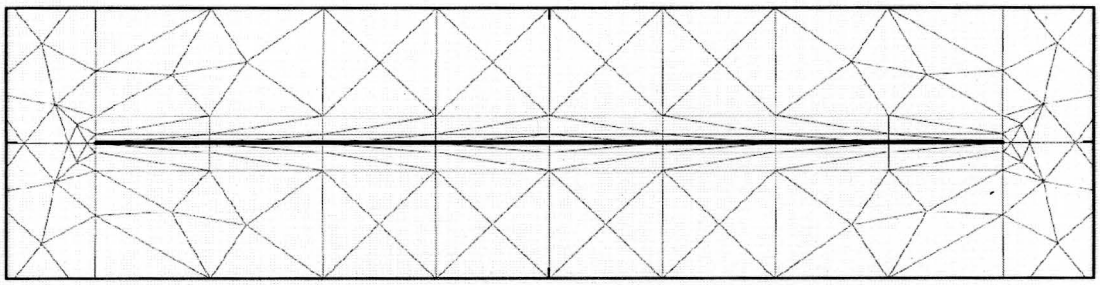


Figure 71 – Viscous mesh near a flat plate surface before smoothing.

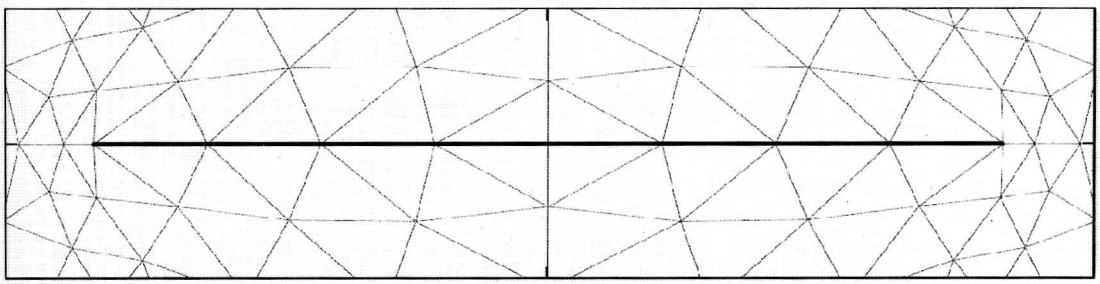


Figure 72 – Viscous mesh near a flat plate surface after smoothing.

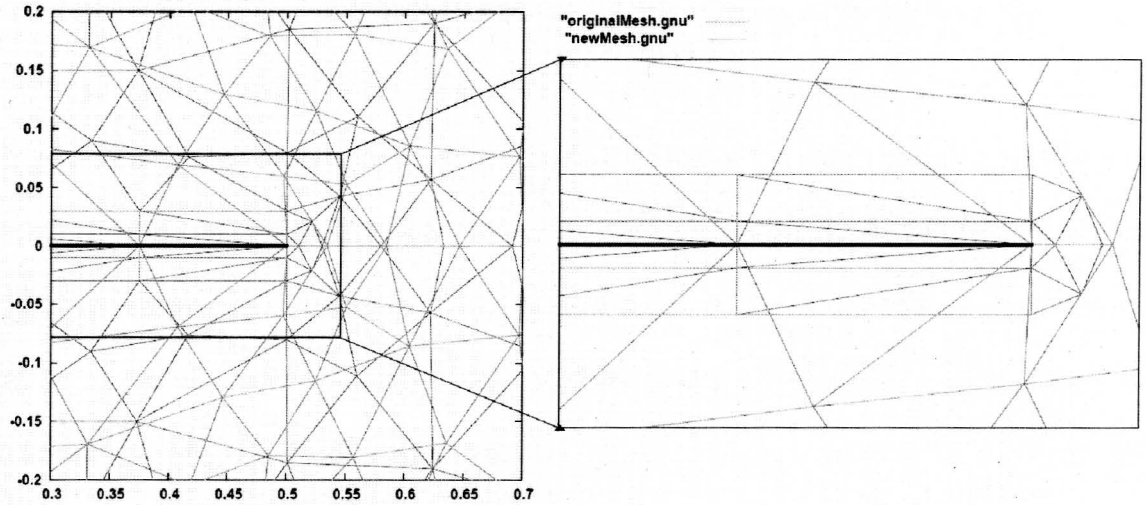


Figure 73 – Close-up of flat plate near end before (red) and after (green) smoothing.

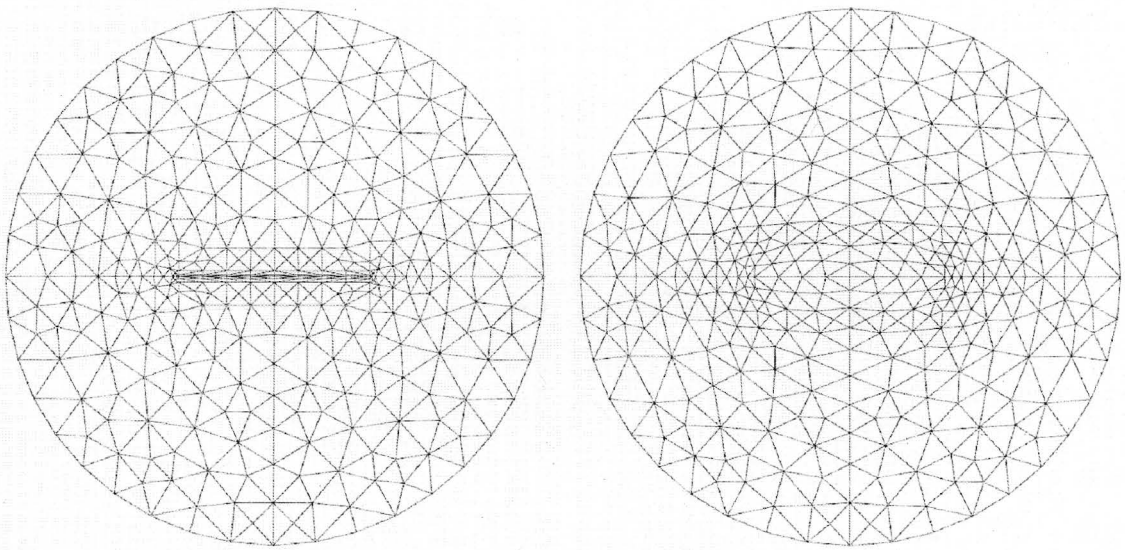


Figure 74 – Viscous mesh before and after smoothing.

As shown in Figure 74, the mesh, after smoothing has been performed, is now a viable mesh but is completely incapable of capturing a viscous boundary layer. An additional problem can also arise if the elliptic smoothing equations are used without any knowledge of the original mesh and it is just assumed that the final mesh should be optimized to drive all of the elements to an equilateral shape; this is illustrated in Figure

75. Figure 75 shows what happens if one of the viscous elements near the edges is reversed (which might arise depending on how a marching algorithm has been implemented to create viscous layers). In this scenario, the mesh points directly above and below the end of the flat plate (shown as black circles on Figure 75) are driven away from the surface and – because there are now three inward nodes trying to pull the end nodes toward the center of the flat plate – the end nodes move inward away from the end. The node shown as a black square in Figure 75 has nowhere to go because the symmetric nature of the grid prevents the node from moving up or down and thus skewed cells result.

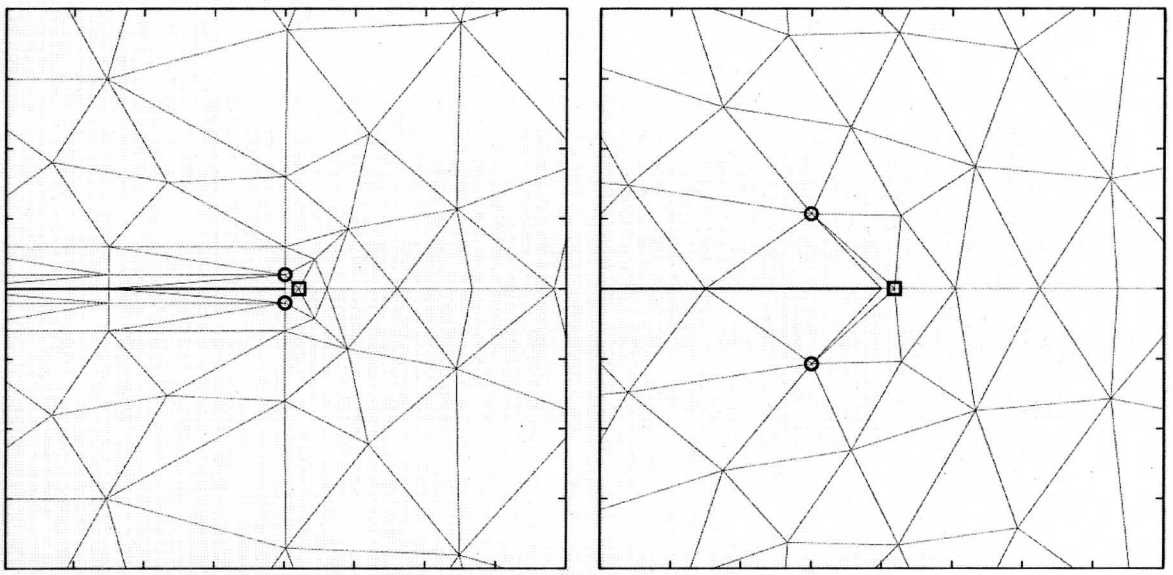


Figure 75 – Skewed elements resulting from smoothing a viscous mesh.

5.2 – Rigidly Deformed Viscous Region

One approach that was explored was to rigidly deform the viscous layers using direct coordinate mapping. This is a technique that is used in the Kestrel software [14] [15] and, although the technique lacks some of the elegance of applying a smoothing equation to

the entire mesh, in many cases it may be the most pragmatic approach. When applying this technique, each interior node in the viscous region was associated with a boundary node on a no-slip viscous surface. The associated boundary node was then used to drive the mapping (i.e. coordinate translation and rotation) for the node in the viscous region.

The assumption was made that, for a node in the viscous region, the shortest edge-length attaching a node to its neighbors is the edge that would lead to the no-slop surface. While this is just an assumption, it is quite reasonable given the nature of a viscous mesh. Standard geometric progression will ensure that the spacing increases with distance from the no-slip surface and the high aspect ratio of the viscous-region cells will ensure that the neighbors in the longitudinal direction will be connected via edges that are longer than the edges leading to the surface. If this high aspect ratio is not the case, the mesh was probably not designed to capture a viscous boundary layer in the first place and it may be logical to revert to the standard isotropic Winslow equations.

For each node, the connectivity will be known. A recursive algorithm is used to walk around the connected edges and determine the closest connected neighbor. This neighboring node is then tested to determine if it is a boundary node. If it is not, it becomes the new point about which the algorithm explores until its closest neighbor is found. This continues until a point is reached which is on a viscous boundary surface. Once the surface node is identified, it is used to drive the coordinate mapping. As seen in Figure 76 and Figure 77, this technique does a good job of retaining the original characteristics of the mesh near the surface as the surface is deformed.

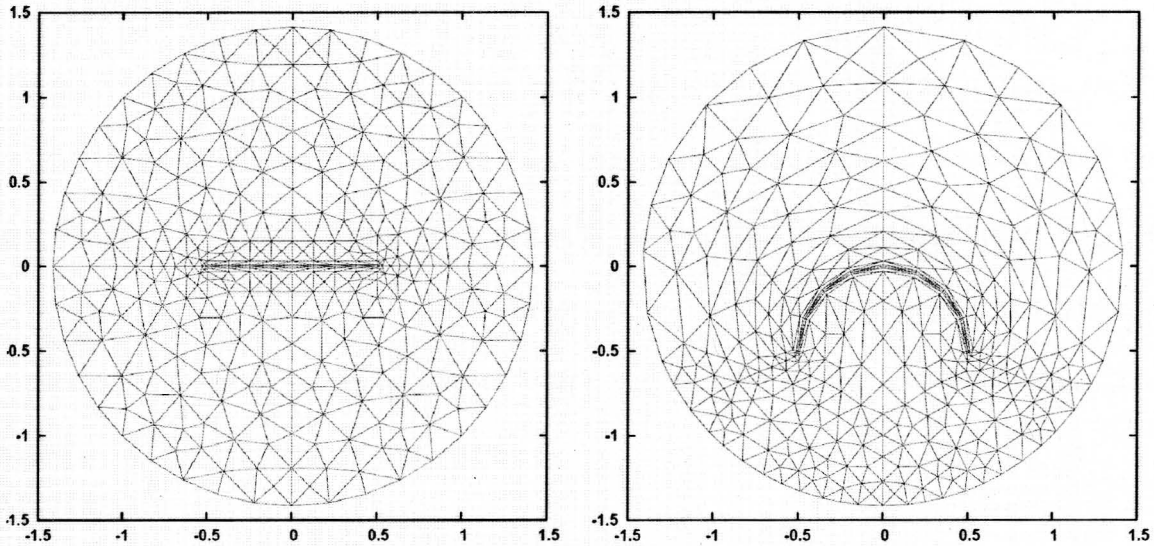


Figure 76 – Mesh movement using direct coordinate mapping.

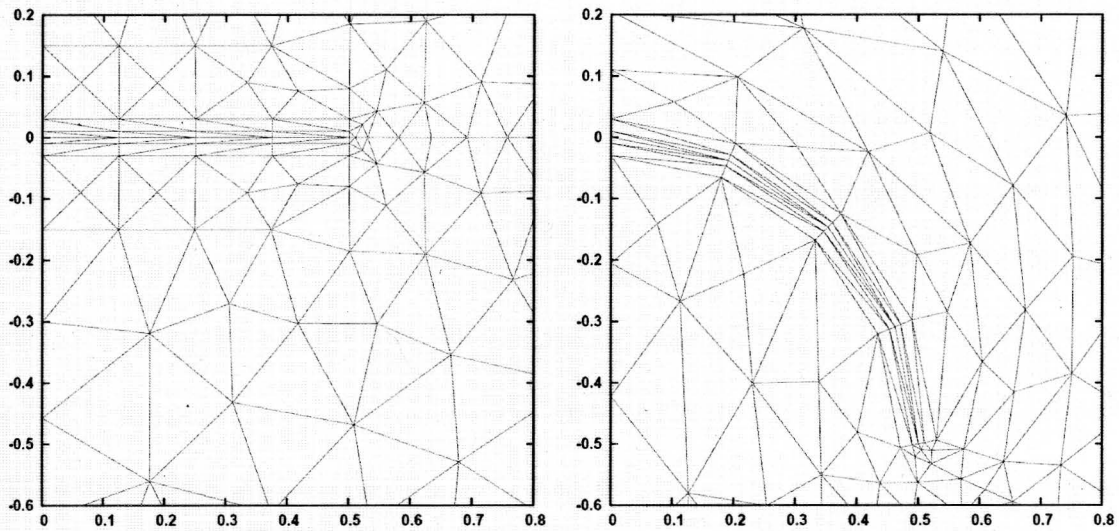


Figure 77 – Mesh movement using direct coordinate mapping shown near the viscous surface.

The rigid deformation has effectively moved the surface that is used as the boundary for the Winslow smoothing from the geometry surface to the edge of the viscous region of the mesh. While this may be the most pragmatic solution, it does not address the question as to whether or not the Winslow elliptic smoothing equations can be used to efficiently modify a viscous region. Thus, a more elegant solution is sought where the computational space is manipulated in such a way that smoothing the entire grid – both viscous and inviscid regions – is possible.

5.3 – Hybrid Mesh

One of the paramount benefits of employing the Winslow elliptic smoothing equations is the ability of the equations to conform the mesh around a surface that that has been deformed in some way. This gives the equations the potential to have powerful applications to moving bodies and aeroelasticity. Consider the case where the flat-plate surface shown on the left of Figure 74 is rotated 45 degrees. If the Winslow equations are applied to the mesh using the equal angle, equal edge-length (isotropic) virtual control volume method for constructing the computational space, the mesh will indeed conform to the new position of the inner flat-plate surface but any meaningful viscous spacing will be lost. This is shown in Figure 78 below.

Examining the way that the isotropic computational space affects the mesh in the viscous boundary layer, it becomes apparent that a better method is required. One method that will allow the Winslow equations to smooth the mesh without destroying the viscous spacing is to use the original unmodified mesh to generate the control volumes in

computational space for each of the viscous nodes rather than using the virtual control volumes constructed on a unit circle. This computational space based on the original mesh will be referred to as an anisotropic control volume in computational space because the properties will no longer be independent of orientation.

Using the original mesh has some drawbacks because there is little flexibility in manipulating the computational space. If the original mesh is of high quality, the computational space will be of high quality. If the original mesh is of low quality, the computational space will be of low quality. Furthermore, if the original mesh is of low quality, the final smoothed physical mesh will almost certainly be of low quality but a high quality original mesh does not guarantee a high quality modified (smoothed) physical mesh. In all the examples used in this section, the original mesh was a high quality mesh but it will be demonstrated that difficulties, nonetheless, arise due to the inflexibility of using the original mesh as the computational space.

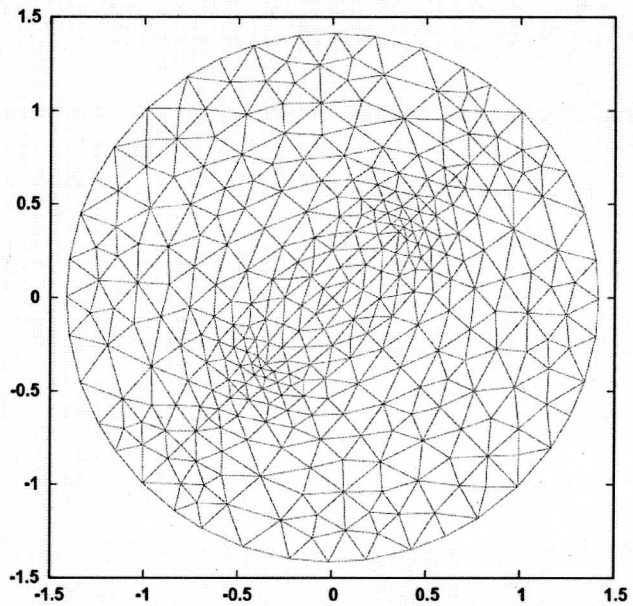


Figure 78 – Flat plate rotated and smoothed using equal angle equal edge-length (isotropic) virtual control volumes.

5.3.1 – Comparison of Hybrid Methodologies

It has been shown (such as on Figure 78) that using isotropic virtual control volumes as the computational space for all internal nodes causes any viscous spacing characteristics of the physical mesh to be lost. A possible mechanism for alleviating this loss of viscous spacing is to pursue a hybrid approach where the nodes in the viscous region are treated differently from the nodes in the inviscid region. For nodes in the inviscid region, where isotropic properties are desired, the conventional isotropic (equal angle equal edge-length) virtual control volumes will be used. For nodes in the inviscid region, where anisotropic properties are desired, anisotropic control volumes in computational space can be constructed from the original unmodified physical mesh.

An investigation was performed that compared the effect of utilizing the hybrid mechanism described above vs. simply using the original unmodified physical space for all interior nodes (which would have the effect of utilizing anisotropic control volumes for all interior nodes). In both cases (described in greater detail as Method 1 and Method 2 below) the outer boundary utilizes isotropic virtual control volumes. This is required so that the ghost node methodology described in Chapter 4 could be employed to close the virtual control volume computational space stencils and the outer boundary points could be allowed to float. The two methods are summarized below and a comparison of the results is shown in Figure 79 and Figure 80. The most noticeable difference is near the end of the flat plate when it had been translated (shown on Figure 79) and when the flat plate was warped into the shape of a semi-circle (shown on Figure 80).

Method 1 – All Interior Nodes Based on the Original Physical Mesh.

For this implementation, all of the interior nodes, both viscous and inviscid, have a computational space based on the original undeformed physical mesh. This means that all interior nodes will be treated as anisotropic and no interior nodes will be smoothed using the conventional isotropic virtual control volumes. The only nodes that use a computational space based on an isotropic (equal angle, equal edge-length) virtual control volume are the nodes on the outer boundary, which are allowed to float. The results from this method are shown on the left hand side of Figure 79 and Figure 80.

Method 2 – Hybrid Computational Space.

For this implementation, the nodes in the viscous region use a computational space based on the original undeformed physical mesh (an anisotropic computational space) while the nodes in the inviscid region (as well as the outer boundary nodes) use an isotropic computational space. The results from this method are shown on the right hand side of Figure 79 and Figure 80.

When the flat-plate surface was rotated 45 degrees, both methods generated very similar results. However, when the flat-plate surface was translated, shown on Figure 79, and warped, shown on Figure 80, the meshes generated using the two methods diverged more significantly. For the cases involving translation and warping, the hybrid method (method 2) generated better results.

The most telling case involved warping the flat-plate into a semi-circle. This case was interesting because it subjected the smoothing algorithm to the most significant stress. This case is also interesting because it is a good test case for what might be encountered when examining a problem (e.g. an aeroelastics problem) where the shape of a surface was changing in some way from its original shape. As expected, using isotropic virtual control volumes for the computational space results in a viable mesh but all of the viscous boundary layer spacing associated with the original mesh is lost. Figure 80, shows that when the original mesh and hybrid mesh methods are employed, the characteristics of the original spacing in the viscous region is generally preserved. The subsequent figures (Figure 81 and Figure 82), however, show that there are significant deficiencies with using these methods to smooth a significantly deformed mesh.

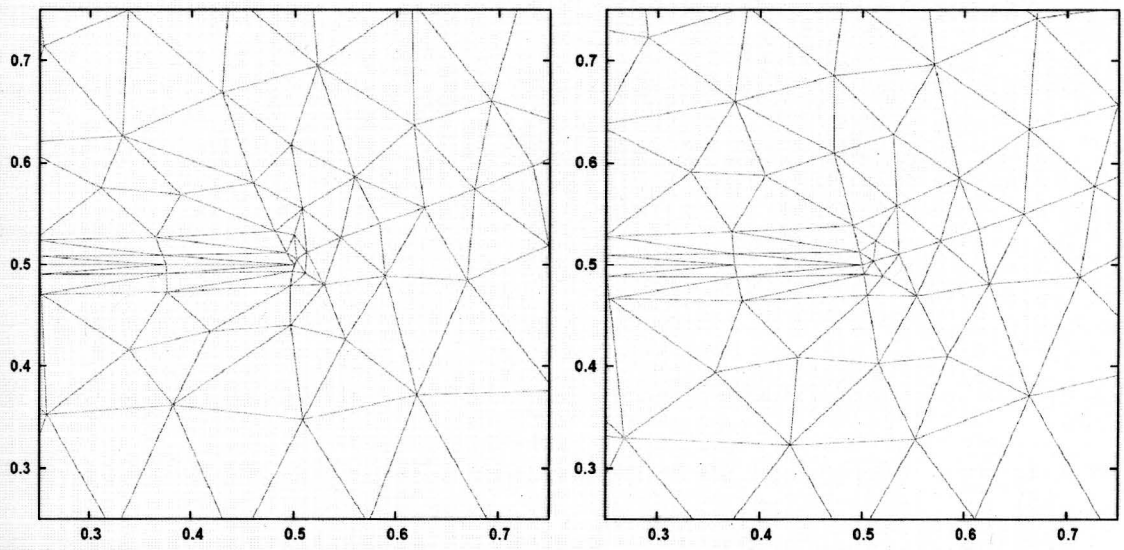


Figure 79 – Close-up near the end of the flat plate after surface has been translated and the mesh has been smoothed.

The mesh on the left uses a computational space based on the original physical mesh for all interior mesh points in the domain. The mesh on the right uses the hybrid methodology described above.

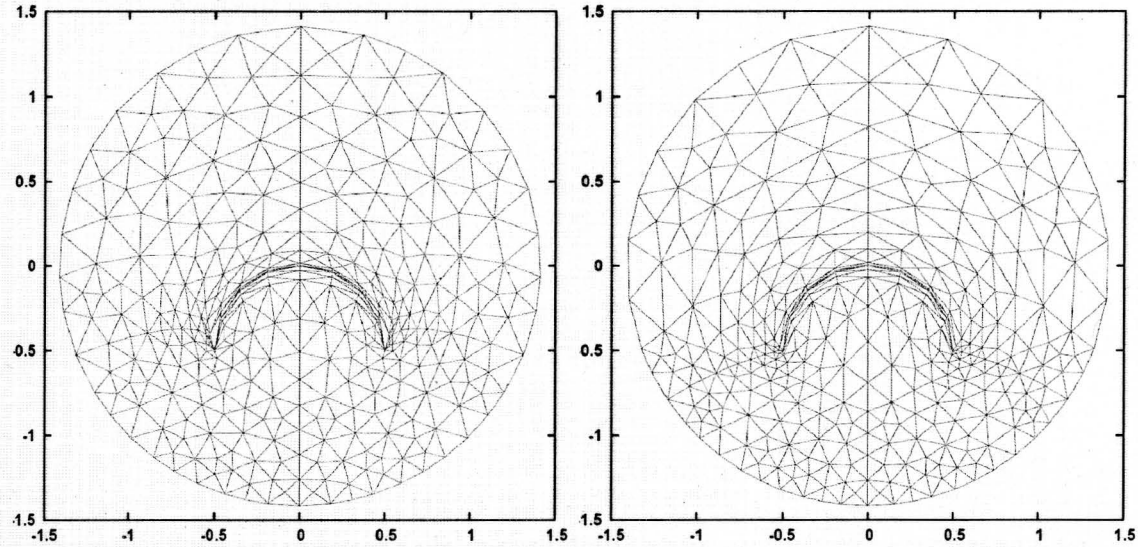


Figure 80 – Warped mesh using original anisotropic (left) and hybrid (right) virtual control volumes. The mesh on the left uses a computational space based on the original physical mesh for all interior mesh points in the domain. The mesh on the right uses the hybrid methodology described above.

The two meshes shown on Figure 80 look similar but a close-up near the end of the warped flat-plate (Figure 81) reveals that they are actually quite different. The hybrid method (shown on the right of Figure 81) results in a better mesh; the all anisotropic method, by contrast, results in a non-viable mesh with grid crossing near the end of the flat-plate.

Although the hybrid method results in a viable mesh, it is still far from what would be considered to be an ideal mesh. The ideal mesh, illustrated on the left of Figure 82, would keep all of the viscous nodes placed in such a way that the edge attaching them to the viscous surface would remain orthogonal to the surface.

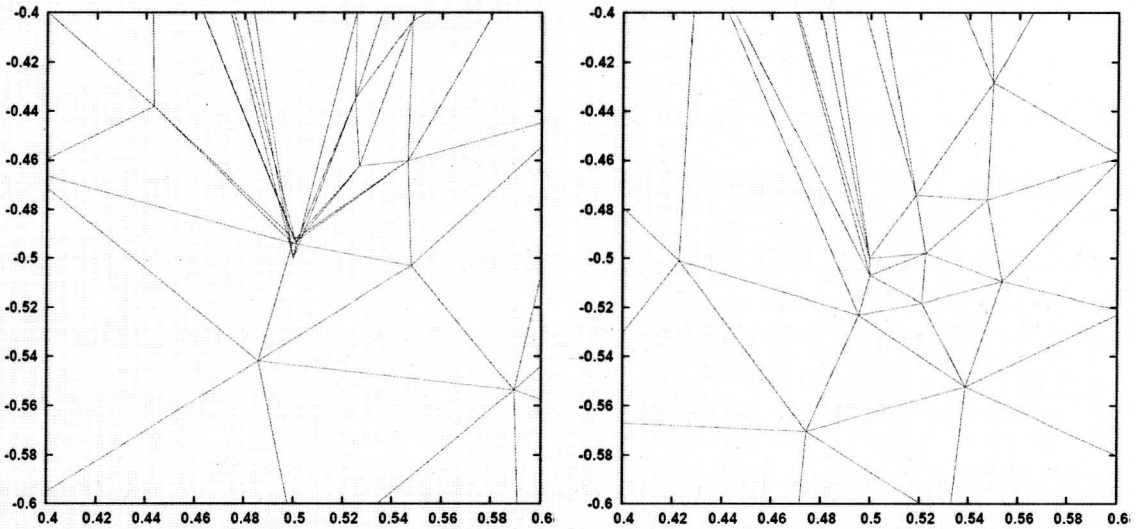


Figure 81 – Close-up of warped mesh using original anisotropic (left) and hybrid (right) virtual control volumes.

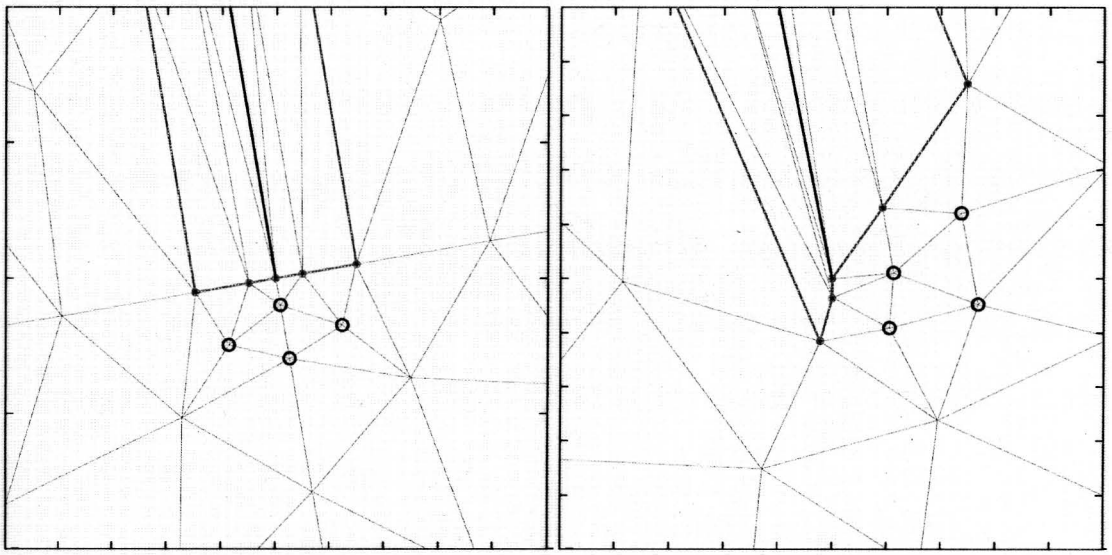


Figure 82 – Ideal mesh in viscous region vs. hybrid implementation

5.3.2 – Effect of Increasing Mesh Size/Density

The hybrid computational space was revisited using a larger and denser mesh. The intention of this exploration was to determine if adding connectivity layers between the inner and outer surfaces would have a positive effect on the final smoothed mesh. Several modifications were made to the viscous flat-plate mesh, the results of which can be seen on Figure 83 below. The first change was that the radius of the outer boundary was roughly doubled from $\sqrt{2}$ (left most mesh on Figure 83) to 3 (other two meshes on Figure 83). Second, the number of viscous layers was increased from 2 to 4. Finally, the density of the mesh was increased. The original mesh was comprised of 346 nodes.

The newer meshes had 1,236 nodes for the coarser mesh and 5,104 nodes for the finer mesh. Additional information is tabulated in Table 3. Smoothing was applied to the 1,236 node mesh (large-coarse) using the hybrid computational space where the node in the viscous region had a computational space based on the original mesh. Recall that when the original mesh was used, the viscous layers pulled away from the surface (this was shown in Figure 80). It was found that adding more nodes and more viscous layers does not alleviate this problem.

	Original	Large Coarse	Large Fine
Outer Boundary Radius	1.414	3.0	3.0
Number of Nodes	346	1236	5104
Number of Elements	636	2367	9792
Nodes on Outer Boundary	41	81	401
Number of Viscous Layers	2	4	4

Table 3 – Flat plate mesh characteristics.

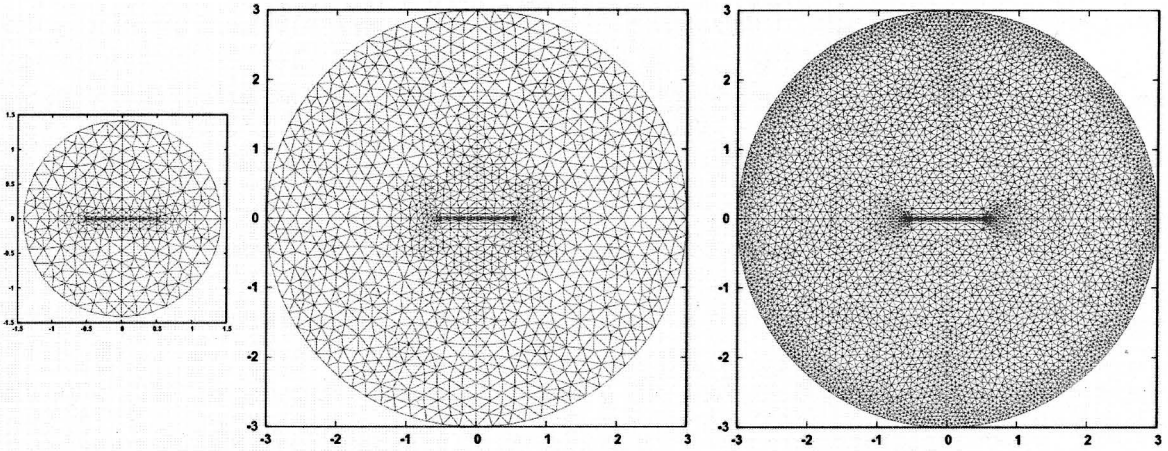


Figure 83 – Comparison of flat plate meshes of varying size and density.

An unexpected result of reexamining the viscous smoothing using a hybrid computational space was that having a larger mesh with more separation between the inner and outer surfaces (both in physical space and in layers of connectivity) did not improve the manner in which the mesh points forming the most direct connection between the flat-plate end-points and the outer boundary behaved. This is illustrated in Figure 84. Ideally, the mesh points would come off the flat-plate surface in a manner such that the mesh points would extrude outward from the inner surface in a normal manner. When the flat plate is undeformed, the mesh points that most directly connect the endpoints of the flat-plat to the outer boundary extend from the ends of the flat-plate to the outer surface in a horizontal fashion (this "direct-connection line" is shown as a horizontal line of edges on Figure 83 and is highlighted in blue on Figure 84).

When the flat-plate is warped into a semi circle, the direct-connection line should extend from the flat-plate endpoints to the outer surface in a near-vertical fashion to preserve the original characteristics of the mesh near the flat-plate ends. As conspicuously shown on Figure 84, this does not happen. When a larger mesh was used,

the hypothesis was that if there were more cells in the interior, the direct-connection lines would rotate downward more and would connect the inner and outer surfaces in a more vertical fashion. This seemed to make sense because, if more cells exist, it was assumed that the direct-connection lines would be able to move down farther without causing the inviscid mesh elements to diverge greatly away from their ideal equilateral shape. However, the larger mesh actually decreased the shift in the direct-connectivity line between the flat-plate end points and the outer boundary.

When the flat-plate embedded in the 5,104 node mesh (large-fine) was warped into a semi-circle and the mesh was smoothed, the solution mesh became highly crossed – the solution did not diverge but rather converged to an invalid (crossed) mesh. When the mesh was closely examined it was found that in the first iteration of smoothing, when the endpoint on the flat-plate was forced downward via a prescribed-motion deformation, the point just off the end point horizontally was driven upward.

In the coarser meshes, the smoothing algorithm recovered from this anomaly and the mesh points near the flat-plate surface were driven down to regain their respective positions near the surface. However for the denser mesh, the algorithm never recovered and the endpoints continued to be driven up resulting in an invalid mesh. This is illustrated in Figure 85.

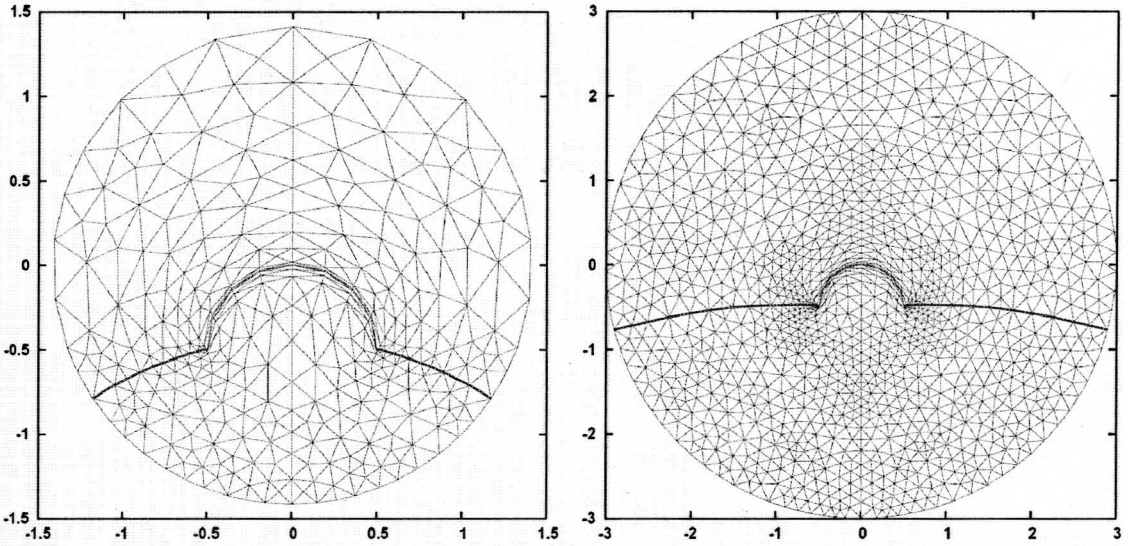


Figure 84 – Behavior of nodes directly connecting the flat plate's end points to the outer boundary.

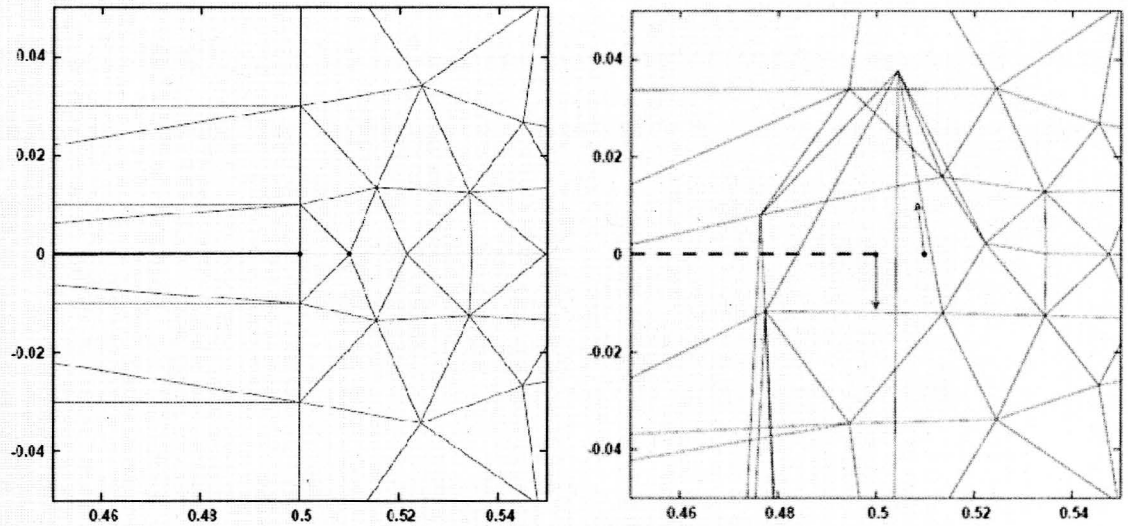


Figure 85 – Behavior of an off-body node on the first smoothing iteration.

5.4 – Computational Space Reference Frame

If a small mesh with a single interior node, such as the box5 mesh shown on Figure 86, is smoothed using the physical space for the computational space control volume, the smoothing algorithm will return the exact same mesh that it started with. This, in itself, is not a very interesting case. However, an interesting characteristic of the smoothing equations is that the computational space is reference-frame insensitive; i.e., it is unaffected by translation, scaling or rotation. The algorithm could use any of the transformed computational spaces shown on Figure 87 and the final mesh would not change from the original mesh. Because of this property, simply rotating the control volumes in order to try to affect the orientation of one of the edges (e.g. the edge between the surface and the first viscous node) will not have any effect on the final (smoothed) mesh.

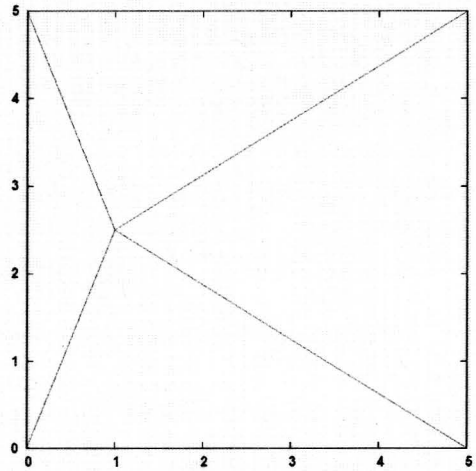


Figure 86 – Box5 mesh before and after smoothing.
The simple box5 mesh is smoothed using the original mesh as the computational space and the original mesh is returned. This is the case regardless of the manner in which the computational space is rotated, translated or scaled. Any of the computational spaces shown in Figure 87 can be used and smoothed mesh will remain unchanged.

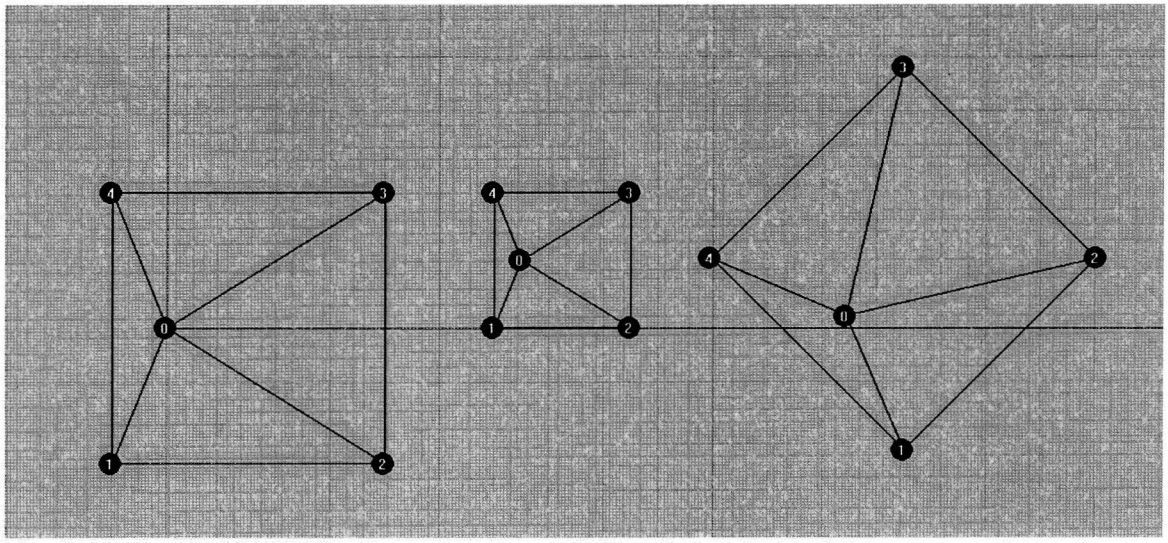


Figure 87 – Examples of computational space control volumes

5.5 – Manipulating the Computational Space

Consider Node 35, shown on Figure 88 and Figure 89. This node is in the viscous region and so it is appropriate that it have a computational space control volume based on the original mesh. This is because the original characteristics of the surrounding mesh will need to be retained as smoothing is performed. The computational space (based on the original mesh) for node 35 is shown on Figure 89.

The only information that node 35 has to determine how it is to move, comes from the control volume shown in Figure 89. Node 35 only knows what the edge lengths and internal angles are supposed to be, relative to one another. It does not know that edge 35-1 is supposed to stay the same length in order to maintain a desired off-body (y+) spacing. Node 35 also does not know that edge 35-1 is supposed to be orthogonal to the surface, which is shown as a dotted line on Figure 89. As the flat-plate surface is warped, the nodes on the surface will move farther apart (this is illustrated on Figure 90). This

causes edge 35-36 to expand, and this, in turn, drives the rest of the edges to expand as well because they are trying to maintain the same *relative* shape.

The first attempts to better maintain the viscous spacing as the surface was warped was to reduce the size of edge 35-1 such that it was the proper off-body spacing relative to the new (warped) surface-node spacing. After edge 35-1 was resized, angle 1-35-36 was then set equal to angle 4-35-1; this had the effect of increasing the magnitude of angle 35-1-4 in physical space. When these initial fixes were implemented, the grid was improved somewhat and a comparison of the mesh before and after the fix has been implemented is shown on Figure 91. This figure illustrates that node 35 now has a position closer to its original viscous spacing but the mesh is still not ideal.

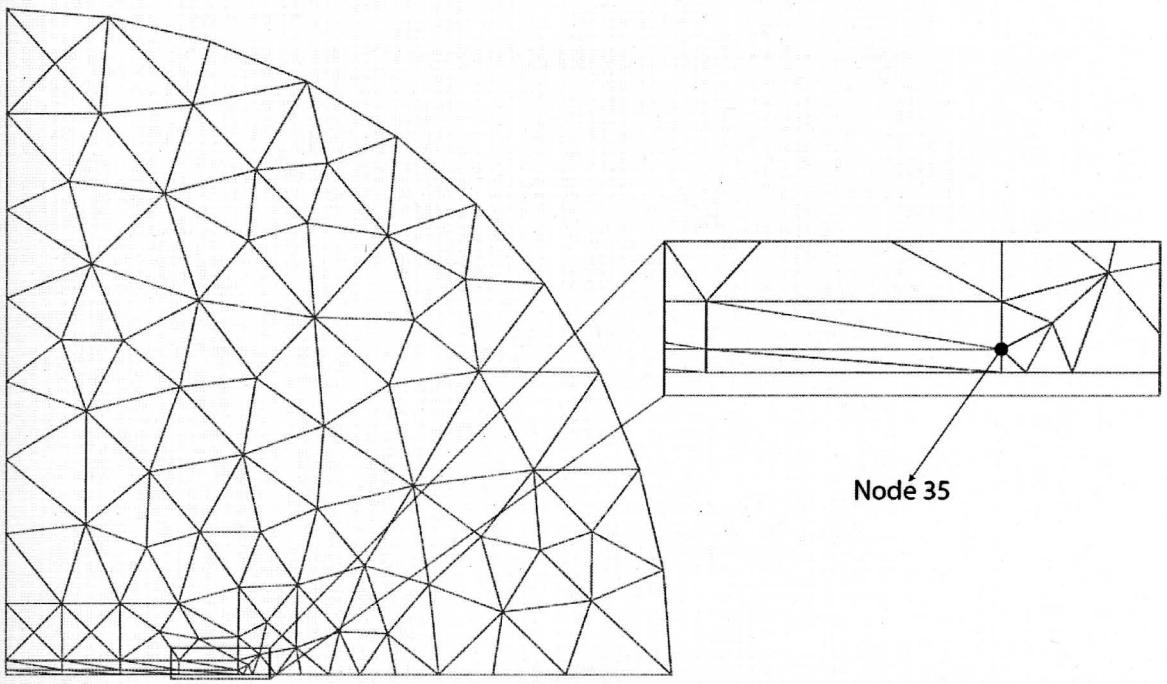


Figure 88 – Viscous node (node 35).

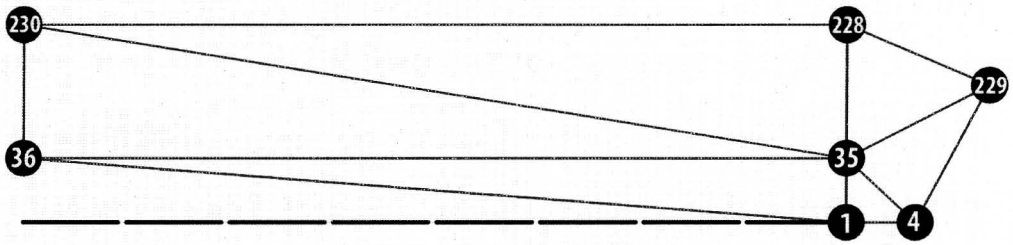


Figure 89 – Computational space for node 35.

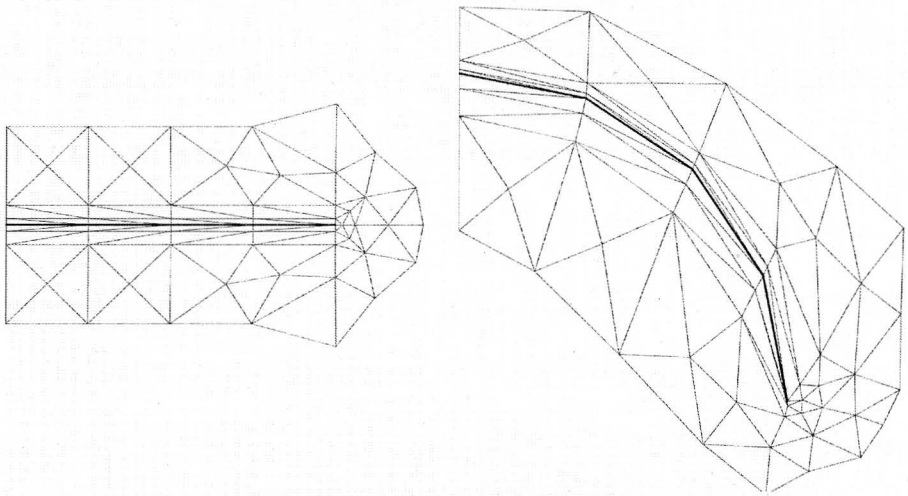


Figure 90 – Hybrid mesh responding to surface deformation.

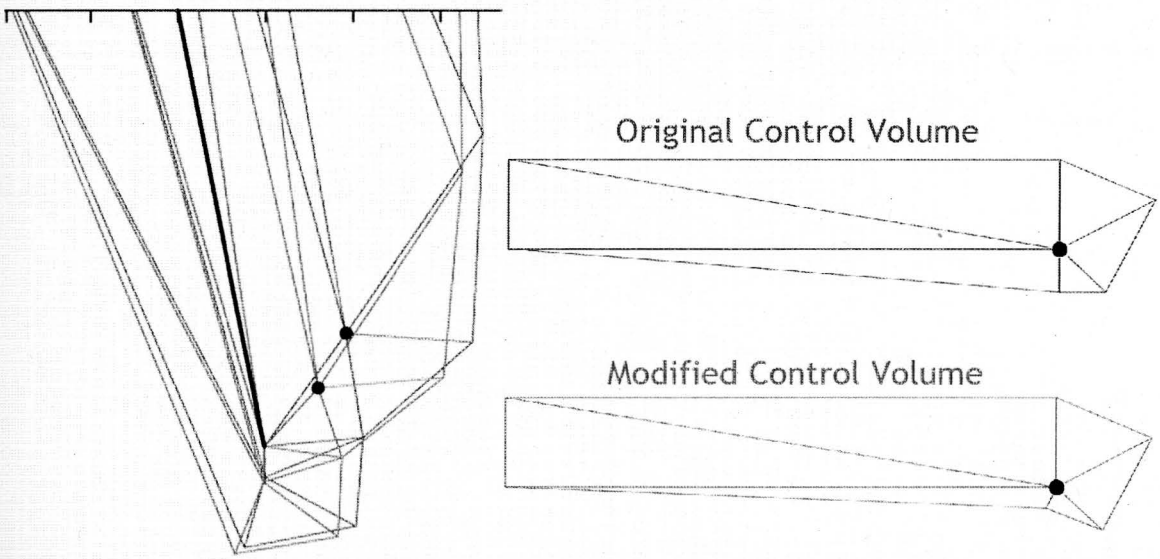


Figure 91 – Viscous mesh, near the edge of the flat plate, before and after the viscous control volume fix has been implemented.

In this chapter, several methodologies and mechanisms were explored in the attempt to preserve the viscous spacing characteristics in viscous regions of a mesh as the surfaces embedded in the mesh were transformed in some manner. Using isotropic virtual control volumes generally generates high quality meshes but, because of the isotropic nature of the computational space, the physical space looses the ability to capture viscous flow characteristics where the characteristics of the flow are highly direction-dependent.

The method that was explored in the greatest detail was a hybrid approach where the nodes in the inviscid region continued to be smoothed using an isotropic (equal angle equal edge-length) virtual control volume while the nodes in the viscous region used an anisotropic control volume based on the original undeformed physical mesh. This hybrid method had the ability to preserve some of the desired viscous spacing characteristics but, in many cases, did not result in a high quality mesh with low skewness in the cells. This method is also limiting because a high quality physical mesh, which is not always available, is required and because any flexibility with regards to changing the characteristics of the mesh (such as modifying the mesh for different Reynolds numbers) is severely diminished. Because of these limitations, other methods for smoothing viscous regions of a mesh are explored in Chapter 6 and Chapter 7.

Chapter 6 – Riemannian Metric Tensors

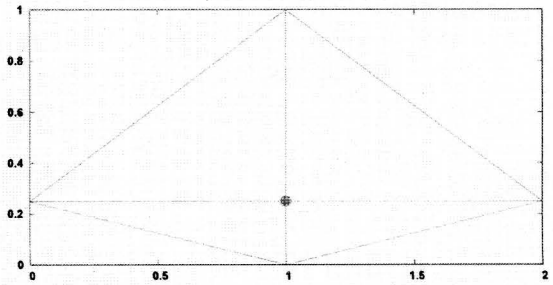
It has been demonstrated that the Winslow elliptic smoothing equations can be used to smooth an unstructured mesh by employing uncoupled virtual control volumes as the computational space for each node [5]. Because the virtual control volumes (computational space) are not affected by the physical mesh (physical space), smoothing can be performed even if the original physical mesh is of low quality or, due to mesh movement or deformation, has even become unviable (i.e., the mesh has grid crossing, negative volumes etc.). The way the Winslow smoothing methodology has conventionally been implemented is that isotropic virtual control volumes with equal angles and equal edge-lengths have been employed as the computational space. This has the effect of driving all of the triangular elements in physical space to exhibit isotropic behavior; that is, to be as close to equilateral as possible within the confines of the overarching system. This is the ideal situation for inviscid regions of a mesh but is inadequate for viscous regions. Because of the anisotropic nature of fluid flow in a viscous region, large aspect-ratio mesh elements are required. Conventional isotropic Winslow smoothing techniques do not preserve these anisotropic – high aspect-ratio – elements so a technique was investigated to manipulate the computational space using Riemannian metric tensors.

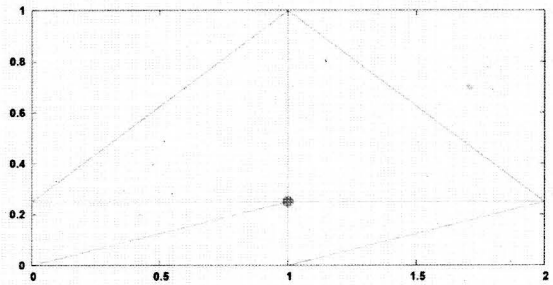
6.1 – Viscous Region Elements

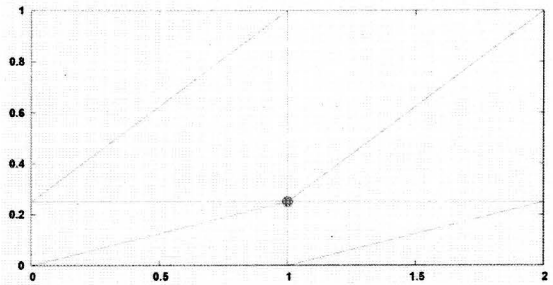
In general, a node in the viscous region will be surrounded by high-aspect-ratio elements whose characteristics are defined by the surrounding nodes. Except in unusual

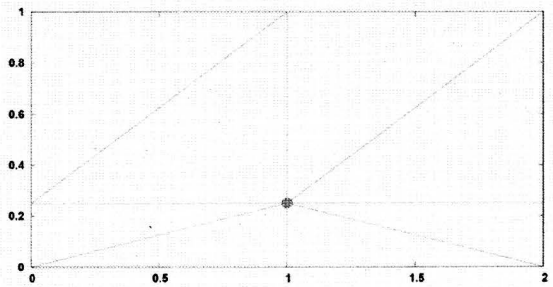
cases, a 2D node in the viscous region will have a valence count between 4 and 8. Typical mesh structures for a given node in the viscous region are shown below on Figure 92. As stated in the introduction chapter, the node-of-interest – i.e. the node being smoothed at a given instant (which is shown as green on Figure 92) will often be referred to in this document as the central node and is always located at the origin in computational space.

The two structures that will be focused on in this chapter are the valence-4 and the valence-6 structures. This is because the valence-6 case is the most common and the valence-4 is the easiest case on which to describe the viscous smoothing logic.









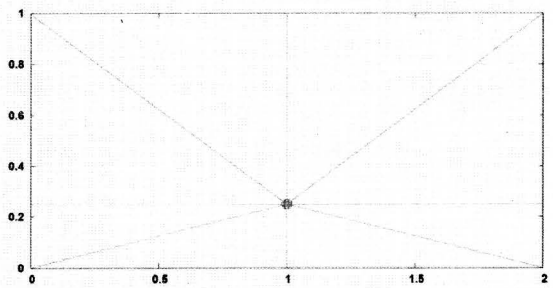


Figure 92 – Typical mesh structures surrounding a node in a viscous region.

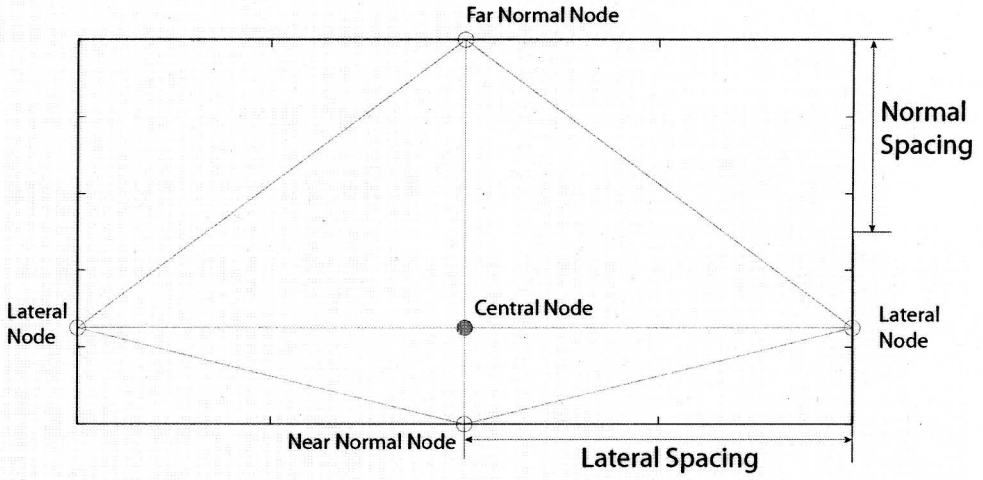


Figure 93 – Mesh structure for a valence-4 viscous node.

The structure of the mesh surrounding a node in the viscous region is determined by the normal spacing (orthogonal to the surface) and the lateral spacing (parallel to the surface) of the mesh near a viscous surface. For a given valence-4 node, there will be two edges normal to the surface and two edges parallel to the surface. The edges connecting the central node to the lateral nodes will generally be close to the same length but the edges connecting the central node to the normal nodes will generally be different lengths because the spacing usually has some sort of geometric progression that increases the cell size as the mesh moves away from a surface. When applying a Riemannian metric tensor to a virtual control volume, normal spacing will be treated as the average of the edge length between the central node and the near-normal-node and the central node and the far-normal-node. Figure 93 attempts to elucidate the terminology and spacing distances in physical space.

6.2 – Isotropic Computational Space

The isotropic computational space is defined by a unit circle where the first node is at $\xi_1=(1,0)$ and the rest of the nodes are distributed evenly around the unit circle. The isotropic computational space for a valence-4 node and a valence-6 node can be seen below on Figure 94.

Using the valence-4 node as an example, Figure 95 shows what happens if the isotropic computational space is used to smooth the mesh. The node that is being smoothed pulls away from the surface (from y=0.25 to y=0.45). This is not the desired effect in a viscous region because it will undermine the mesh's ability to capture a viscous boundary layer.

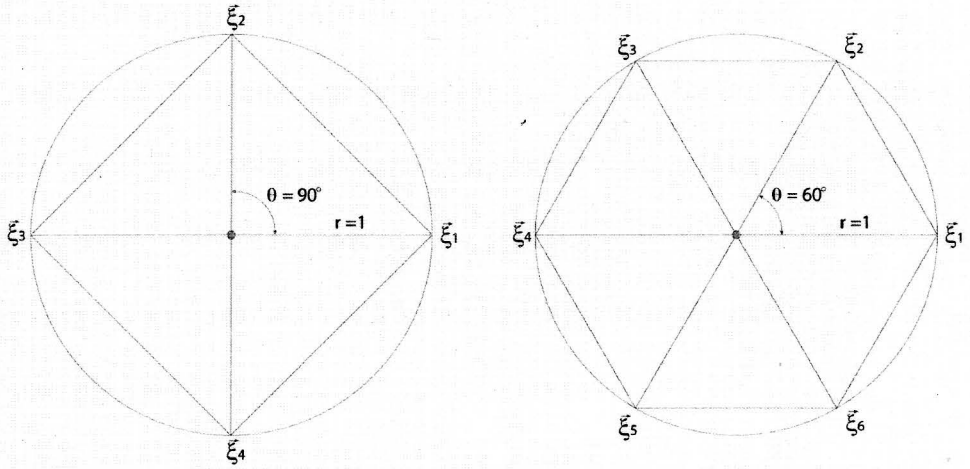


Figure 94 – Isotropic computational space for valence-4 and valence-6 nodes.

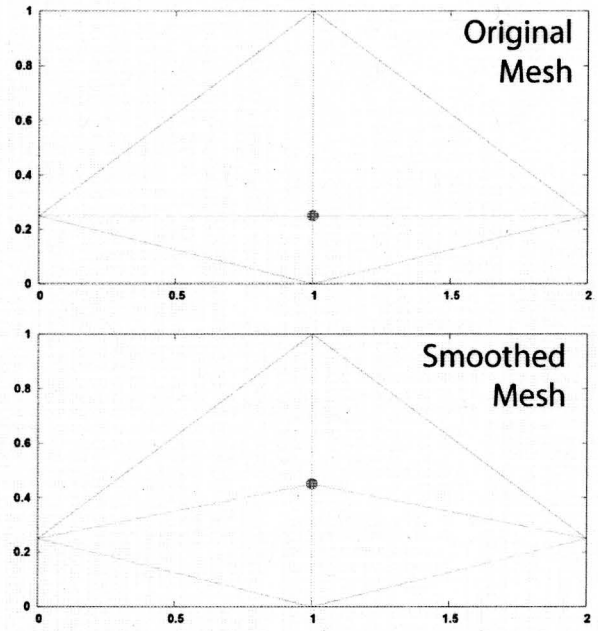


Figure 95 – Original mesh vs. smoothed mesh.

6.3 – Calculating the Riemannian Metric Tensor

To manipulate the computational space in such a way that it is more representative of the physical mesh, the concept of Riemannian metric tensors is employed. The concept of Riemannian metric tensors is based on the generation of the transformation matrix that will transform a unit circle to an ellipse (or an ellipse to a unit circle). The conventional way to write the Riemannian Metric Tensor (M) is as follows:

$$M = R\Lambda R^{-1} \tag{6.1}$$

In equation (6.1), the columns of the matrix R are the right eigenvectors of M and the diagonal matrix Λ has diagonal values which are the eigenvalues of M.

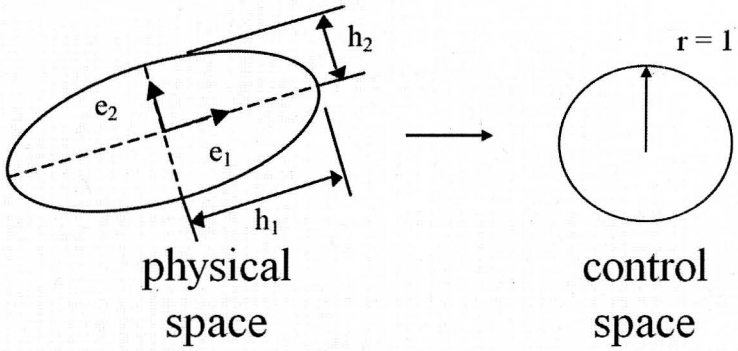


Figure 96 – Traditional interpretation of a Riemannian metric tensor.

A geometric interpretation of M is that R is a rotation matrix and Λ is a scaling matrix. The columns of R are the basis vectors for the ellipse and the scaling is traditionally defined using Figure 96 and equation (6.2).

$$M = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 \end{bmatrix} \begin{bmatrix} h_1^{-2} & 0 \\ 0 & h_2^{-2} \end{bmatrix} \begin{bmatrix} \vec{e}_1^{\,T} \\ \vec{e}_2^{\,T} \end{bmatrix} \tag{6.2}$$

There are some subtleties to the Riemannian metric tensor that make the behavior of the tensor open to misinterpretation. Defined in the way shown above, M will not transform a point on an ellipse to an equivalent point on the unit circle. Rather, using the definition for metric length shown with equation (6.3), the tensor will map a point on the ellipse to the unit length. In other words, it will take a point on the ellipse and map it to unity.

$$d_{AB} = \sqrt{\overrightarrow{AB}^T [M] \overrightarrow{AB}} \tag{6.3}$$

In equation (6.3), $\overrightarrow{AB}$ is the vector from the center of the ellipse to the ellipse surface. For an ellipse centered at zero mapped to the unit circle, this becomes:

$$\vec{x}^T M \vec{x} = 1 \tag{6.4}$$

where $\vec{x}$ is any point on the ellipse

It is important to understand the nature of the scaling (eigenvalue) matrix, Λ, and how it must be applied. If M is being used to map a vector to the scalar value of 1 (i.e., $\mathbb{R}^2 \rightarrow \mathbb{R}^1$), the axes of the ellipse are squared to compute the eigenvalues. However if the metric tensor is being used to map a point on the ellipse to an equivalent point on the unit circle, the values or the axes are not squared. The convention used herein is that $\vec{e}_1$ and h_1 will be associated with the normal spacing while $\vec{e}_2$ and h_2 will be associated with the lateral spacing. Depending on the nature of the mapping, the scaling matrix will be defined as follows:

Mapping a point on the ellipse to a unit scalar ($\mathbb{R}^2 \rightarrow \mathbb{R}^1$): $\Lambda = \begin{bmatrix} \frac{1}{h_1^2} & 0 \\ 0 & \frac{1}{h_2^2} \end{bmatrix}$

Mapping a point on the ellipse to a point on a unit circle ($\mathbb{R}^2 \rightarrow \mathbb{R}^2$): $\Lambda = \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix}$

When manipulating the computational space used for Winslow smoothing, it is generally of more practical value to map a vector-defined point in the computational space to another vector-defined point ($\mathbb{R}^2 \rightarrow \mathbb{R}^2$). Because this is the case, the non-squared values will be used to define Λ. Because of the way the virtual control volumes are constructed, the values on the unit circles are already known; what is really needed is

a transformation from the unit circle to the ellipse $\left(M^{-1}\right)$. Fortunately, M^{-1} is easy to calculate based on the matrix property $(AB)^{-1} = B^{-1}A^{-1}$.

It can be shown through elementary matrix manipulation that the inverse of M can be calculated using the inverse of Λ:

$$M = R\Lambda R^{-1}$$
$$M^{-1} = \left(R\Lambda R^{-1}\right)^{-1}$$
$$M^{-1} = \left(R^{-1}\right)^{-1}\left(R\Lambda\right)^{-1}$$
$$M^{-1} = R\Lambda^{-1}R^{-1}$$

Since Λ is a diagonal matrix, the inverse of the matrix is just the inverse of the entries. So, when mapping a point on the unit circle to a point on an ellipse, equation (6.5) will be used.

$$M^{-1} = R\Lambda^{-1}R^{-1} \tag{6.5}$$

where: $$\Lambda^{-1} = \begin{bmatrix} h_1 & 0 \\ 0 & h_2 \end{bmatrix}$$

6.3.1 – Elements of the Tensor

The tensor M can be decomposed into a rotation matrix and a scaling matrix using a single value decomposition algorithm. The rotation matrix, R, is an orthogonal matrix where the columns (which form an orthonormal set) represent the basis vectors for the orientation of the transformed space. The scaling matrix is a diagonal matrix where the diagonal elements are the eigenvalues of M. The decomposition in two and three dimensions is shown below:

$$M = R\Lambda R^{-1}$$

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

$$\text{3-D} \rightarrow M = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 & \vec{e}_3 \end{bmatrix} \begin{bmatrix} h_1^{-2} & 0 & 0 \\ 0 & h_2^{-2} & 0 \\ 0 & 0 & h_3^{-2} \end{bmatrix} \begin{bmatrix} \vec{e}_1^{\,T} \\ \vec{e}_2^{\,T} \\ \vec{e}_3^{\,T} \end{bmatrix}$$

A geometric interpretation of the metric is the transformation of an ellipse in physical space to a circle in control space (or, in 3D, the transformation of an ellipsoid in physical space to a sphere in control space) and so M can be written as $M = R\Lambda R^T$.

Note from the equations above, that $R^{-1} = R^T$. This is because R is an orthogonal matrix[1] (a matrix is defined as orthogonal if the columns form an orthonormal set, i.e. a set of orthogonal *unit* vectors) [16]. The diagonal terms of Λ correspond to the inverse of the squared target sizes h_1 and h_2 along the prescribed directions $\vec{e}_1$ and $\vec{e}_2$.

The characteristics of M can be determined from R and Λ. M must be a certain class of matrix because not all square matrices are diagonalizable (a matrix, A, is diagonalizable if there is a diagonal matrix D such that A is similar to D (i.e. $P^{-1}AP = D$)). Also, a square matrix, A, with real entries will not necessarily have real eigenvectors. However, if A is a real symmetric matrix, then the eigenvalues of A are real and, if all of the eigenvalues of A are positive then A is symmetric positive definite (i.e. $\vec{x}^T A\vec{x} > 0$ for all $\vec{x} \neq 0$).

[1] Orthogonal matrix is an unfortunate bit of terminology. "Orthonormal matrix" would clearly be a better term, but it is not standard (Poole, Linear Algebra, pg 371).

There exists an invertible matrix R and a diagonal matrix Λ such that $R^{-1}MR = \Lambda$ (or, $M = R\Lambda R^{-1}$) if and only if the columns of R are linearly independent eigenvectors of M and the diagonal entries of Λ are the eigenvalues of M corresponding to the eigenvectors in R. Further, the Spectral Theorem says that if A is an $n \times n$ real matrix, then A is symmetric if and only if it is orthogonally diagonalizable [16]. Obviously, from the definitions of R and Λ, M is orthogonally diagonalizable so M will be symmetric positive definite.

For a single triangle, there are 3 known vectors that define the edges of the triangle. Mapping each of these vectors to a unit length allows a linear system to be generated to solve for 3 unknowns. The matrix, M, has 4 entries but, because M is symmetric, $M_{12} = M_{21}$ and M can be determined by solving the linear system. For a given triangle, such as the one shown below on Figure 97, the elements of M can be found by solving system (6.6) [5]. It is important to note that the same metric tensor will be generated regardless of the way the edge vectors are defined. That is, $\vec{v}_1$, $\vec{v}_2$ and $\vec{v}_3$ can be interchanged and the same tensor will be generated.

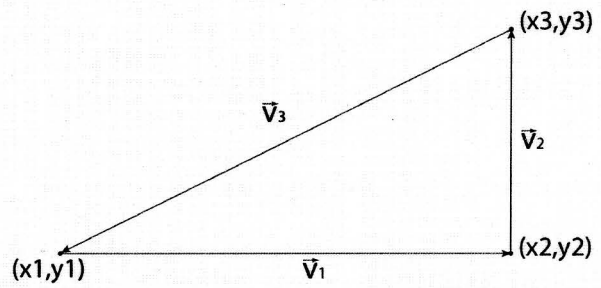


Figure 97 – Triangular mesh element used for calculating the Riemannian metric tensor.

$$\begin{bmatrix} v_{1x}^2 & 2v_{1x}v_{1y} & v_{1y}^2 \\ v_{2x}^2 & 2v_{2x}v_{2y} & v_{2y}^2 \\ v_{3x}^2 & 2v_{3x}v_{3y} & v_{3y}^2 \end{bmatrix} \begin{bmatrix} M_{11} \\ M_{12} \\ M_{22} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{6.6}$$

6.3.2 – Scaling

The computational space has conventionally been constructed on a unit circle (i.e., a circle with a radius of one). This is done for simplicity but the computational space is insensitive to scaling so a circle with a radius of any other length could be used as well. When transforming the computational space using Riemannian metric tensors, it is possible to continue to use a unit scale but this is not necessary. To keep a unit scale, the ellipse to which the computational space will be mapped will have a major axis of 1. In this case, Λ will be constructed as follows:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

where:

$$\lambda_1 = \frac{h_1}{h_2}$$

$$\lambda_2 = 1$$

This establishes the following scaling factor (C_S) between physical and computational space:

$$C_S = \frac{\lambda_1}{h_1} = \frac{\lambda_2}{h_2} \tag{6.7}$$

where: λ_1 = length of axis associated with $\vec{e}_1$

λ_2 = length of axis associated with $\vec{e}_2 = 1$

h_1 = normal spacing in physical space

h_2 = lateral spacing in physical space

Because the major axis of the ellipse is one, the scaling factor to operated on a unit scale becomes $C_S = \frac{1}{h_2}$. Again, this is not necessary because the computational space is insensitive to scaling. It may, however, make analyzing the system easier by being able to compare the computational spaces before and after transformation on the same scale. If (as in the simple example mesh shown above on Figure 95) the lateral spacing is 1, the scaling factor will be $C_S = \frac{1}{h_2} = 1$.

6.4 – Elliptic-Mapped Computational Space

6.4.1 – Origin Centered Elliptic Computational Space

One interesting observation is that when the computational space is mapped from a unit circle to an ellipse, equal angles are not maintained. This was not necessarily known a priori. It was originally thought that the transformation might map in the way shown in Figure 98-A (preserving angles) rather than Figure 98-B.

After analyzing the results from the transformation that takes place using the Riemannian metric tensor, it becomes apparent that Figure 98-B is, in fact, the realized behavior. The next (quite interesting and unintuitive) discovery was that transforming a computational space in this manner has absolutely no effect on the physical mesh.

This was a curious discovery. Consider the computational space for the central node shown in Figure 93. The aspect ratio of the normal spacing to the lateral spacing is 0.5 so the computational space before and after a Riemannian metric transformation is shown on Figure 99.

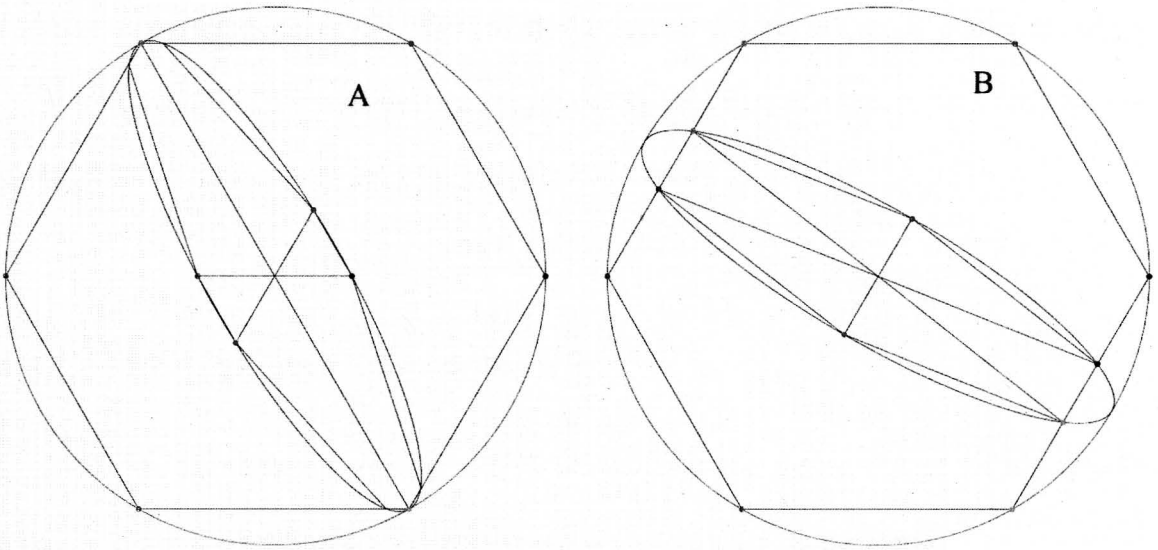


Figure 98 – Possible circle-to-ellipse transformations.

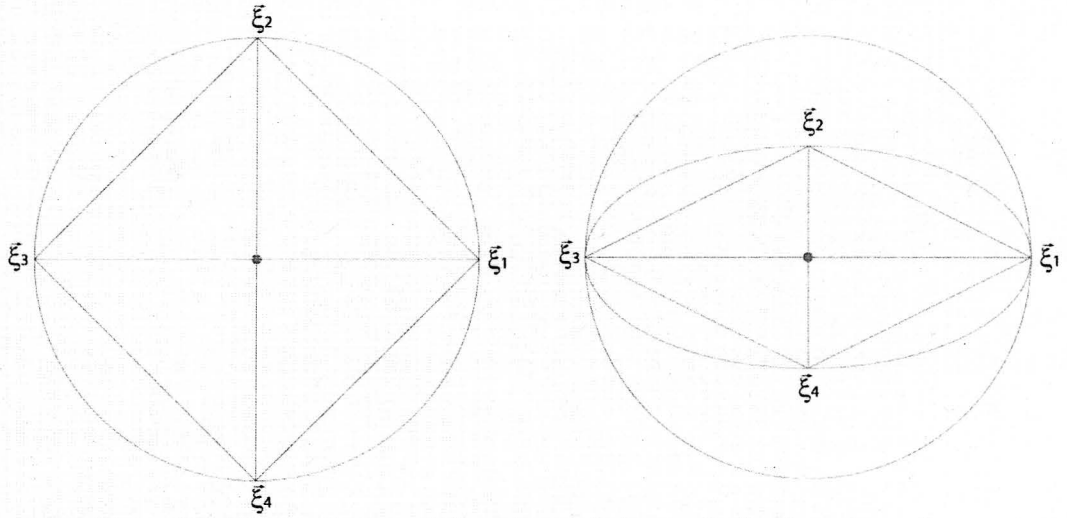


Figure 99 – Valence-4 computational space before and after transformation.

The Winslow equations with zero forcing functions, which are $\alpha\frac{\partial^2 x}{\partial\xi^2}-2\beta\frac{\partial^2 x}{\partial\eta^2}+\gamma\frac{\partial^2 x}{\partial\eta^2}=0$ and $\alpha\frac{\partial^2 y}{\partial\xi^2}-2\beta\frac{\partial^2 y}{\partial\eta^2}+\gamma\frac{\partial^2 y}{\partial\eta^2}=0$, will have different coefficients (α,β,γ) as well as different element surface normal vectors and areas; yet, when the system is solved, the differences all cancel out and the solution is unchanged. In

order to effect a change in the physical space, the computational space must not only be "squished" using the Riemannian metric tensor, it must also be offset so that the gradients of the physical coordinates are changed in computational space.

6.4.2 – Offset Elliptic Computational Space

Initially, the equation for an ellipse offset from the origin, which is a nonlinear multivariate polynomial and shown as equation (6.8), seems intimidating.

$$Ax^2 + Bxy + Cy^2 - (2Ax_0 + By_0)x - (2Cy_0 + Bx_0)y + (2Ax_0{}^2 + Bx_0y_0 + Cy_0{}^2 - 1) = 0 \quad (6.8)$$

where:

$$A = \frac{\cos^2\alpha}{a^2} + \frac{\sin^2\alpha}{b^2}$$

$$B = 2\cos\alpha\sin\alpha\left(\frac{1}{a^2} - \frac{1}{b^2}\right)$$

$$C = \frac{\sin^2\alpha}{a^2} + \frac{\cos^2\alpha}{b^2}$$

Equation (6.8) becomes more tractable, however, when manipulated into the form shown on equation (6.9).

$$A(x+x_0)^2 + B(x+x_0)(y+y_0) + C(y+y_0)^2 = 1 \quad (6.9)$$

This form now more closely resembles the equation for an ellipse which has not been translated away from the origin (in fact, it is the same equation with x_0 and y_0 set to zero):

$$Ax^2 + Bxy + Cy^2 = 1 \quad (6.10)$$

This gives insight into how the Riemannian metric tensor can be applied. Recall that the tensor is defined as $M = R\Lambda R^{-1}$ where the rotation matrix, R, has the basis vectors for

the ellipse as the columns (i.e., $R=[\vec{e}_1 \quad \vec{e}_2]$). Another way of thinking about R is as the transformation of rotation where the angle, α, rotates from the Cartesian frame to the new frame (i.e., the x coordinate is rotated to the direction of $\vec{e}_1$ and y coordinate is rotated to the direction of $\vec{e}_2$). In this case, R becomes:

$$R=\begin{bmatrix}\cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha\end{bmatrix} \tag{6.11}$$

Equation (6.4) states that a vector is mapped to a unit length using the relationship $\vec{x}^T M \vec{x}=1$. It is instructive to break this down using the rotation matrix from (6.11).

$$\vec{x}^T M \vec{x}=1$$

$$\begin{bmatrix}x \\ y\end{bmatrix}^T R\Lambda R^{-1}\begin{bmatrix}x \\ y\end{bmatrix}=1$$

$$\begin{bmatrix}x \\ y\end{bmatrix}^T\begin{bmatrix}\cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha\end{bmatrix}\begin{bmatrix}\frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2}\end{bmatrix}\begin{bmatrix}\cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha\end{bmatrix}\begin{bmatrix}x \\ y\end{bmatrix}=1$$

$$\begin{bmatrix}x\cos\alpha+y\sin\alpha \\ -x\sin\alpha+y\cos\alpha\end{bmatrix}^T\begin{bmatrix}\frac{1}{a^2} & 0 \\ 0 & \frac{1}{b^2}\end{bmatrix}\begin{bmatrix}x\cos\alpha+y\sin\alpha \\ -x\sin\alpha+y\cos\alpha\end{bmatrix}=1$$

$$\begin{bmatrix}\frac{x\cos\alpha+y\sin\alpha}{a^2} \\ \frac{-x\sin\alpha+y\cos\alpha}{b^2}\end{bmatrix}^T\begin{bmatrix}x\cos\alpha+y\sin\alpha \\ -x\sin\alpha+y\cos\alpha\end{bmatrix}=1$$

$$\left(\frac{\cos^2\alpha}{a^2}+\frac{\sin^2\alpha}{b^2}\right)x^2+\cos\alpha\sin\alpha\left(\frac{1}{a^2}-\frac{1}{b^2}\right)xy+\left(\frac{\sin^2\alpha}{a^2}+\frac{\cos^2\alpha}{b^2}\right)y^2=1 \tag{6.12}$$

It can be observed that equation (6.12) is the equation for a rotated (but not offset) ellipse: $Ax^2 + Bxy + Cy^2 = 1$.

Now, knowledge of an offset ellipse can be combined with the behavior of Riemannian metric tensors and the equation for the metric length ($\vec{x}^T M \vec{x} = 1$). This coalescence of knowledge can be used to map a point on an offset ellipse to the unit length ($\mathbb{R}^2 \rightarrow \mathbb{R}^1$) or to a point on the unit circle ($\mathbb{R}^2 \rightarrow \mathbb{R}^2$). Using the computational space coordinate system where $\vec{\xi} = \begin{bmatrix} \xi \\ \eta \end{bmatrix}$ the transformation to the unit length becomes:

$$\left(\vec{\xi} - \vec{\xi}_0\right)^T M \left(\vec{\xi} - \vec{\xi}_0\right) = 1 \tag{6.13}$$

where: $$M = R \begin{bmatrix} \frac{1}{h_1^2} & 0 \\ 0 & \frac{1}{h_2^2} \end{bmatrix} R^{-1}$$

And, the equation for mapping a point on the ellipse to a point on the unit circle ($\mathbb{R}^2 \rightarrow \mathbb{R}^2$) can be modified to become:

$$M\left(\vec{\xi} - \vec{\xi}_0\right) = \vec{\xi}_C \tag{6.14}$$

where: $$M = R \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix} R^{-1}$$

In equation (6.14), $\vec{\xi}_C$ is the point on the unit circle, which is already known. What is needed is the point on the transformed computational space (i.e. the point on the ellipse, $\vec{\xi}$). This is found as:

$$M\left(\vec{\xi}-\vec{\xi}_0\right)=\vec{\xi}_c$$
$$M\vec{\xi}-M\vec{\xi}_0=\vec{\xi}_c$$
$$M\vec{\xi}=\vec{\xi}_c+M\vec{\xi}_0$$

$$\vec{\xi}=M^{-1}\left(\vec{\xi}_C+M\vec{\xi}_0\right) \tag{6.15}$$

The angle that defines the rotation matrix is determined by the position of the near-normal node in the list of neighbors. The near-normal node will not have the same position in the list for each node but will have an arbitrary position that is based on the overall node-numbering scheme and the way the triangular elements are looped over in the smoothing code to create the neighbor list. The process for creating the offset anisotropic computational space is detailed in the following section.

6.5 – Valence-4 and Valence-6 Node Applications

It is instructive to follow the transformation of computational space through the entire process that transforms it from an equal angle, equal edge-length isotropic virtual control volume to a transformed anisotropic control volume appropriate for generating anisotropic mesh elements in physical space. To investigate this transformation, the valence-4 and valence-6 node cases will be considered.

The manipulation of the computational spaces for these two cases (valence-4 and valecne-6) is carried out below where all figures and equations specifically relating to the valence-4 case are on the left and all figures and equations specifically relating to the valence-6 case are on the right. Figure 100 shows the physical space for the two cases and Figure 101 shows the initial unmodified isotropic computational spaces. Also on

$$M\left(\vec{\xi}-\vec{\xi}_0\right)=\vec{\xi}_c$$
$$M\vec{\xi}-M\vec{\xi}_0=\vec{\xi}_c$$
$$M\vec{\xi}=\vec{\xi}_c+M\vec{\xi}_0$$

$$\vec{\xi}=M^{-1}\left(\vec{\xi}_C+M\vec{\xi}_0\right) \tag{6.15}$$

The angle that defines the rotation matrix is determined by the position of the near-normal node in the list of neighbors. The near-normal node will not have the same position in the list for each node but will have an arbitrary position that is based on the overall node-numbering scheme and the way the triangular elements are looped over in the smoothing code to create the neighbor list. The process for creating the offset anisotropic computational space is detailed in the following section.

6.5 – Valence-4 and Valence-6 Node Applications

It is instructive to follow the transformation of computational space through the entire process that transforms it from an equal angle, equal edge-length isotropic virtual control volume to a transformed anisotropic control volume appropriate for generating anisotropic mesh elements in physical space. To investigate this transformation, the valence-4 and valence-6 node cases will be considered.

The manipulation of the computational spaces for these two cases (valence-4 and valecne-6) is carried out below where all figures and equations specifically relating to the valence-4 case are on the left and all figures and equations specifically relating to the valence-6 case are on the right. Figure 100 shows the physical space for the two cases and Figure 101 shows the initial unmodified isotropic computational spaces. Also on

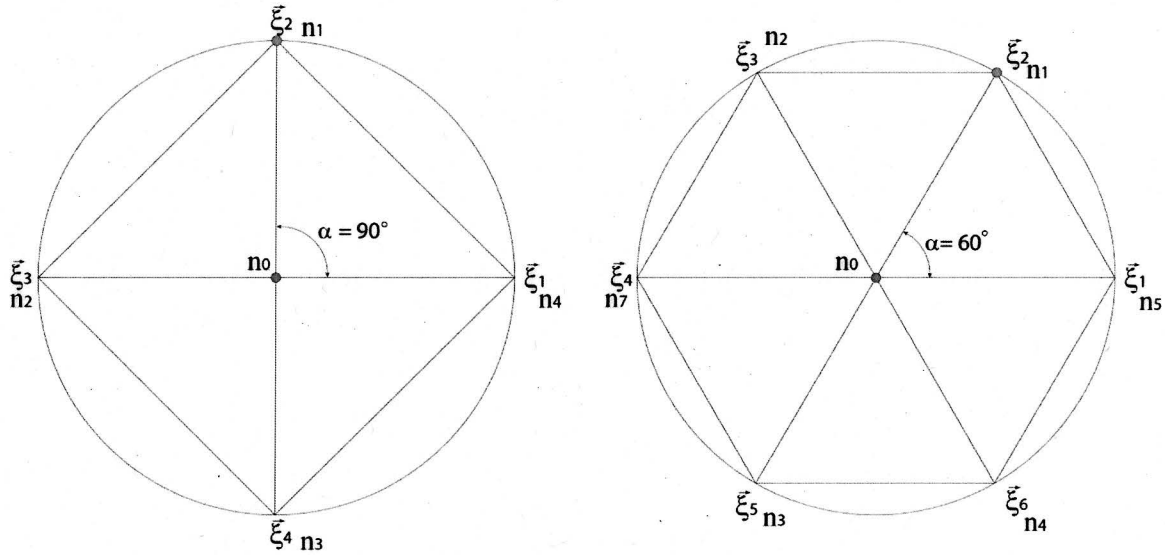


Figure 101 – Isotropic Computational space.

The angle that defines the direction to the near-normal neighbor node in computational space (α in Figure 101), along with the scaling necessary to capture the normal and lateral characteristics of the physical space mesh, are plugged into equation (6.16) to generate the transformation matrix for each of the two cases.

The transformation matrices that will transform the points on the unit circle in the isotropic computational space (the central node's neighboring nodes) to points on an ellipse, which is also centered at the origin, are shown below where the valence-4 case is on the left and the valence-6 case is on the right. A geometric interpretation of the transformation tensors is shown on Figure 102.

$$M = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 \end{bmatrix} \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix} \begin{bmatrix} \vec{e}_1^{\,T} \\ \vec{e}_2^{\,T} \end{bmatrix}$$

$$M = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$M = \begin{bmatrix} \vec{e}_1 & \vec{e}_2 \end{bmatrix} \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix} \begin{bmatrix} \vec{e}_1^{\,T} \\ \vec{e}_2^{\,T} \end{bmatrix}$$

$$M = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix}$$

$$M = \begin{bmatrix} 1.25 & 0.433 \\ 0.433 & 1.75 \end{bmatrix}$$

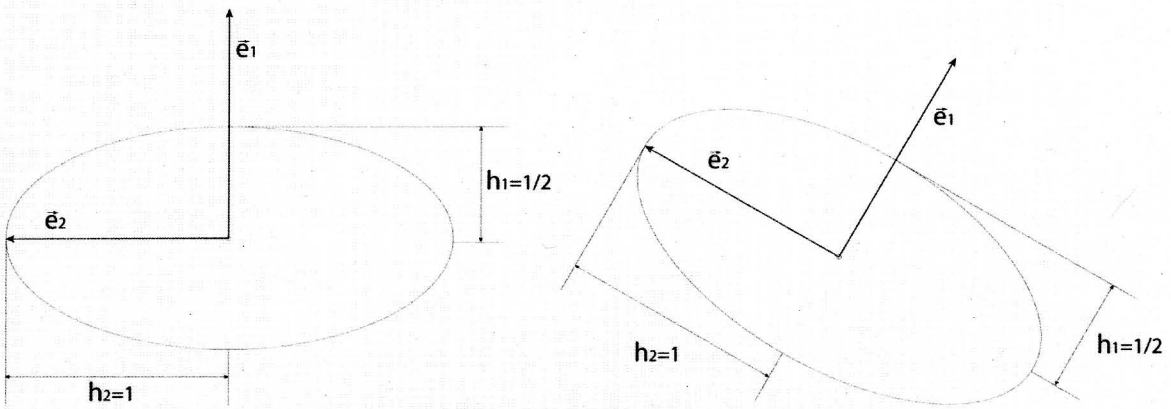


Figure 102 – Geometric interpretation of transformation tensors.

It is important to remember that M is the transformation that will transform a point on an ellipse to a point on a unit circle. However, the points on the unit circle are already known; what is actually needed are the points on the ellipse so instead of M, what will actually be used for the transformation is the inverse of M (i.e., M^{-1}). In the Transformations, which can be seen below, $\vec{\xi}_C$ is a point on the unit circle and $\vec{\xi}$ is a new transformed point on an ellipse. When these transformation relationships are applied to the original computational spaces, the results are shown on Figure 103.

$$\vec{\xi} = M^{-1}\vec{\xi}_C$$

$$\vec{\xi} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \vec{\xi}_C$$

$$\vec{\xi} = M^{-1}\vec{\xi}_C$$

$$\vec{\xi} = \begin{bmatrix} 0.875 & -0.2165 \\ -0.2165 & 0.625 \end{bmatrix} \vec{\xi}_C$$

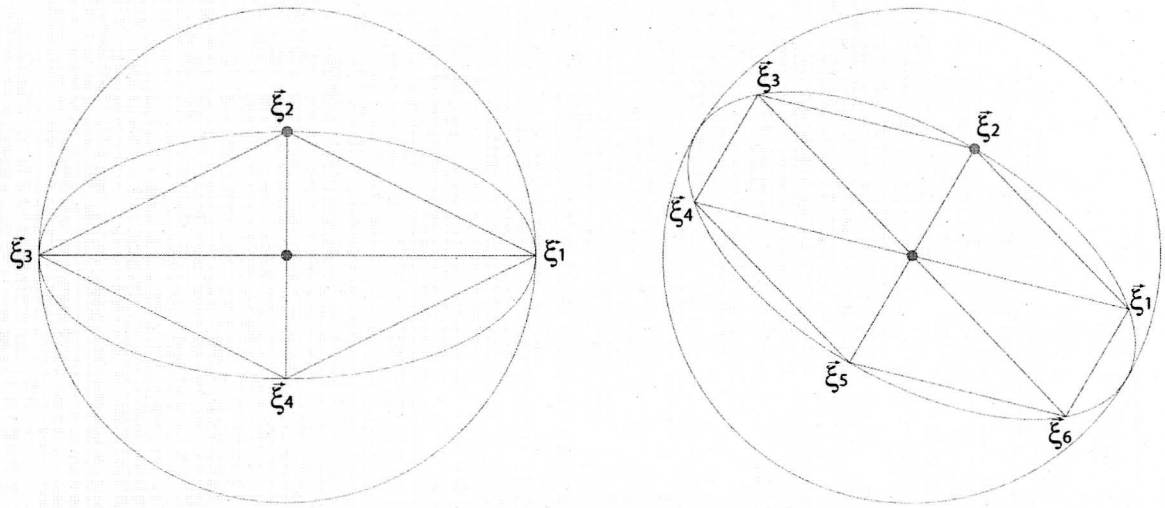


Figure 103 – Anisotropic (elliptic-mapped) origin-centered computational space.

Recall that $\vec{\xi}_0$ is the notation for the offset in computational space. This offset will not be immediately known. Instead, what will be known is the offset in physical space. The physical offset will need to be repositioned in order to generate the offset in computational space.

In computational space, the offset will be directly opposite $\vec{e}_1$ and the magnitude of the offset will be $h_1 - \|\vec{x}_0\|$ where $\vec{x}_0$ is the vector from the central node to the near-normal node. (If the computational space has been scaled to be represented by a unit ellipse, the magnitude will be $C_S\left(h_1 - \|\vec{x}_0\|\right)$.) The offset in computational space can be constructed as:

$$\vec{\xi}_0 = \begin{bmatrix} \cos(\alpha + 180) \\ \sin(\alpha + 180) \end{bmatrix} \left(h_1 - \|\vec{x}_0\|\right) \tag{6.17}$$

Returning to the valence-4 and valence-6 example cases, we get the transformation relationships shown below (valence-4 case on the left and valence-6 case on the right). The final transformed offset anisotropic computational spaces are shown on Figure 104.

$$\vec{\xi}_0 = \begin{bmatrix} \cos(90+180) \\ \sin(90+180) \end{bmatrix} (0.5-0.25) \qquad \vec{\xi}_0 = \begin{bmatrix} \cos(60+180) \\ \sin(60+180) \end{bmatrix} (0.5-0.25)$$

$$\vec{\xi}_0 = \begin{bmatrix} 0 \\ -0.25 \end{bmatrix} \qquad \vec{\xi}_0 = \begin{bmatrix} -0.125 \\ -0.125\sqrt{3} \end{bmatrix}$$

$$\vec{\xi} = M^{-1}\left(\vec{\xi}_C + M\vec{\xi}_0\right) \qquad \vec{\xi} = M^{-1}\left(\vec{\xi}_C + M\vec{\xi}_0\right)$$

$$\vec{\xi} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \left(\vec{\xi}_C + \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \vec{\xi}_0 \right) \qquad \vec{\xi} = \begin{bmatrix} 0.875 & -0.2165 \\ -0.2165 & 0.625 \end{bmatrix} \left(\vec{\xi}_C + \begin{bmatrix} 1.25 & 0.433 \\ 0.433 & 1.75 \end{bmatrix} \vec{\xi}_0 \right)$$

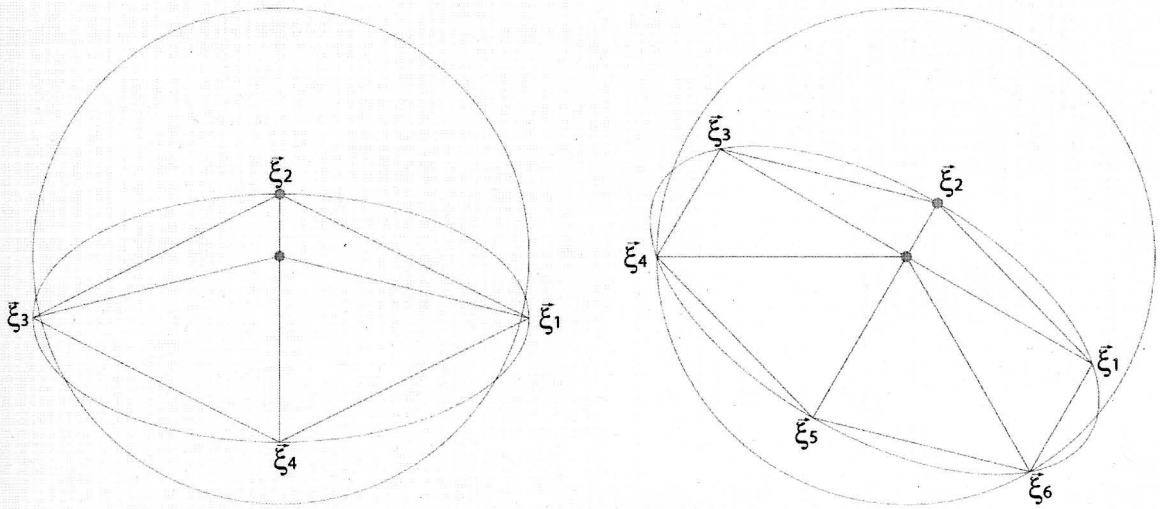


Figure 104 – Anisotropic (elliptic-mapped) offset computational space.

6.6 – Test Results

6.6.1 – Box9 Mesh

A test mesh containing 9 nodes (8 boundary nodes and one free node) was created and a valence-4 node case was examined using several different computational space setups. In the cases where an offset was utilized, the offset was calculated in accordance with equation (6.17). The original mesh is shown on Figure 105 and the results are shown on Figure 106 through Figure 109.

Although none of the results give back the exact original mesh (since none of the computational spaces perfectly captures the volumetric gradient characteristics of the physical mesh), it appears that the best results occur when the computational space is modified using the offset Riemannian metric tensor.

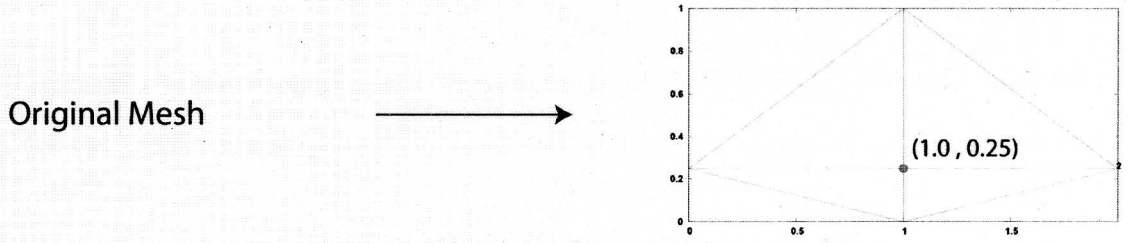


Figure 105 – Original valence-4 mesh before smoothing.

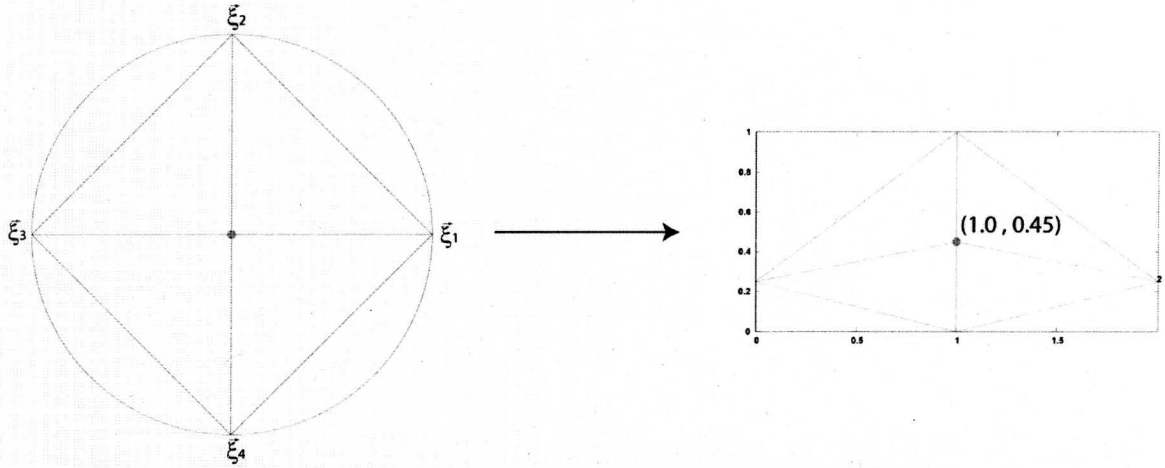


Figure 106 – Mesh smoothed using isotropic computational space.

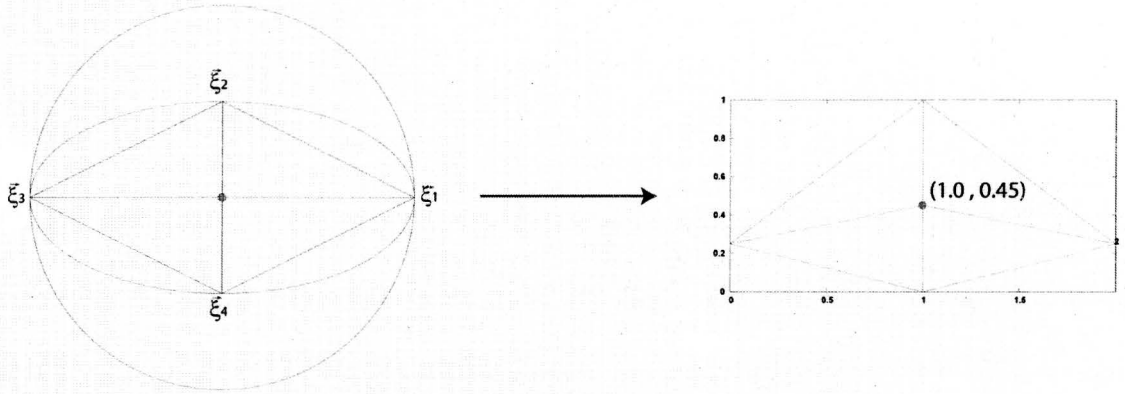


Figure 107 – Mesh smoothed using computational space modified with Riemannian metric tenor centered at the origin.

This case exhibits no change in physical space compared with the isotropic computational space case.

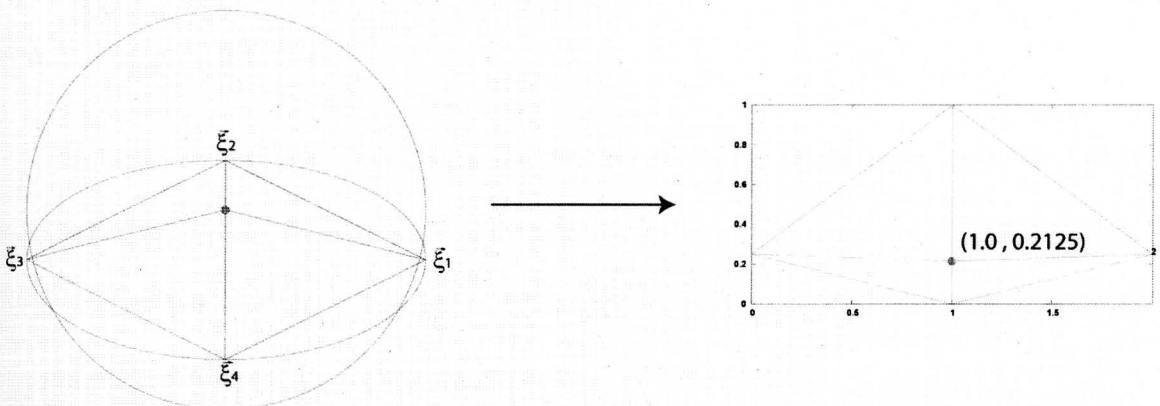


Figure 108 – Mesh smoothed using computational space modified using offset Riemannian metric tensor.

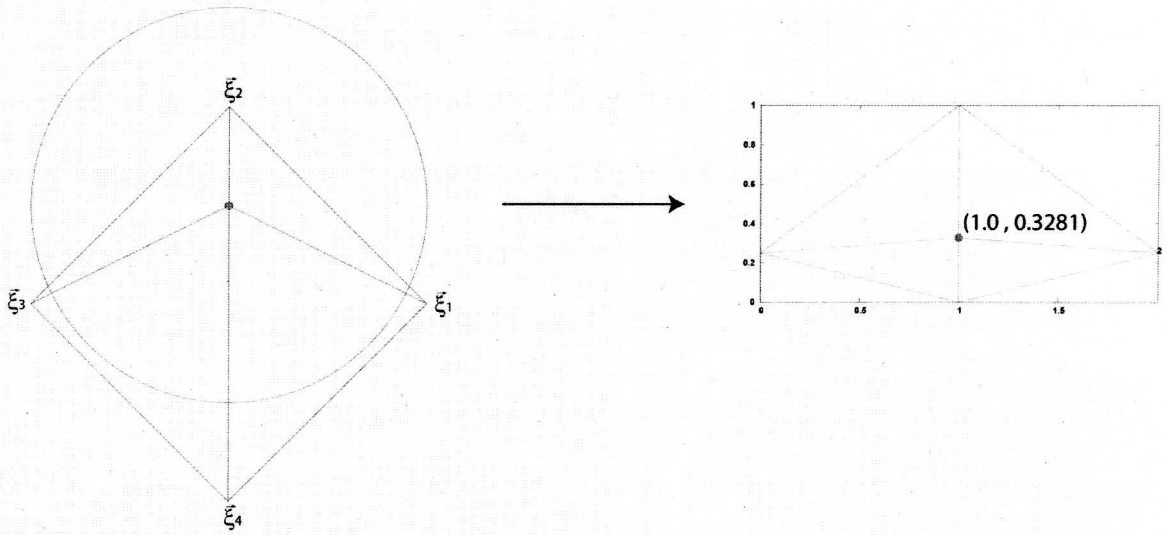


Figure 109 – Mesh smoothed using computational space offset from the origin but not modified using Riemannian metric tenor.

6.6.2 – Box16 Mesh

The next test case that was run was a case with multiple free nodes in a viscous-like formation. All of the free viscous nodes in this case were valence-6. The results, which are shown on Figure 110 show that the anisotropic computational space is able to retain the viscous characteristics of the mesh as it is smoothed.

6.6.3 – NACA0012

In order to explore and refine the behavior of the Winslow elliptic smoothing equations, the equations were applied to a NACA0012 airfoil mesh with viscous spacing near the airfoil surface using anisotropic control volumes. The full domain of the unsmoothed NACA0012 mesh is shown below on Figure 111.

Figure 112 shows a close-up of the NACA0012 airfoil with the original viscous spacing and again after the Winslow smoothing equations have been applied to the mesh using isotropic control volumes for the computational space. It can be seen that, when the Winslow equations are applied in this manner, the viscous spacing is not retained near the surface and the ability of the mesh to capture a viscous boundary layer has been lost.

Now, instead of isotropic control volumes being used everywhere, anisotropic control volumes were used in the viscous region. The anisotropic Winslow algorithm calculates a normal and lateral spacing for each node in the viscous region (based on the original mesh). It also calculates the necessary rotation angles and transforms the computational space for each node in a manner similar to that shown on Figure 113.

Using anisotropic virtual control volumes for the computational space results in the NACA0012 mesh shown on Figure 114.

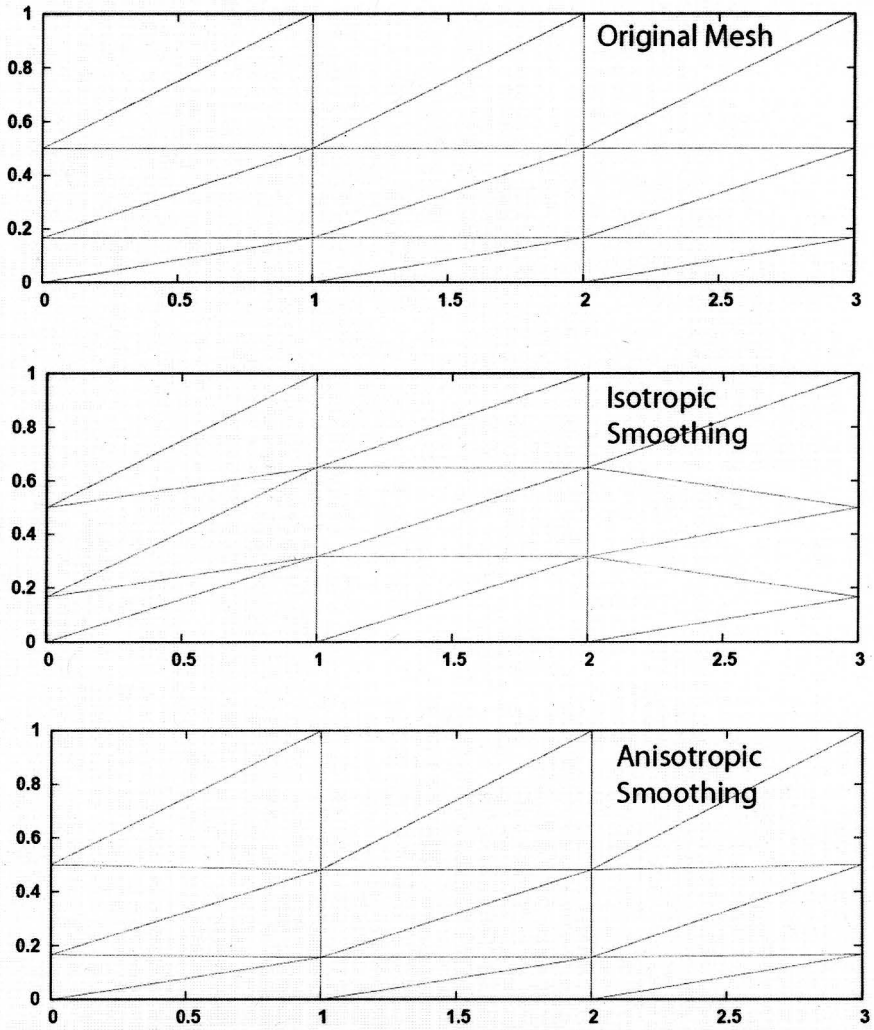


Figure 110 – Comparison of mesh smoothing using isotropic and anisotropic computational space.

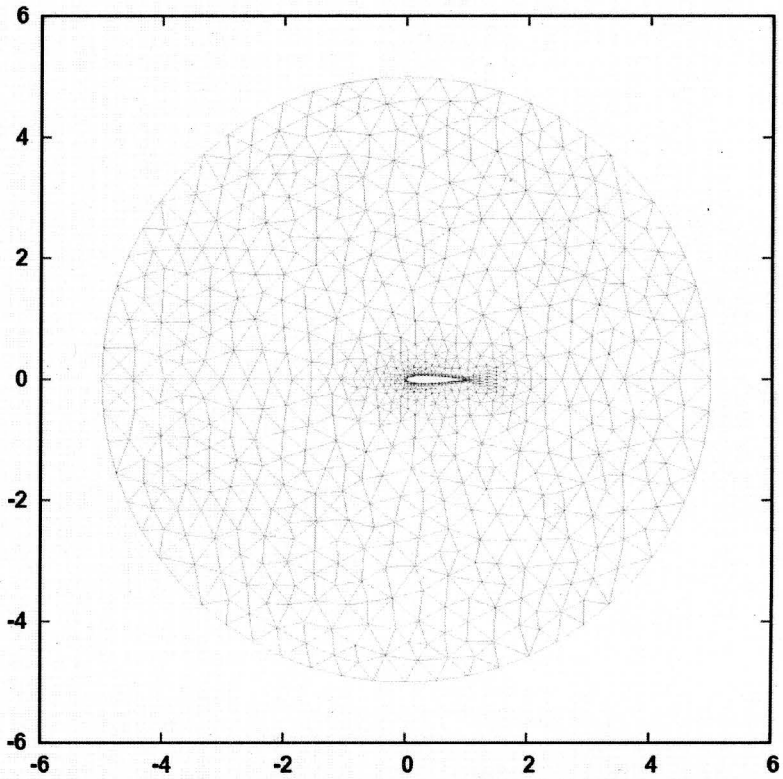


Figure 111 – Viscous mesh surrounding a NACA0012 airfoil.

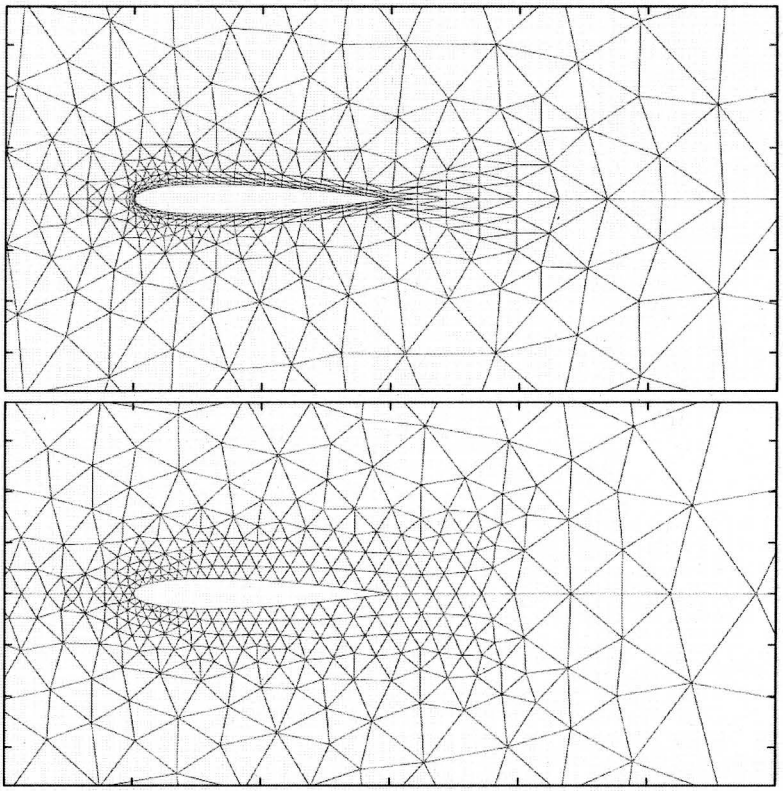


Figure 112 – Viscous mesh before and after isotropic Winslow smoothing.

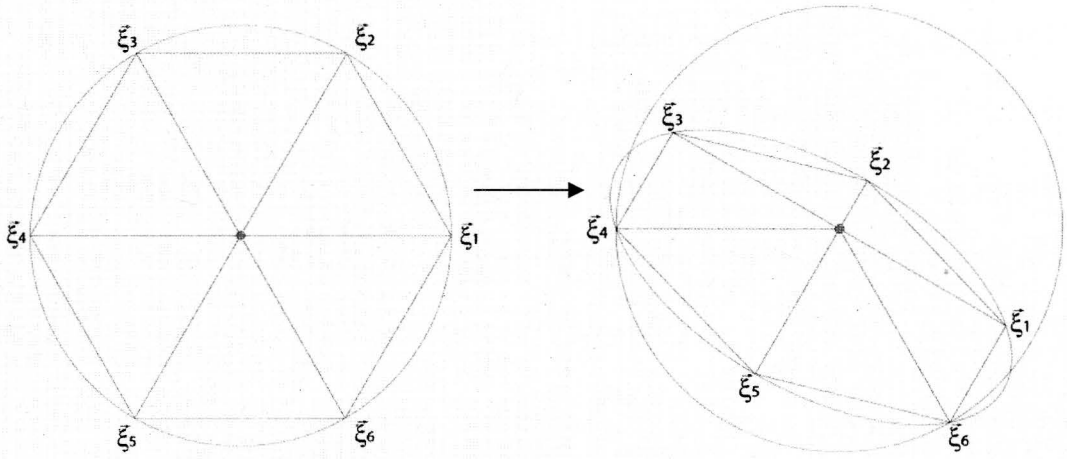


Figure 113 – Transformed computational space.

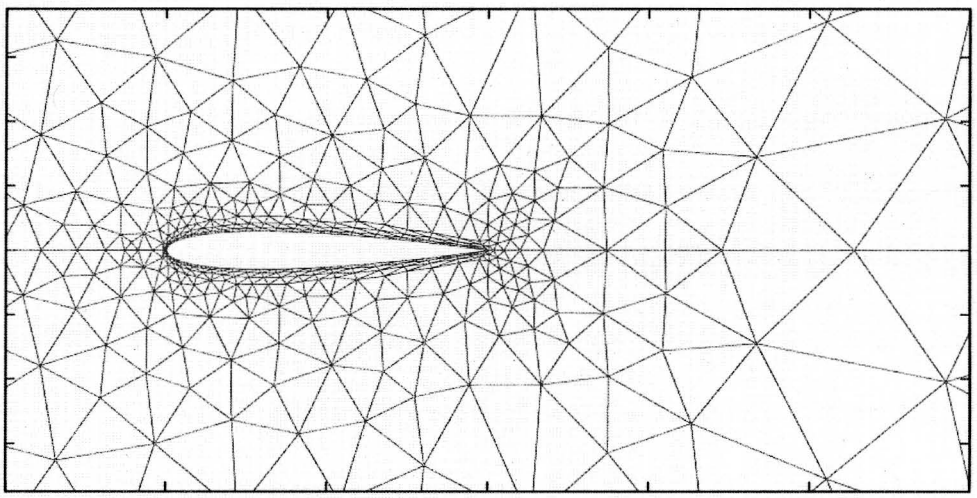


Figure 114 – NACA0012 airfoil mesh smoothed using anisotropic computational space.

6.6.4 – Sharp Corners

Figure 114 shows that, using anisotropic control volumes for the computational space, the viscous spacing is retained. However, zooming in on the back of the airfoil reveals that there is an issue with the sharp trailing edge. This is illustrated on Figure 115 and zoomed further in Figure 116. In these figures, shown below, the first node off the sharp trailing edged is marked in green for easy identification.

Consider the node directly off the trailing edge (highlighted in green on Figure 115 and Figure 116). This node (as well as the next two nodes trailing the airfoil) was marked as a viscous node. However, the viscous methodology had no effect because the node was on a symmetry plane and therefore had a zero offset in computational space.

To deal with this, logic was added to the anisotropic Winslow algorithm to manually apply a weighted offset to a sharp corner. Consider a case where the trailing edge is the first neighbor in the neighbor list for the central node (the central node is highlighted in green on Figure 117). The computational space for an offset of 0.0, 0.2 and 0.5 can be seen on Figure 117 and the resulting physical meshes can be seen on Figure 118.

Problems with sharp corners not only occur with viscous spacing but are also an issue with Winslow smoothing in general. Because of this, offset computational spaces can be advantageous even when an inviscid mesh is desired.

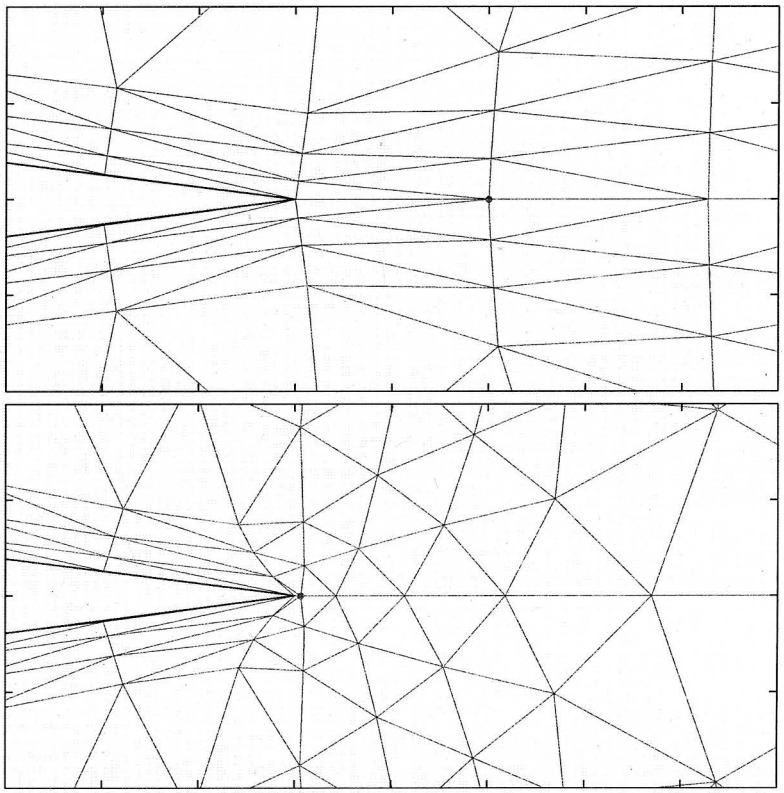


Figure 115 – Trailing edge of a NACA0012 mesh before and after smoothing has been performed.

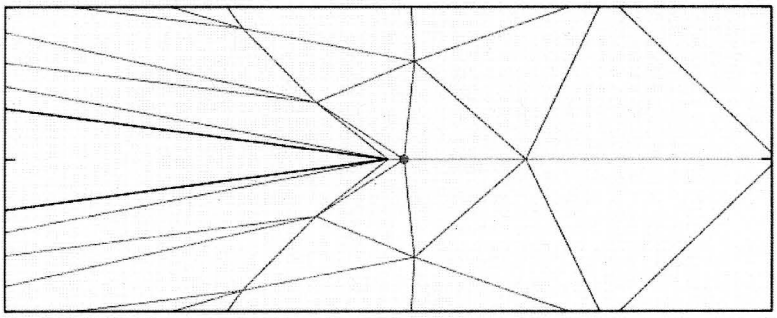


Figure 116 – Close-up of trailing edge after mesh has been smoothed.

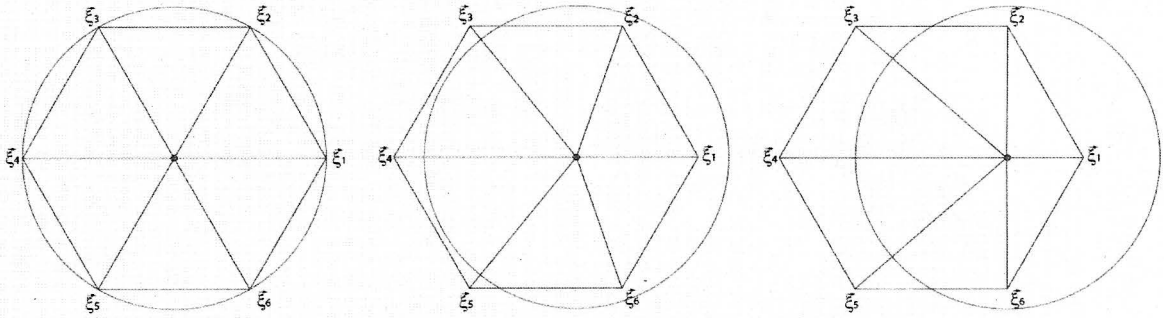
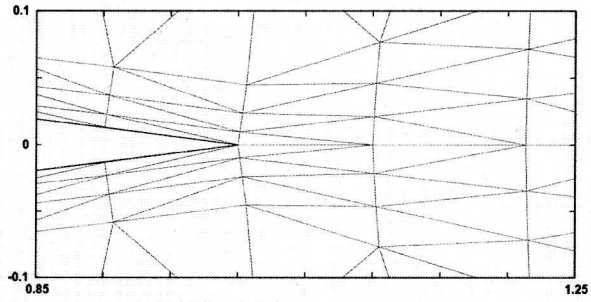
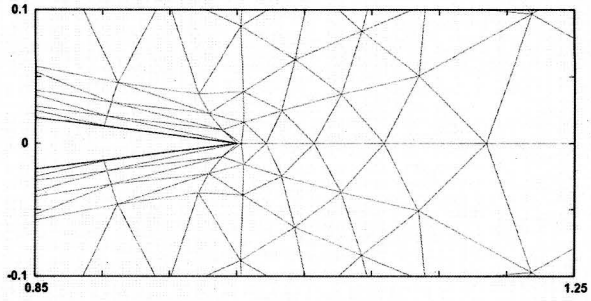


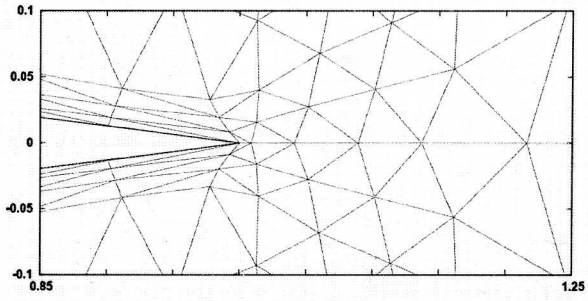
Figure 117 – Offsets in Computational space of 0.0, 0.2 and 0.5.



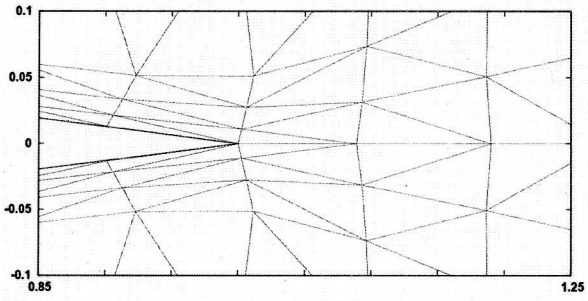
Original Mesh



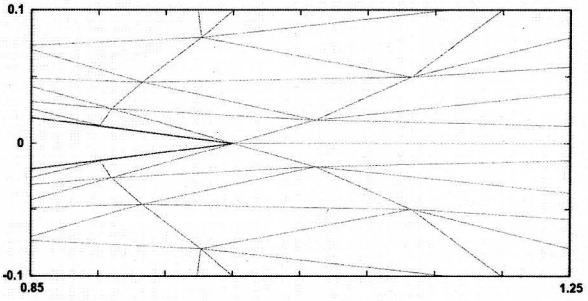
Offset = 0.0



Offset = 0.2



Offset = 0.5



Offset =0.7

Figure 118 – Physical space generated from various virtual control volume offsets.

6.6.5 – Anisotropic Weighting Factor

One other issue that needed to be addressed was that, using an anisotropic computational space in the manner described above, the mesh is drawn in toward the surface in areas of high curvature; this is most noticeable at the leading edge. The original mesh is compared to the smoothed mesh near the leading edge on Figure 119 and it can be observed that the mesh is obviously being driven closer to the airfoil surface.

One way this can be addressed is by adding a weighting factor to the offset that is used when manipulating the computational space of the nodes in the viscous region. The effect of applying an offset can be seen below on Figure 120, where the nodes in the viscous region are given a weighting factor of 0.8 and 0.6 respectively (compared to 1.0 in Figure 119).

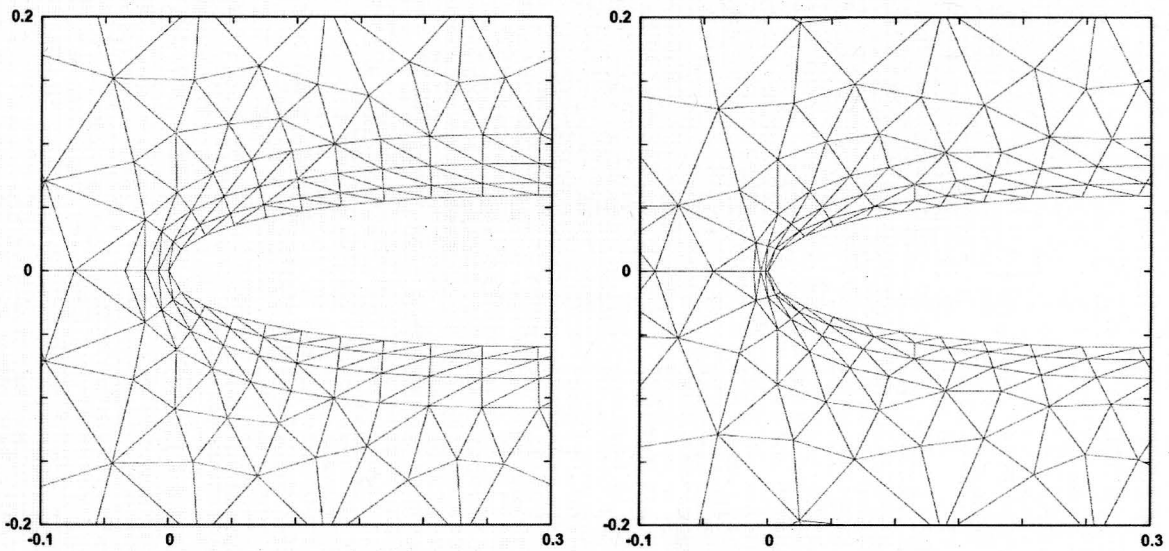


Figure 119 – NACA0012 leading edge before and after smoothing.

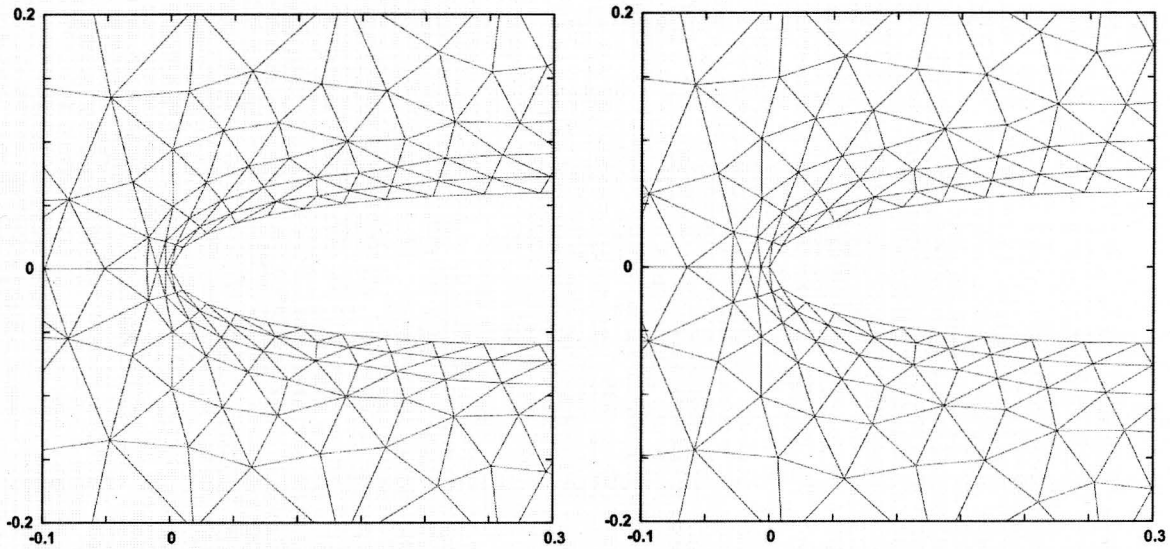


Figure 120 – NACA0012 smoothed leading edge after an offset weighting factor has been applied.

Because the Winslow equations are influenced by neighboring grid spacing (whereas the original viscous spacing was uniform around the airfoil regardless of other influences) it is not possible to perfectly match the original spacing everywhere on the airfoil using a uniform weighting factor.

6.6.6 – Elliptic-Mapped vs. Hybrid Computational Space

Observation of the effect of the computational space on the physical mesh leads to the conclusion that the shape of the computational space does not directly affect the physical mesh but, rather, it is the gradients of the physical coordinates with respect to the computational space that affects the physical mesh. The most revealing example of this is that the computational space can be mapped from a circular shape to a significantly different elliptic shape and the physical mesh will not be affected in the least if the gradients remain unchanged.

By emulating the physical mesh qualities by applying an offset to the computational space surrounding a given node, the mesh gradient (most notably the geometric progression in a viscous layer) can be simulated. However, the orthogonal orientation of the mesh with respect to the surface can be degraded. This is illustrated on Figure 121.

When the elliptic-mapped computational space (i.e., computational space that has been mapped from a unit circle to an ellipse using Riemannian Metric Tensors) is used, large angles may be generated. This is of concern because it has been shown that flow prediction accuracy decreases if one of the angles of a triangular element becomes large [17].

If an iteratively adapted computational space (described in detail in the next section) is not used, the best way to keep the orientation of the mesh orthogonal near the surface is to use the original unmoved physical space near the surface as the computational space. Note that this requires that a viable viscous mesh exists, which may or may not be the case. When physical elements in the viscous region are used as the computational space, the results are superior to those generated using elliptic-mapped virtual control volumes as the computational space. This is illustrated on Figure 122.

Even if the exact original physical mesh is used as the computational space in the viscous region, there is still an issue with the mesh being driven into the surface at a corner (sharp or otherwise). This is noticeable at the moderate corners near the leading edge but is especially obvious at the trailing edge (see Figure 123).

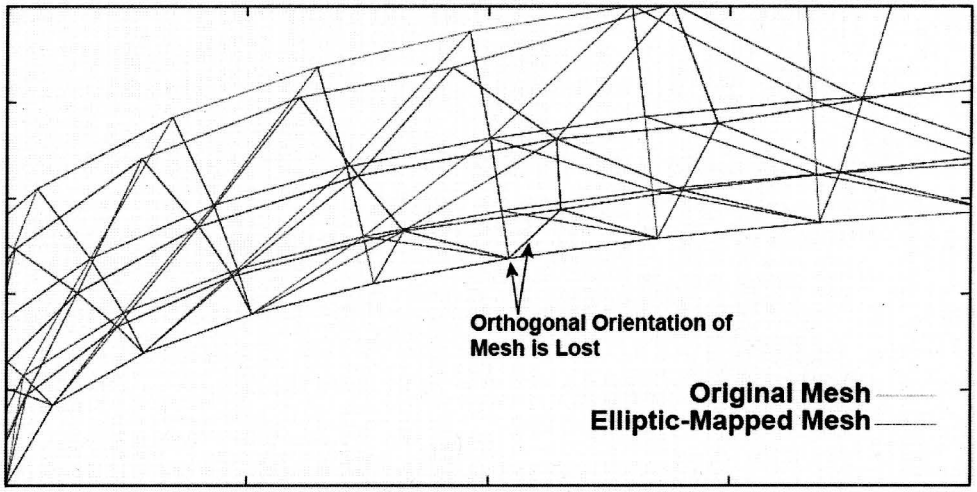


Figure 121 – Viscous layers after smoothing.

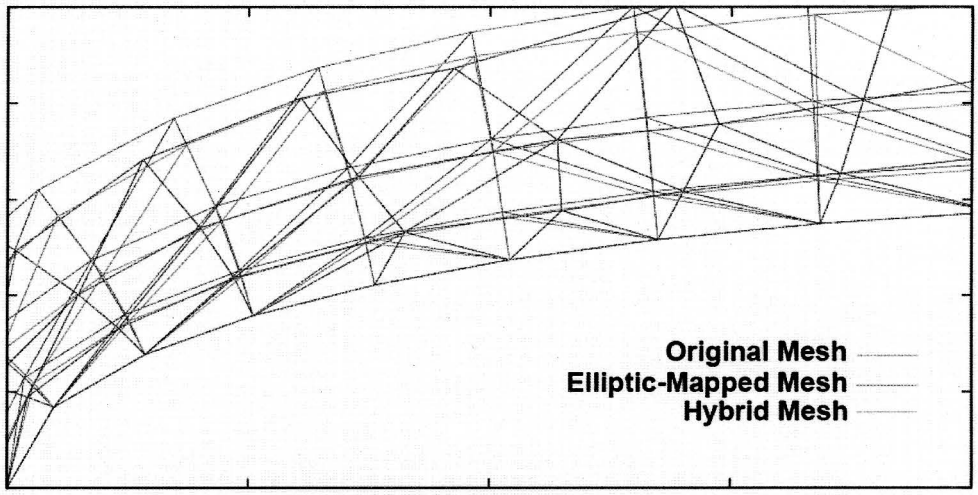


Figure 122 – Mesh comparison using differing computational space manipulations.

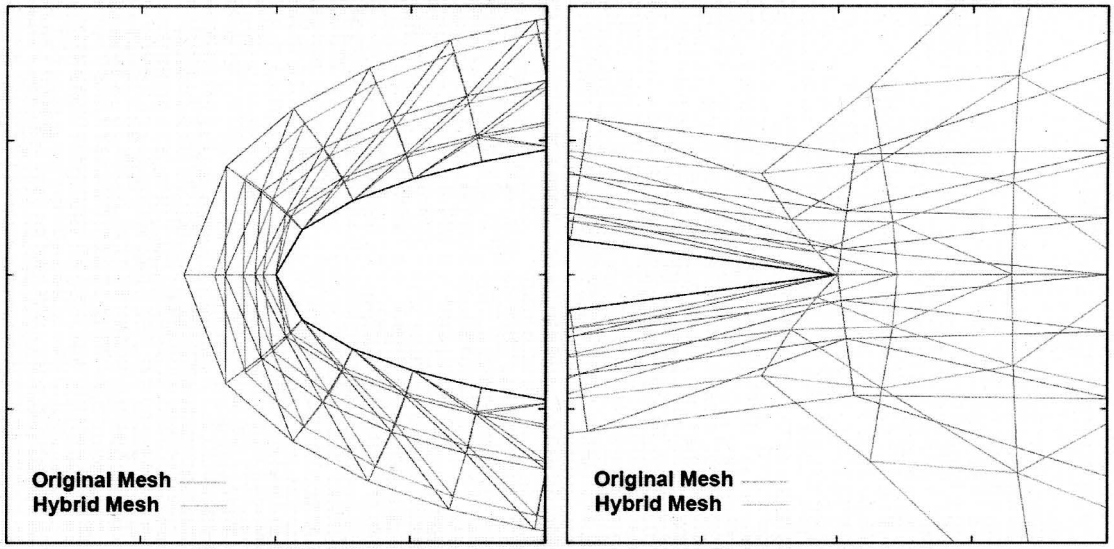


Figure 123 – Smoothed mesh attracted to surface near corners.

In an attempt to better understand the corner issue described above, a simple airfoil with very specific angles was created (shown on Figure 124) and used to see if there were any exploitable correlations between the angles of corners on a surface and the magnitude of the amount the mesh was being driven toward the surface. An example of a smoothed mesh compared to the original can be seen on Figure 125. An exploitable correlation was not detected but the simpler geometry did provide a good testbed for further observation.

The rough airfoil shown on Figure 124 was used to conduct the first test combining the viscous region Winslow algorithms with the moving boundary node algorithms. The rough airfoil was rotated 45° from its original position and the surrounding mesh was smoothed using the various Winslow smoothing implementations. Good results were generated and the results are shown on Figure 126 and Figure 127.

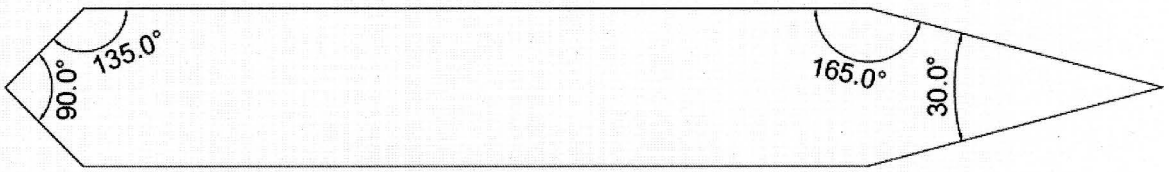


Figure 124 – Rough airfoil with explicitly defined interior angles.

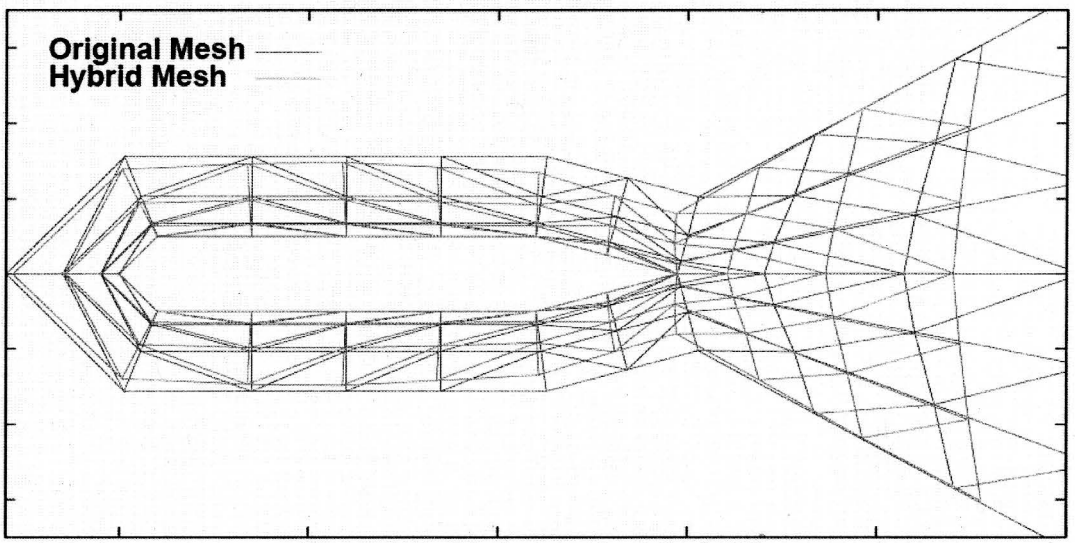


Figure 125 – Mesh around rough airfoil.

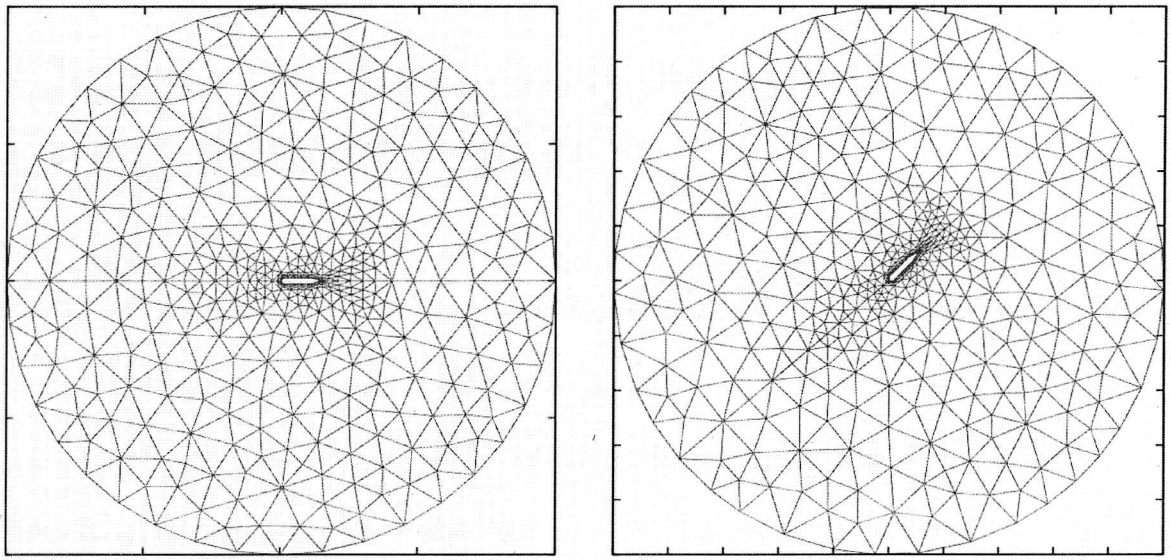


Figure 126 – Results combining algorithms for Winslow on surface nodes with Winslow in viscous regions.

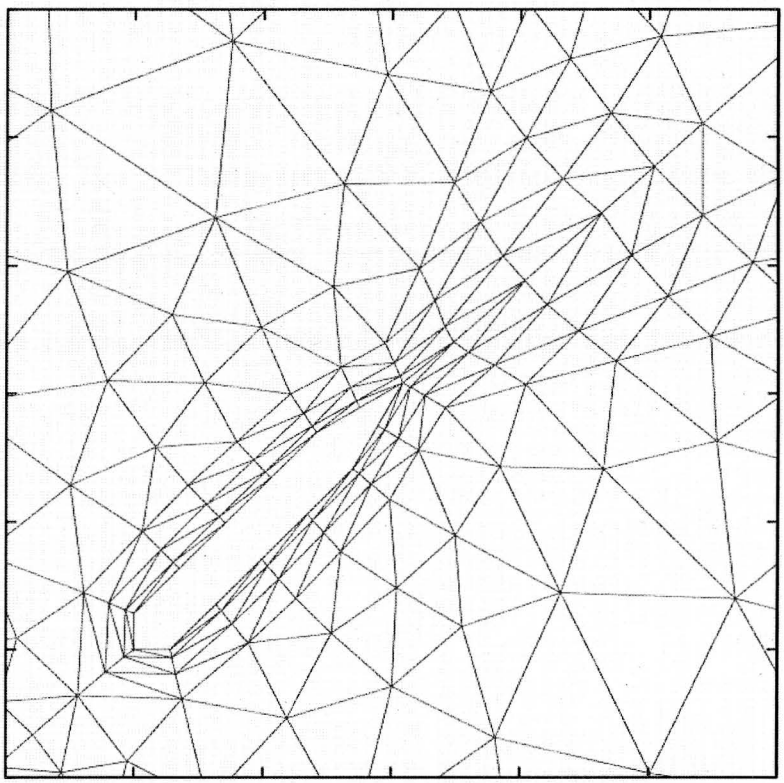


Figure 127 – Close up of rotated rough airfoil with viscous spacing.

Chapter 7 – Iteratively Adapted Computational Space

In order to get a high-quality smoothed mesh in the viscous layer surrounding a no-slip wall using the techniques described in Chapter 7, there needs to be a viable viscous mesh to reference (one with properties similar to the properties desired in the final mesh). Obviously, this might not always be the case so it is necessary that a technique be devised that allows for the creation of a viscous mesh with general spacing properties even if a viable viscous mesh is not available. This technique needs to allow a mesh with an amenable connectivity structure to be smoothed in such a way that it could match a desired viscous profile based on an initial off-body spacing and geometric progression of the viscous layers. The technique that was devised for this purpose was to iteratively adapt the computational space to meet the requirements of the physical system.

The initial mesh that was used to experiment with this new technique was an inviscid (isotropic) mesh around the rough airfoil first seen in Chapter 7. The mesh is shown below on Figure 128 and, near the airfoil, on Figure 129.

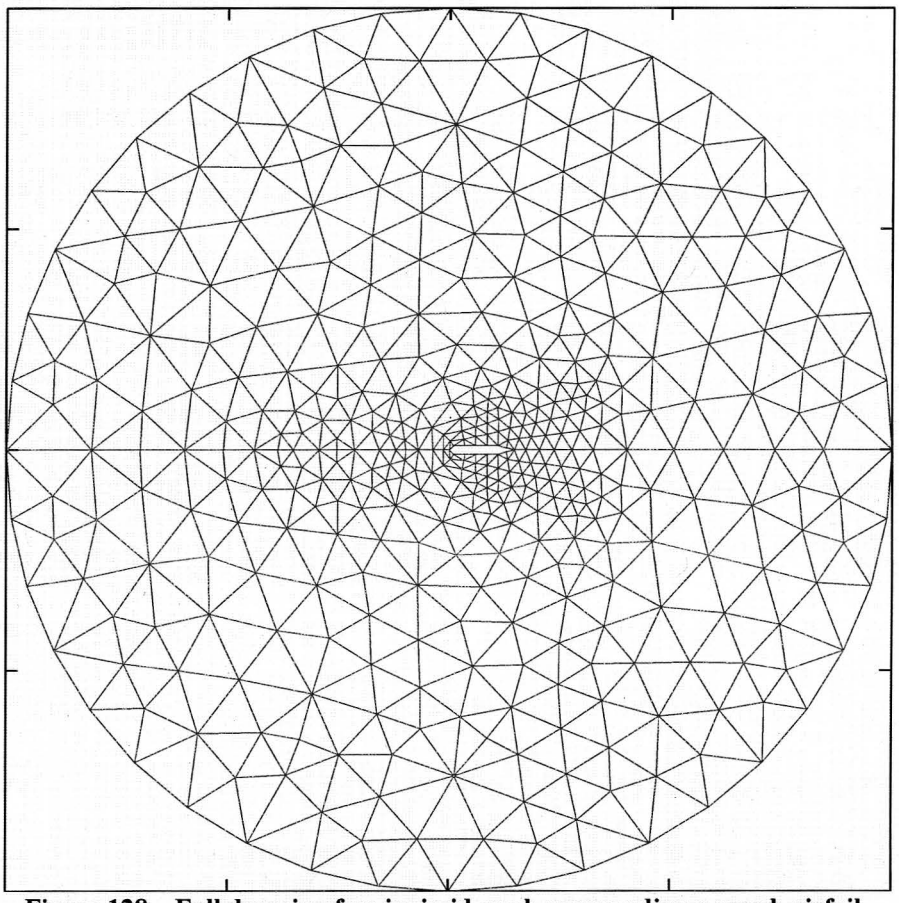


Figure 128 – Full domain of an inviscid mesh surrounding a rough airfoil.

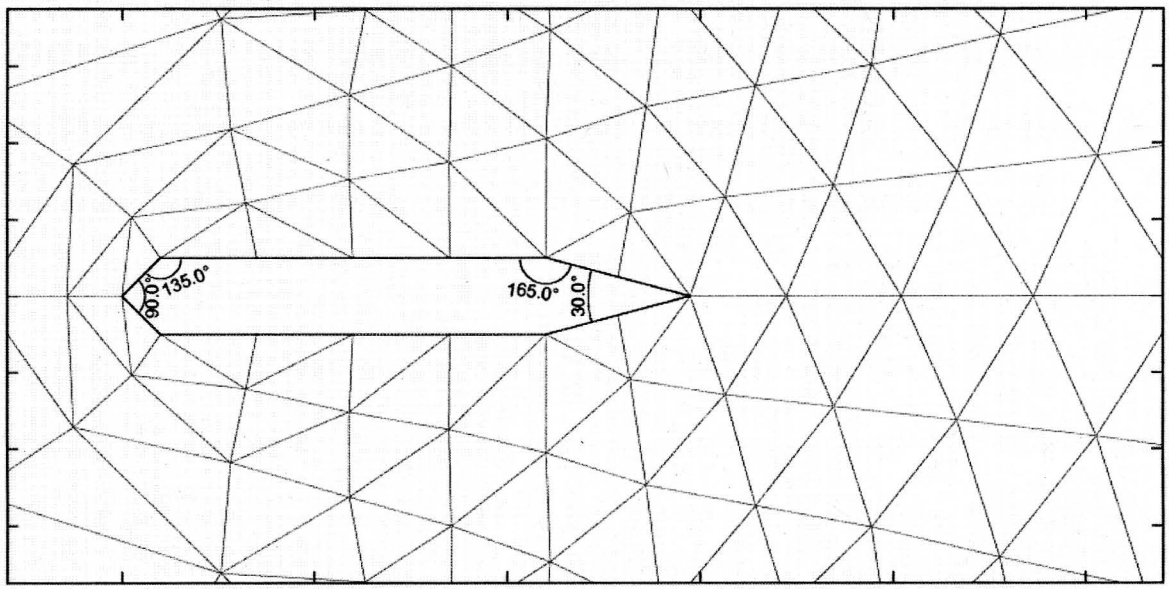


Figure 129 – Inviscid mesh around a rough airfoil near the airfoil surface.

7.1 – Normal Offset

Throughout the development of the iteratively adaptive technique and algorithm, the computational space that is used as the starting point on which to adapt is the equal angle, equal edge-length virtual control volume shown as the first image on Figure 117. The initial step in the development of an iteratively adaptive viscous smoothing algorithm was to use a geometric progression factor to create the offset in computational space that would be normal (orthogonal) to the no-slip surface in physical space. The offset can be defined as the distance from the central node in computational space to the point at which the center of all the neighboring nodes is located. (A note on terminology: as mentioned in chapter 7, the node referred to as the central node is the node-of-interest or the node being smoothed, it is always located at the origin in computational space and the neighboring nodes are translated around it to accommodate the characteristics of the physical mesh.) The direction of the normal offset is determined by which of the central node's neighbors is the near-surface node (where near-surface node refers to the neighboring node either directly on the no-slop surface or on the direct path back to the surface). This near-surface node will have a given position in the list of neighbors and that position will determine where the node is positioned in computational space.

In the case where the associated surface node is located at position ξ_1, the offset in computational space will be as shown on Figure 130. By offsetting the computational space in this manner, the distance between the central node in computational space and the near-surface node (distance L_1) is now less than the distance between the central node and the node opposite the surface (distance L_2). This gradient in computational space causes the node in physical space to move closer to the surface as well.

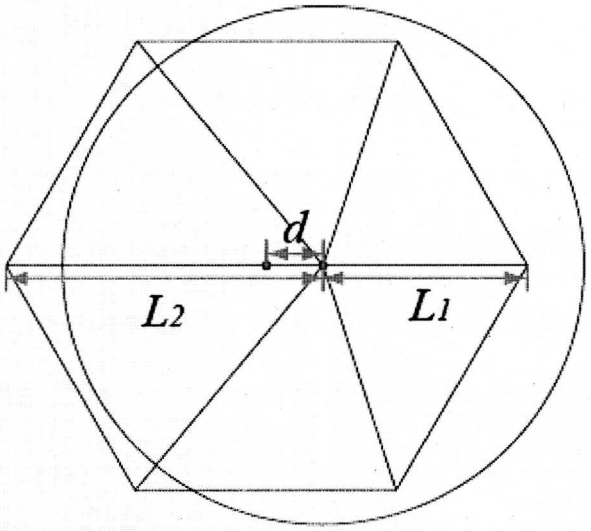


Figure 130 – Computational space offset normal to a viscous surface.

The magnitude of the offset (d in Figure 130, where the virtual control volume is based on a unit circle; i.e., radius=1) is based on the geometric progression factor (g) and is calculated as follows.

$$g = \frac{L_2}{L_1}$$
$$L_2 = 2 - L_1$$
$$L_1 = 1 - d$$
$$g = \frac{1+d}{1-d}$$

A little additional algebra gives the equation in terms of the offset, d:

$$g = \frac{1+d}{1-d}$$
$$g - dg = 1 + d$$
$$dg + d = g - 1$$

$$d = \frac{g-1}{g+1} \tag{7.1}$$

If nothing but the geometric progression based normal offset is used to modify the computational space control volumes, a viscous mesh can be achieved without iterative computational space adjustment but, in order to get a desired off-body spacing, the number of layers in the viscous region would have to be such that the normal distance of the viscous region (i.e., the distance from the surface to the edge of the viscous region) was pre-calculated to equal the sum of the progressive distances of each layer. This is a limitation that can be overcome by adding a loop into the Winslow solver that recalculates the offset "on the fly" to match off-body spacing in the first viscous layer and the progressive spacing in the following layers. Consider the mesh shown below on Figure 131 with an initial off-body spacing of 0.04 and a geometric progression factor of 1.5. This will give the viscous layer distances shown in Figure 131, where *p1*, *p2* and *p3* are all associated with the surface node *p0*.

After a determined number of smoothing iterations, the normal distances in the viscous region (i.e. the distance from the viscous node to the associated surface node) will be compared to the desired distances and, if they do not match, the computational space will be adjusted accordingly. For example, if the distance *d2*, is greater than 0.6, the offset will increase. This will increase the computational-space gradient in the normal direction and draw *d2* closer to *d1*. When this technique is applied to the inviscid mesh shown on Figure 128, the mesh shown in Figure 132 and Figure 133 results.

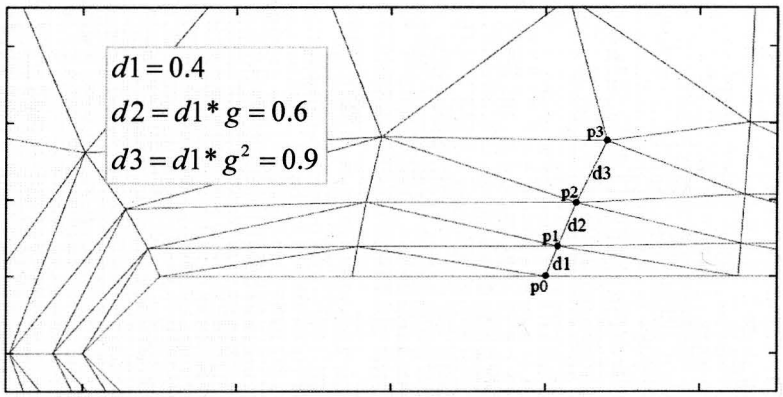


Figure 131 – Mesh smoothed using normal offset.

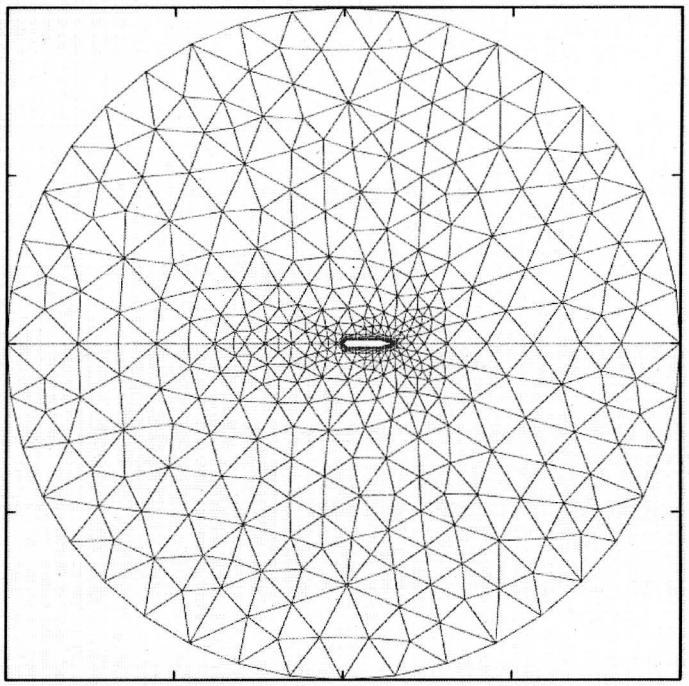


Figure 132 – Rough airfoil smoothed using normal offset computational space.

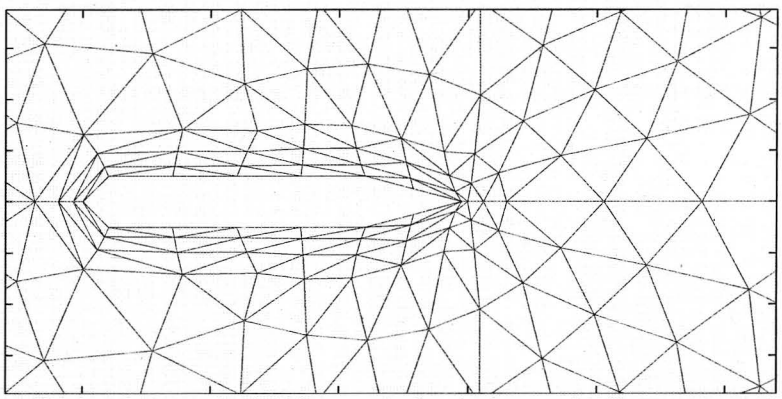


Figure 133 – Close-up of rough airfoil smoothed using normal offset computational space.

7.2 – Normal and Lateral Offset

As shown on Figure 132 and Figure 133, the spacing profile of the viscous layers is quite good. However, using only the normal offset algorithm, it is not possible to manipulate the spacing in the lateral direction (parallel to the surface) so the nodes in the viscous region do not remain orthogonal to their associated surface point. This results in the situation shown in Figure 134 where the angle, α, begins to get larger and can result in numerical instability [17].

The lateral spacing issue is handled by implementing another offset in computational space, this time in a direction orthogonal to the normal offset direction. The computational space will now look like Figure 135.

There are two possibilities to direct the lateral offset: normal offset direction $+90^{o}$ or normal offset direction -90^{o}. The direction is based on which side of the surface normal the node is on. However, this can be handled implicitly by using the cross product (equation (7.2)) of the unit normal surface vector $(\vec{n})$ and the unit vector that passes through the node that is going to be smoothed $(\vec{v})$ (illustrated on Figure 136). The cross product of two vectors is itself a vector but because two-dimensional meshes are being utilized, the direction will simply be $\hat{k}$ or $-\hat{k}$ so the value of the cross product can be effectively treated as a scalar value.

$$\vec{n} \times \vec{v} \tag{7.2}$$

The cross product defined by equation (7.2) is extremely well suited for the task of smoothing a viscous node in the lateral direction. It is a powerful tool because it gives information on both magnitude and direction. By using the cross product, the initial direction of the lateral offset can always be given by the normal direction + 90°. If the cross product is negative, the direction will effectively become -90° so the direction does not have to be explicitly adjusted if the node switches from one side to the other side of the surface normal. Also, as the node gets closer to the normal direction, the magnitude of the cross product is equal to twice the area of the triangle formed by the two vectors ($\vec{n} \times \vec{v} = 2A$, the area is shown in light blue on Figure 136) will get smaller and smaller so the cross product (multiplied by some relaxation factor) intrinsically makes a powerful parameter that can be used to adjust the offset and measure convergence. Because the magnitudes of the vectors in Figure 136 and the radius of the virtual control volume have all been normalized to 1, the initial offset can be calculated as the height (h) of the triangle formed by the two vectors. The area of a triangle is $A = \frac{1}{2}bh$. The base (b) of the triangle is one, so h becomes:

$$h = 2Ab = \vec{n} \times \vec{v} \tag{7.3}$$

The computational space for the viscous nodes is adjusted at given intervals (say, every 100 smoothing iterations). The algorithm that is used to calculate the normal and lateral offsets and adjust the computational space accordingly is described by the following bullets. When this algorithm is implemented and applied to the rough airfoil, the results shown on Figure 137 and Figure 138 are achieved.

- The initial normal offset is calculated based on the geometric progression factor (see equation (7.1)), which is read in as an input.
- The initial lateral offset is calculated as the distance h (see equation (7.3)), based on the cross product of the unit surface normal vector ($\vec{n}$) with the unit vector from the associated surface node to the central node ($\vec{v}$).
- The normal offset direction is calculated based on the position of the near-surface node in the array of neighboring nodes. (For the node in the first viscous layer, the near-surface node will be the associated surface node, for a node in the 2nd layer, the near-surface node will be the associated node from layer one, etc. For a node with 6 neighbors: If the near surface node is first in the neighbor-array, the offset will be away from the 0^{o} position; if the near-surface node is second in the neighbor-array, the offset will be away from the 60^{o} position, etc.)
- The initial lateral offset direction is determined to be the normal offset direction $+90^{o}$. (If the cross product is negative, this will effectively behave like -90^{o}.)
- At each computational space adjustment iteration the normal offset is adjusted by 10% of its original value to target in on the offset that will give the proper off-body spacing.
- At each computational space adjustment iteration the lateral offset is adjusted by the new cross product multiplied by a relaxation parameter to target a position normal to the no-slip surface (a relaxation factor of 0.1 has been found to give good results but might need to be decreased as the initial off-body spacing gets closer to the viscous surface).

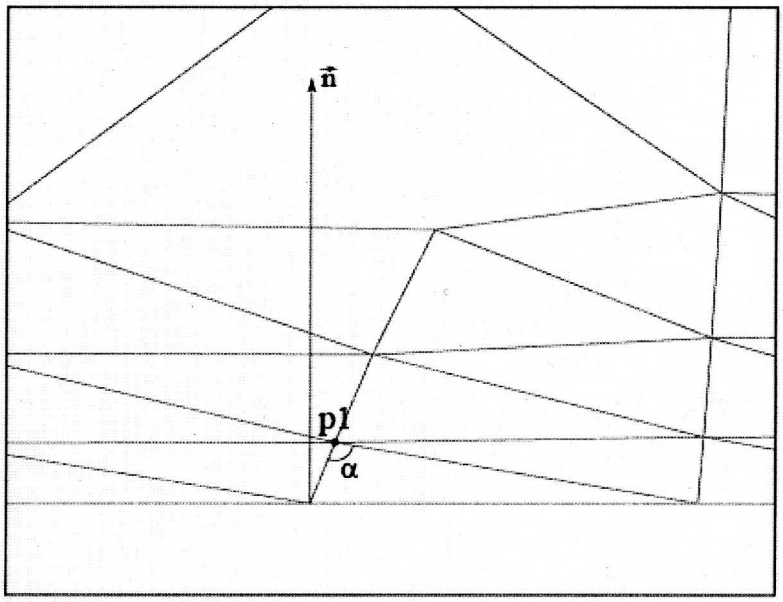


Figure 134 – Normal offset computational space can result in large angles.

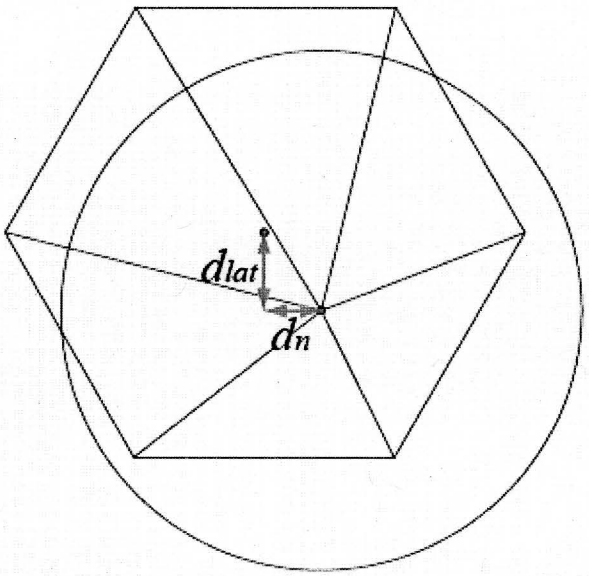


Figure 135 – Computational space with offsets in both the normal and lateral directions.

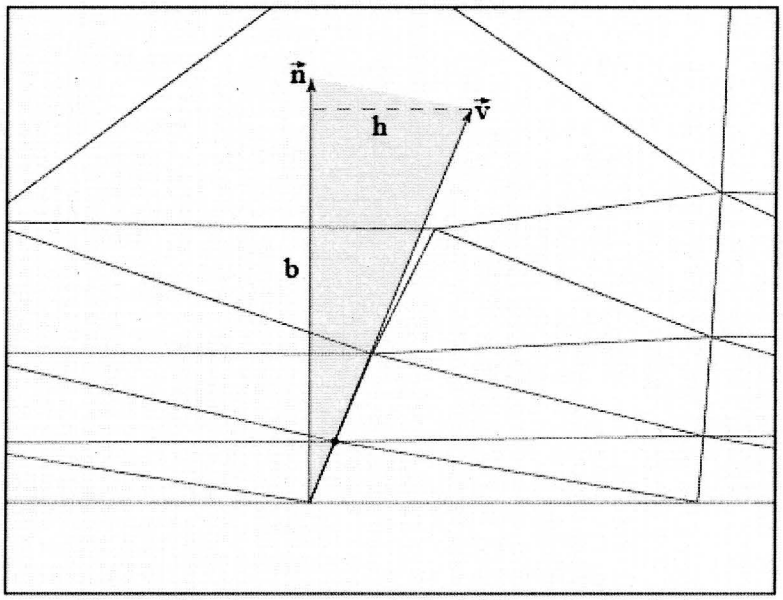


Figure 136 – Area representation of the cross product.
This figure shows the area representation of the cross product that is formed by the unit normal vector of the surface at the location of the associated surface node and the unit-length vector from the associated surface node through the off-body node of interest.

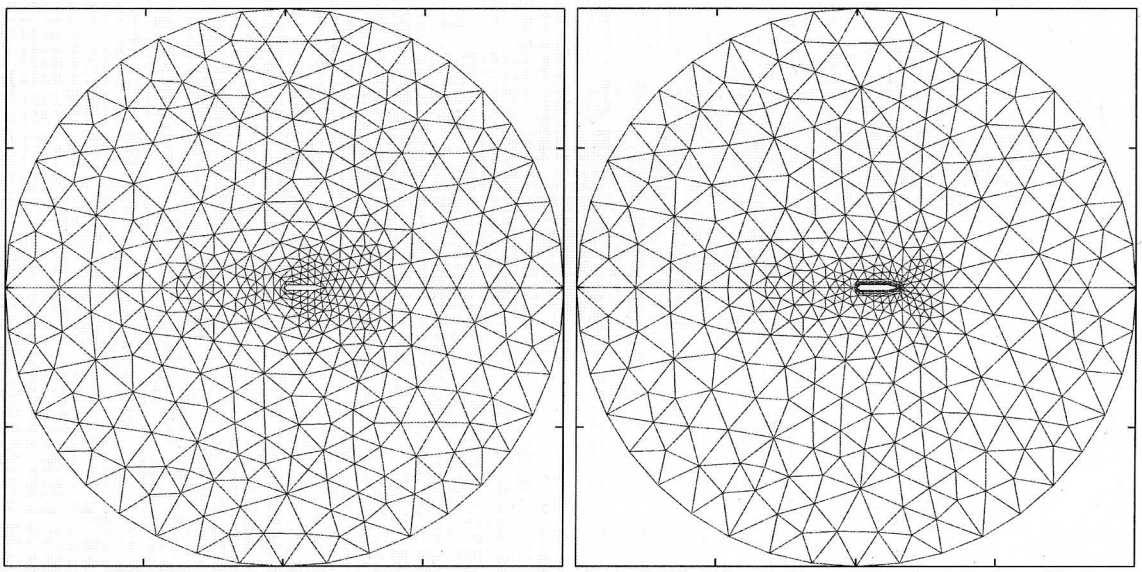


Figure 137 – Rough airfoil mesh smoothed using iteratively adapted computational space in the viscous region.

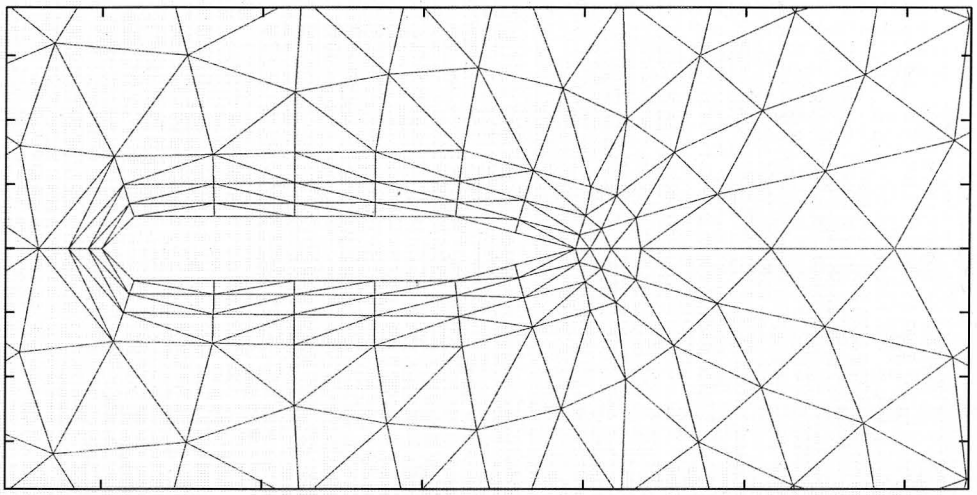


Figure 138 – Close-up of smoothed viscous region near the rough airfoil surface.

7.3 – Viscous NACA0012 Results

To further explore the Winslow equations using iteratively adapted offset computational space, the NACA0012 airfoil was employed. The full domain surrounding the NACA0012 airfoil is shown on Figure 139. A close up of the original mesh is shown on the left side of Figure 140 and Figure 141. The mesh that was used as the starting point for the smoothing was essentially an inviscid mesh; however, the mesh was generated using extrusion off the airfoil surface so that the connectivity was amenable to treating the nodes in the viscous region as viscous layers.

The viscous region surrounding the NACA0012 airfoil was specified to have five viscous layers and an off-body spacing and geometric progression factor were also specified (these values are specified as inputs in a viscous parameter file read in by the code). Once all of the viscous region parameters were specified, the iteratively adaptive Winslow smoothing algorithm was applied to the inviscid NACA0012 mesh. The results are shown on the right side of Figure 140 and Figure 141. Figure 140 and Figure 141 illustrate that the iteratively adaptive Winslow algorithm is doing an admirable job of meeting the viscous mesh requirements (i.e. mesh spacing sufficient to capture a viscous boundary layer and orthogonality of nodes in progressive layers of the viscous region).

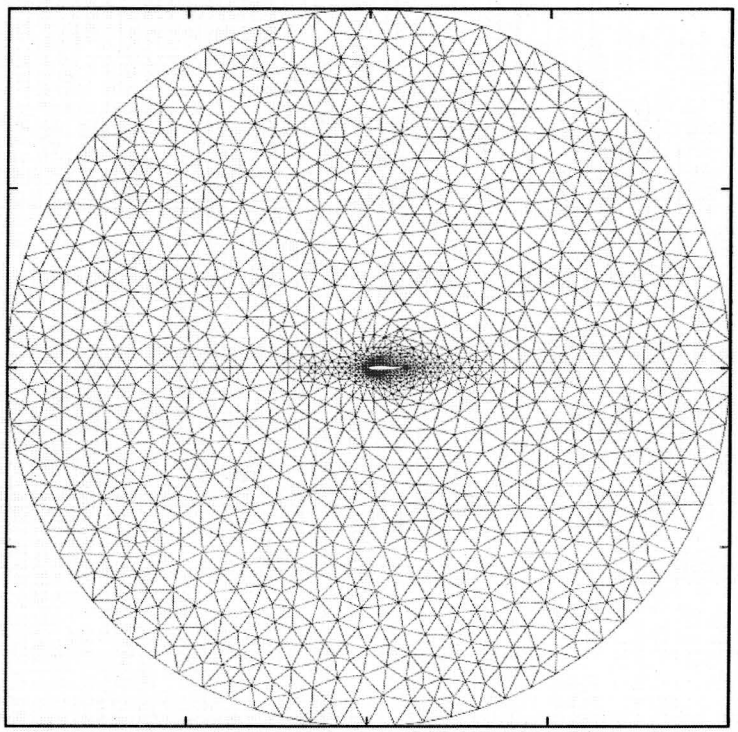


Figure 139 – Inviscid domain surrounding a NACA0012 airfoil.

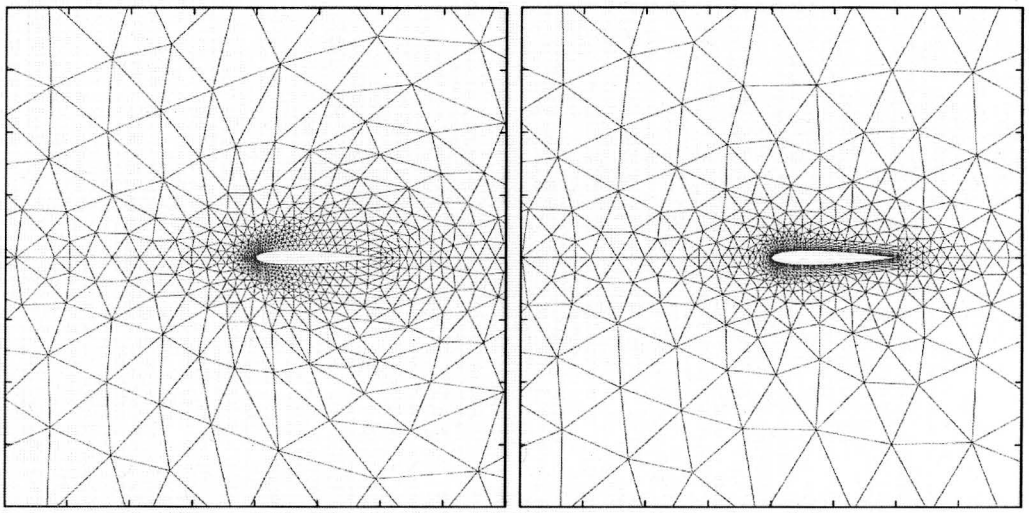


Figure 140 – NACA0012 airfoil before and after smoothing.

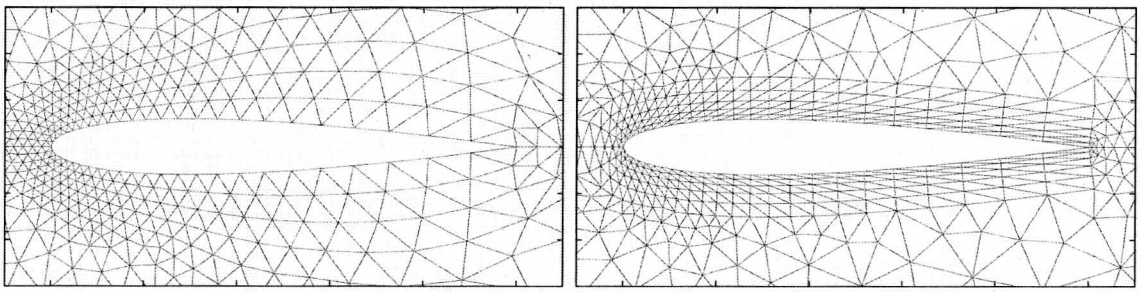


Figure 141 – Close up of NACA0012 airfoil before and after smoothing.

7.3.1 – Rotation and Translation

The Winslow elliptic smoothing equations using an iteratively adaptive computational space algorithm were tested on situations where an airfoil (the NACA0012) was moved around within the domain. The airfoil was rotated, translated and then both rotated and translated simultaneously. In any of the cases, the mesh can start out as viscous, inviscid or even an unviable mesh with grid crossing and negative volumes. The Winslow equations will cause the mesh to conform to the new position and orientation of the airfoil and the offset computational space will maintain the viscous layer regardless of orientation. The results from the mesh movements can be seen below on Figure 142, Figure 143 and Figure 144.

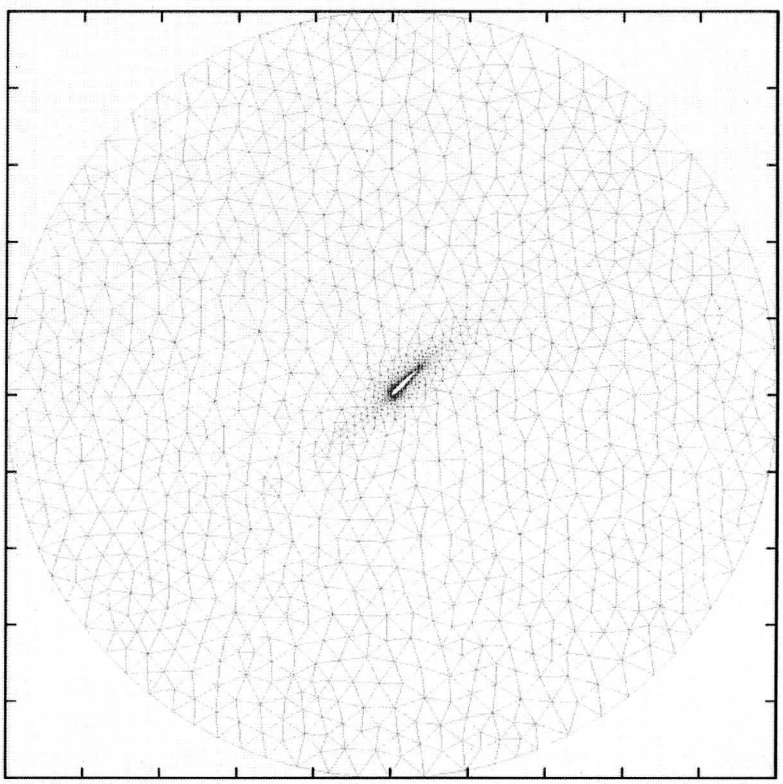


Figure 142 – NACA0012 rotation using Winslow iteratively-adaptive computational space algorithm.

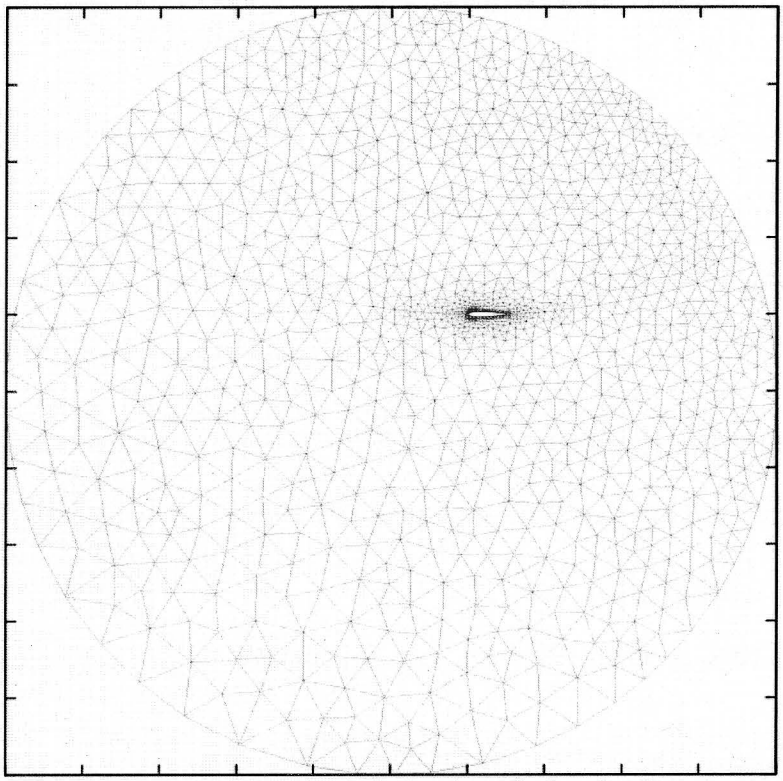


Figure 143 – NACA0012 translation using Winslow iteratively-adaptive computational space algorithm.

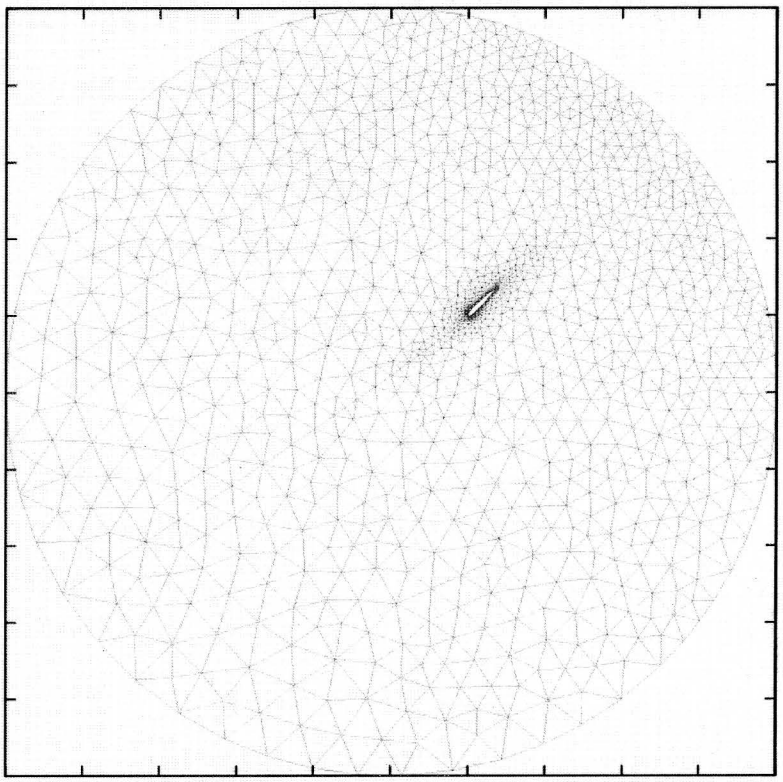


Figure 144 – NACA0012 rotation and translation using Winslow iteratively-adaptive computational space algorithm.

7.3.2 – Airfoil Shape Modification

The next area that was explored was to look at how the equations behaved on a deforming airfoil surface. This has many practical applications to parametric design where it may be required that many designs be examined but manually generating meshes for each design would be prohibitively expensive.

In many cases, the exact camber of an airfoil (for example) will not be known a priori. A design engineer might want to try many different incremental designs. However, the requirement that a mesh be generated for each design could potentially be significantly detrimental to the design process. By employing the Winslow equations using iteratively adapted offset computational spaces, no additional direct mesh generation would be required. The surface can be significantly deformed from its original shape and the smoothing equations will conform the surrounding mesh to the new shape and maintain the original off-body spacing and desired geometric progression. An example of this is shown on Figure 145.

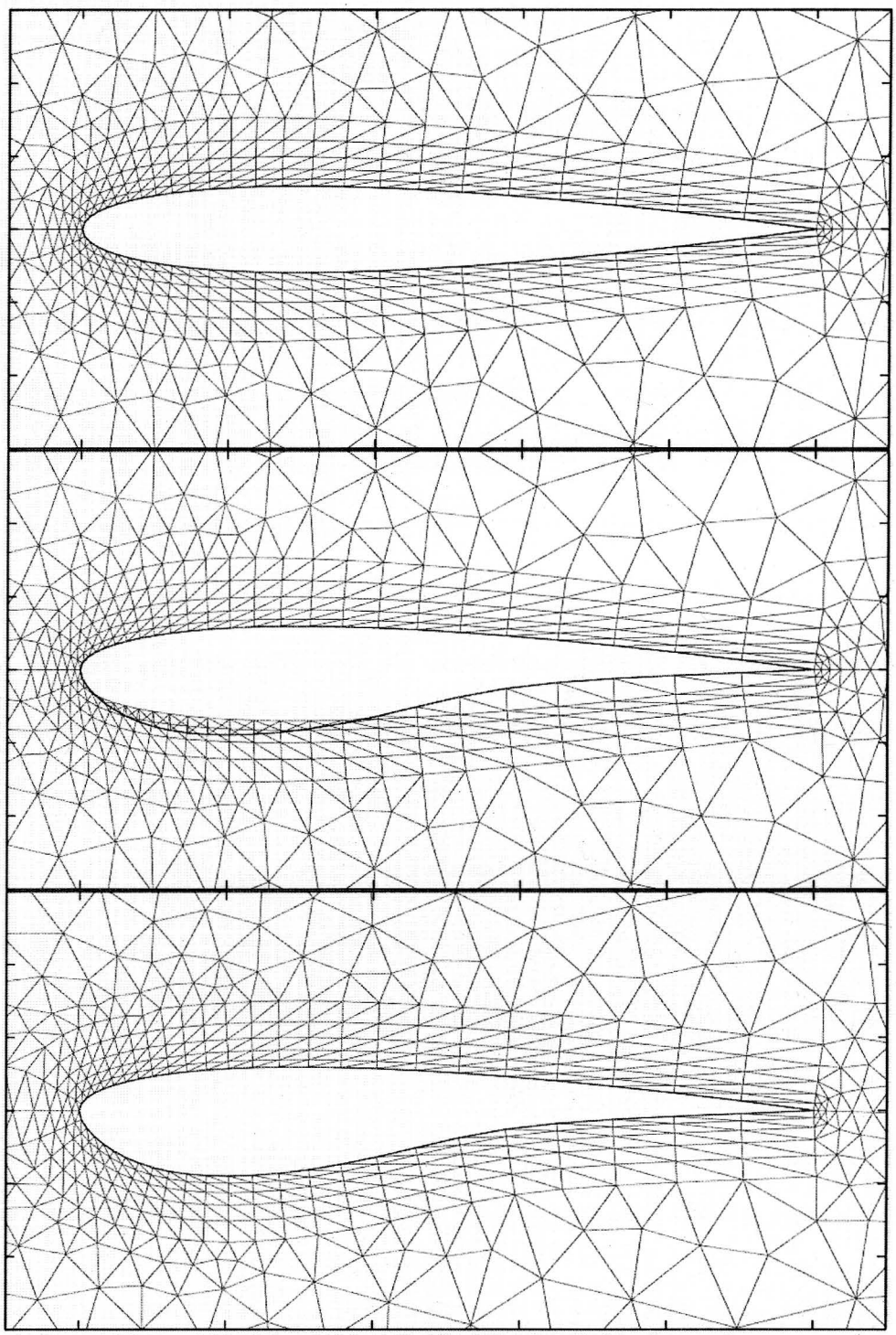


Figure 145 – Viscous mesh conforming to a deforming surface.
Note that even though significant grid crossing occurs when the airfoil shape is modified (second illustration) the iteratively adaptive Winslow algorithm is still able to generate a viscous grid with the desired characteristics.

7.3.3 – Change in Reynolds Number

Another powerful application of the iteratively adaptive Winslow smoothing algorithm would be in a case where the required off-body spacing was changing. This is something that is guaranteed to be an issue in any kind of testing environment because it will always be necessary to explore properties at different Mach numbers and atmospheric conditions. As these conditions change, the required off-body spacing changes as well. The viscous smoothing algorithm has the ability to quickly and efficiently change the off-body spacing or geometric progression of the viscous layers to accommodate changes to the flow characteristics with very little (or none, if the offset is coupled to the flow solution) manual adjustment to the mesh. Results are demonstrated below where the mesh starts out having inviscid properties and is then adjusted to accommodate different flow conditions.

The viscous mesh surrounding a NACA0012 airfoil that was seen in the previous examples of this section has an off-body spacing (Δs) of 0.010. For a y+ of 100 and a Reynolds number of 20,000 (and using a reference length of 1.0) this off-body spacing would be sufficient to yield 1 grid point in the laminar sublayer [18] and thus make this an adequate mesh for examining a viscous flow in a very low speed region using wall functions. The NACA0012 airfoil mesh at these conditions is shown below on Figure 146.

The mesh shown on Figure 146 would only be sufficient for capturing a boundary layer on extremely low speed flows and it would be likely that higher speeds would need to be explored as well. If the speed was increased 5 fold (resulting in a corresponding 5-

times increase in the Reynolds number) the required off-body spacing for the same y+ (y+ = 100) would now be: $\Delta s = 0.00233$. Using the iteratively adapted computational space Winslow equations, no new mesh would need to be manually generated. A simple input change specifying the new off-body spacing would be sufficient to generate a new mesh with the required characteristics (it might also be necessary to adjust relaxation factors to the smoother). A mesh with characteristics necessary to analyze the flow at the new higher Reynolds number is shown on Figure 147 and a close-up comparison of the two meshes, along with the original inviscid mesh, is shown on Figure 148.

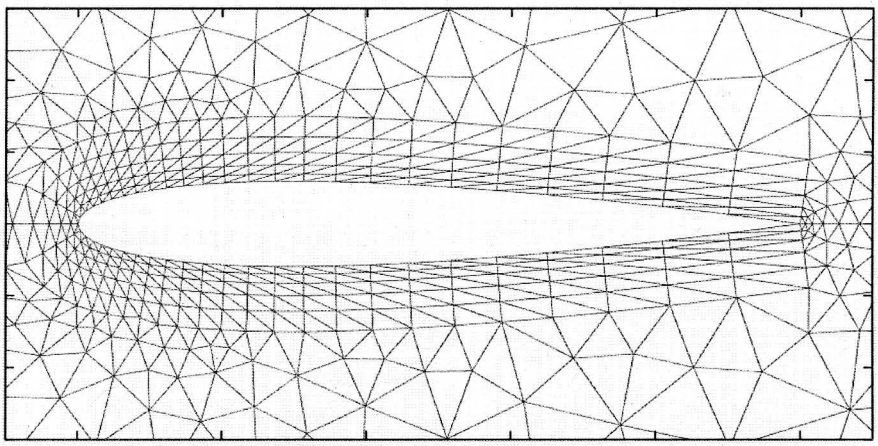


Figure 146 – Mesh with off-body spacing of Δs=0.01 (y+ = 100, Re = 20,000).

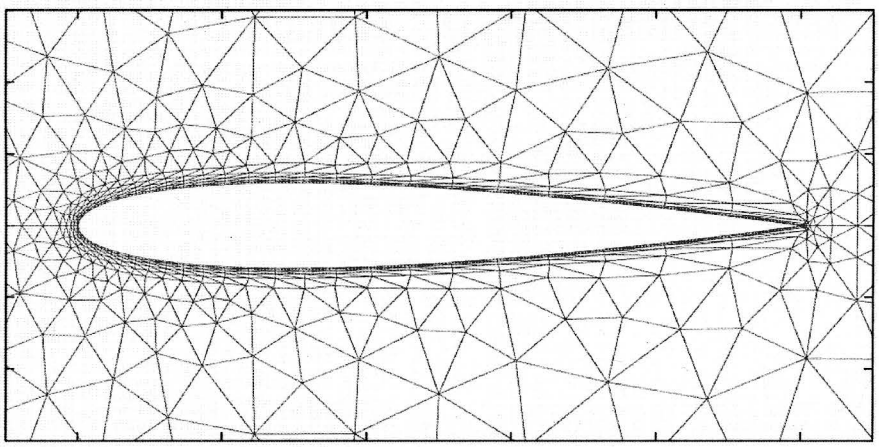


Figure 147 – Mesh with off-body spacing of Δs=0.00233 (y+ = 100, Re = 100,000).

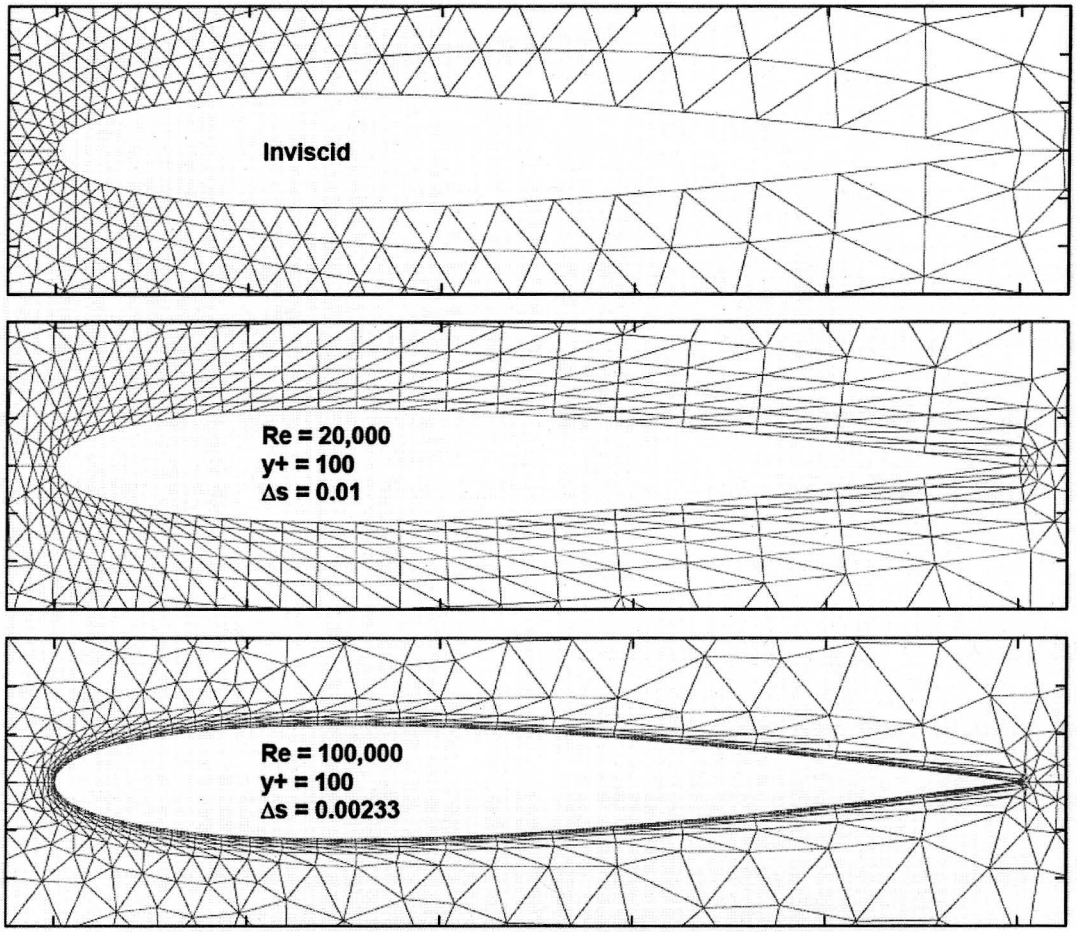


Figure 148 – NACA0012 mesh comparison at varying Reynolds numbers.

Chapter 8 – Conclusions

There are many ways that a region that is to be analyzed might be broken up and discretized. This dissertation focused on breaking up a region of interest using unstructured grid methodology (as opposed structured grid methodology, where the breakdown of the domain reflects some type of consistent geometrical regularity). Although the focus was on unstructured meshes, many of the algorithmic and mathematical techniques found herein owe their origins to structured methods.

Of particular note are the Winslow elliptic smoothing equations. These equations were first applied to structured meshes as far back as the 1960s and, more recently, have started to become powerful tools for smoothing unstructured meshes as well. However, unlike for structured meshing, no implicit computational space exists for unstructured meshes and the computational space must be explicitly defined.

Because the unstructured mesh connectivity needed to be explicitly defined and the valence count (the number of connected nodes or neighbors) varies throughout a domain, no theory exists to generate a general overarching computational space that reflects a consistent geometrical regularity as was done with structured meshes. This difficulty was alleviated when it was determined that it was not necessary for the entire computational mesh to be constructed as an overarching system of nodes and elements but rather, each node in computational space could be treated as an individual virtual control volume and coupled only though the coordinates in physical space, which were treated as free dependent variables.

Using these loosely coupled virtual control volumes as the unstructured computational space, the Winslow elliptic smoothing equations allow interior mesh points to be moved and smoothed to conform to a moving surface. Conventionally, the unstructured computational space for each of the nodes formulated in this manner is surrounded by neighbors using equal angles and equal edge-lengths and it is necessary that the central node in each virtual control volume be fully surrounded by neighboring nodes for the computational template to be complete. This makes the Winslow equations ideal for smoothing non-boundary nodes in inviscid meshes because the equations drive the mesh elements to display an isotropic behavior.

This also, however, makes the conventional Winslow methodology unsuitable to two situations that are common in fluid mechanics. The first is the situation where the nodes on the boundaries need to move to accommodate the movement of the interior mesh nodes, where keeping the boundary nodes static will result in elements that exhibit a sufficient amount of skew that the numerical solutions become unstable or unreliable. The second situation is one where the viscous properties of the flow are important. In order to capture the viscous boundary layer in an efficient manner it is necessary that the mesh elements near the viscous surface not be isotropic but rather have high aspect ratios in order to capture the gradients in the flow where the flow field variables are changing very rapidly normal to the surface but much more mildly in the direction parallel to the surface. Overcoming these two limitations to the Winslow elliptic smoothing equations applied to unstructured meshes were the two major contributions that are presented in this dissertation.

The first of these two major areas of study that was examined was the case where the Winslow equations could be applied to the boundaries of an unstructured mesh. Traditionally, boundary points were always held static in their original position and only interior mesh points were permitted to move. This however caused difficulties if a surface exhibited a significant amount of movement or if a surface that was very close to another surface was moved while the other surface remained static. Both of these situations tended to result in significant skewness in the mesh because, since the connectivity of the mesh was not changing, the mesh elements needed to stretch in order conform to the new surface positions.

The problem with applying the Winslow equations to a point on a surface is that, the virtual control volume that was being utilized for the computational space for each of the nodes needed to have a complete unbroken stencil of neighboring nodes in order for the Winslow equations to be properly solved. For a node on a surface, an unbroken stencil did not exist. The template was completed by utilizing ghost points outside the proper physical mesh domain and several methods of placing the ghost points were explored. After significant exploration into the placement of ghost points, it was determined that the best results occurred by implementing a reflection technique. This technique used a reflection matrix, which took into account the coordinate positions of all of the neighboring nodes within the domain, and reflected this composite neighbor location across a line tangent to the boundary at the location of the boundary node for which a dynamic floating position was desired.

The floating-surface Winslow algorithm was tested on several geometries. The first case was a flat plate in a domain where the outer boundary was relatively close to the flat

plate surface. The flat plate was taken through a full range of movements that included translation, rotation and even the deformation of the surface. Valid meshes were generated for all cases. Several types of boundaries, on which the surface points were allowed to float, were examined as well. The simplest type (a flat boundary) was explored first and then progressively more complicated boundaries were explored using analytically defined boundaries and finally explicitly defined arbitrary boundaries.

Once the Winslow floating-surface algorithms and logic were working correctly on a single boundary using a flat plate, they were extended to be able to handle moving points on multiple boundaries. To test this, a mesh was generated that had a structure similar to the flat-plate mesh but, instead of a flat plate, the inner surface was a NACA0012 airfoil. Just as with the flat plate tests. The tests involving a NACA0012 airfoil and multiple floating-node boundaries also generated valid unstructured smoothed meshes.

The final case that was examined to explore the effects of applying the Winslow equations to the boundaries was to apply Winslow elliptic smoothing to a multi-element airfoil. The airfoil employed for this purpose was the 30P30N multi-element high-lift airfoil. The 30P30N is a three-element airfoil consisting of a central airfoil section with a flap section in the rear and a slat section in the front. The 30P30N airfoil was pitched 30 degrees and the surrounding mesh was smoothed using the Winslow equations to adapt the flowfield mesh to the new airfoil surface position. The mesh was smoothed using both a static outer boundary and a floating-point outer boundary. Good results were generated and the smoothed mesh was tested further by using it to generate a transonic flow solution.

Because the goal of analyzing the 30P30N airfoil was to demonstrate the power of the Winslow Equations applied to a boundary, it made sense to look at how two surfaces that were close together behaved if one of the surfaces was moved relative to the other. For this, the slat at the front of the airfoil was rotated from its original extended (high-lift) position to a retracted position. The mesh was smoothed at each of the two slat rotation steps. The first case that was examined was using static nodes on all of the airfoil boundaries. When this was done, the mesh became highly skewed in the region between the slat and the central airfoil position. The next case that was examined involved letting the boundary points on the surface of the central airfoil section float as the slat was rotated. This gave a much better mesh near the leading edge of the central airfoil section.

The next area where a significant contribution was made was in the application of the Winslow elliptic smoothing equations to a viscous region of a mesh. Prior to this research, the computational space for a given node was constructed by placing each of the connected nodes on a unit circle such that both angles and edge-lengths are all equal for a given computational space control volume. By implementing the computational space in this way, the Winslow equations smooth the physical mesh in such a way that the modified (i.e., smoothed) mesh is highly isotropic and thus ideally suited for an inviscid flow. The nature of a viscous mesh near a no-slip wall, however, is highly anisotropic and is in direct opposition to the goals of the unmodified Winslow equations utilizing equal angles and equal edge-lengths in computational space.

Several methods were tested to make the Winslow equations applicable to a viscous region. One of these methods was to use a hybrid computational space where the virtual control volumes for the nodes in the inviscid region were isotropic and the virtual control

volumes in the viscous regions were based on the original mesh. Another method that was explored at length was to modify the computational space using Riemannian metric tensors to make the characteristics of the computational space more similar to the characteristics of the physical space. Both of these techniques resulted in some success but also had issues such as skewed (though still valid) elements near sharp corners and loss of orthogonality in viscous layers of a mesh.

The final, and most successful, methodology that was developed and explored to apply the Winslow equations to a viscous region of an unstructured mesh was to use an iteratively adapted computational space for each of the nodes in the viscous region. This technique allowed a mesh with an amenable connectivity structure to be smoothed in such a way that it could match a desired viscous profile based on an initial off-body spacing and geometric progression of the viscous layers.

The iteratively adaptive Winslow algorithm was first tested on a rough airfoil and then, to further explore the Winslow equations using iteratively adapted offset computational space, the NACA0012 airfoil was employed. The viscous region surrounding the NACA0012 airfoil was specified to have five viscous layers and an off-body spacing and geometric progression factor were also specified (these values are specified as inputs in a viscous parameter file read in by the code). Once all of the viscous region parameters were specified, the iteratively adaptive Winslow smoothing algorithm was applied to the inviscid NACA0012 mesh and the resulting viscous region of the mesh was generated exactly as would be required to analyze the viscous region of a flow.

The Winslow elliptic smoothing equations using an iteratively adaptive computational space algorithm were tested on situations where an airfoil (the NACA0012) was moved around within the domain. The airfoil was rotated, translated and then both rotated and translated simultaneously. In any of the cases, the mesh can start out as viscous, inviscid or even an unviable mesh with grid crossing and negative volumes. The Winslow equations will cause the mesh to conform to the new position and orientation of the airfoil and the offset computational space will maintain the viscous layer regardless of orientation.

The next area that was explored was to look at how the equations behaved on a deforming airfoil surface, which has many practical applications to parametric design where it may be required that many designs be examined but manually generating meshes for each design would be prohibitively expensive. Again, a proper viscous mesh was generated.

A final application of the iteratively adaptive Winslow smoothing algorithm was a case where the required off-body spacing was changing. This is something that is guaranteed to be an issue in any kind of testing environment because it will always be necessary to explore properties at different Mach numbers and atmospheric conditions. As these conditions change, the required off-body spacing changes as well. It was shown that the iteratively adaptive Winslow smoothing algorithm has the ability to quickly and efficiently change the off-body spacing or geometric progression of the viscous layers to accommodate changes to the flow characteristics.

The research and results described above significantly expand the possible role of the Winslow elliptic smoothing equations as related to unstructured mesh generation and smoothing. The explorations and developments that were made herein make the equations applicable to many common situations in fluid mechanics that they, previously, would not have been well suited.

For the results to truly impact applied practical fluid mechanics, they need to be expanded from two dimensions to three dimensions. This process is already underway to some degree and it is the intent of the author to pursue this further in post-doctoral work.

Bibliography

Bibliography

1. **Anderson, John D.** *Computational Fluid Dynamics, The Basics with Applications.* New York, NY : MGraw-Hill, Inc., 1995. ISBN 0-07-001685-2.

2. **Carpenter, James G.** Time Accurate Unstructured Grid Adaption in Two and Three Dimensions. *A Dissertation Submitted to the Graduate Faculty of North Carolina State University.* Raleigh, NC : North Carolina State University, 2007.

3. **Cavallo, Peter A and Boker, Timothy J.** Efficient Delaunay-Based Solution Adaptation for Three-Dimensional Unstructured Meshes. Dublin, PA : American Institue of Aeronautics and Astronautics, 2000. AIAA-2000-0809.

4. **Winslow, Alan M.** Numerical Solution of the Quasilinear Poisson Equations in a Nonuniform Triangle Mesh. Worcester, MA : Journal of Computational Physics, 1967. Vol. 2, 149-172.

5. **Sahasrabudhe, Mandar.** Unstructured Mesh Generation and Manipulation Based on Elliptic Smoothing and Optimization. *A Dissertation Presented for the Doctorate of Philosophy Degree.* Chattanooga, TN : University of Tennessee at Chattanooga, 2008.

6. **Karman, Steve and Sahasrabudy, Mandar.** Unstructured Elliptic Smoothing Revisited. *47th Aerospace Sciences Meeting and Exhibit.* Orlando, FL : American Institute of Aeronautics and Astronautics, 2009. AIAA-2009-1362.

7. **Stewart, James.** *Calculus, Third Edition.* Pacific Grove, CA : Brooks/Cole, 1995. ISBN 0-534-21798-2.

8. **White, Frank M.** *Viscous Fluid Flow.* Boston, MA : McGraw Hill, 1991. ISBN 0-07-069712-4.

9. **Knupp, Patrick M.** Winslow Smoothing on Two-Dimensional Unstructured Meshes. *Engineering With Computers.* Albuquerque, NM : Springer, 1999. Vol. 15.

10. **Weisstein, Eric W.** "Laplace's Equation." From MathWorld - A Wolfram Web Resource. *http://mathworld.wolfram.com/LaplacesEquation.html.* 2010.

11. **Karman, Steve, Anderson, Kyle and Sahasrabudhe, Mandar.** Mesh Generation Using Unstructrured Computational Meshes and Elliptic Partial Differential Equation Smoothing. Reno, NV : American Institute of Aeronautics and Astronautics, 2005. AIAA-2005-0923.

12. www.pointwise.com. Fort Worth, TX : Pointwise, Inc., 2010.

13. **Smith, Richard W. and Wright, Jeffrey A.** A Classical Elasticity-Based Mesh Update Method for Moving and Deforming Meshes. Orlando, FL : American Institute of Aeronautics and Astronautics, 2010. AIAA-2010-164.

14. **Koomulil, Roy P.** Development of Modules/Components for Fluid-Structure Interaction Framework. Birmingham, AL : University of Alabama at Birmingham, 2008. Contract No. GS04T01BFC0060.

15. Kestrel User's Guide. Eglin AFB, FL : CREATE-AV/Kestrel.

16. **Poole, David.** *Linear Algebra, A Modern Introduction.* Belmont, CA : Thomson Brooks Cole, 2006. ISBN 0-534-99845-3.

17. **Babuska, I. and Aziz, A.K.** On the Angle Condition in the Finite Element Method. *SIAM J. Numer. Anal.* Philadelphia, PA : Society for Industrial and Applied Mathematics, Apr. 1976. Vol. 13. pp 214-226.

18. **Jones, William T.** Gird Spacing Caculator. *http://geolab.larc.nasa.gov/APPS/YPlus/.* Langley, VA : NASA Langley Research Center, 2010.

19. **Rowland, Todd.** "Elliptic Partial Defferential Equation." From MathWorld - A Wolfram Web Resource. *http://mathworld.wolfram.com/EllipticPartialDifferentialEquation.html.* 2010.

20. **Thomas, P.D. and Middlecoff, J.F.** Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations. *AIAA Journal.* Reston, VA : American Institute of Aeronautics and Astronautics, 1979. Vol. 18.

Appendices

Appendix A – Metric Relationships and the Jacobian

Partial derivatives with respect to the two dimensional computational space coordinates (ξ,η) can be written as: $\frac{\partial}{\partial\xi}=\left(\frac{\partial \mathrm{x}}{\partial\xi}\right)\left(\frac{\partial}{\partial \mathrm{x}}\right)+\left(\frac{\partial \mathrm{y}}{\partial\xi}\right)\left(\frac{\partial}{\partial \mathrm{y}}\right)$ and $\frac{\partial}{\partial\eta}=\left(\frac{\partial \mathrm{x}}{\partial\eta}\right)\left(\frac{\partial}{\partial \mathrm{x}}\right)+\left(\frac{\partial \mathrm{y}}{\partial\eta}\right)\left(\frac{\partial}{\partial \mathrm{y}}\right)$. The coefficients in these equations are referred to as the inverse grid metrics. Using a condensed form for the derivatives (i.e., $\frac{\partial \mathrm{x}}{\partial\xi}=\xi_{\mathrm{x}}$ etc.) the above relationships can be written as the following system:

$$\begin{bmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \mathrm{x}} \\ \frac{\partial}{\partial \mathrm{y}} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial\xi} \\ \frac{\partial}{\partial\eta} \end{bmatrix} \tag{A.1}$$

Cramer's rule allows us to construct the relationship between the grid metrics $(\xi_{\mathrm{x}},\xi_{\mathrm{y}},\eta_{\mathrm{x}},\eta_{\mathrm{y}})$ and the inverse grid metrics $(\mathrm{x}_{\xi},\mathrm{x}_{\eta},\mathrm{y}_{\xi},\mathrm{y}_{\eta})$. Using Cramer's rule, new relationships for the partial derivatives in physical space shown as equation (A.2) and equation (A.3) are constructed.

$$\frac{\partial}{\partial \mathrm{x}}=\frac{\begin{vmatrix} \frac{\partial}{\partial\xi} & \mathrm{y}_{\xi} \\ \frac{\partial}{\partial\eta} & \mathrm{y}_{\eta} \end{vmatrix}}{\begin{vmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{vmatrix}} \tag{A.2}$$

$$\frac{\partial}{\partial y} = \frac{\begin{vmatrix} x_\xi & \frac{\partial}{\partial \xi} \\ x_\eta & \frac{\partial}{\partial \eta} \end{vmatrix}}{\begin{vmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{vmatrix}} \tag{A.3}$$

In the equations above, the value in the denominator is the Jacobian, which is defined as:

$$J = \begin{vmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{vmatrix} = x_\xi y_\eta - x_\eta y_\xi \tag{A.4}$$

Using the definition of the Jacobian, equations (A.2) and (A.3) can now be written as follows:

$$\frac{\partial}{\partial x} = \frac{1}{J}\left(y_\eta \frac{\partial}{\partial \xi} - y_\xi \frac{\partial}{\partial \eta} \right) \tag{A.5}$$

$$\frac{\partial}{\partial y} = \frac{1}{J}\left(x_\xi \frac{\partial}{\partial \eta} - x_\eta \frac{\partial}{\partial \xi} \right) \tag{A.6}$$

Comparing equations (A.5) and (A.6) to the chain-ruled equations for partial derivatives in physical space (i.e. $\frac{\partial}{\partial x} = \xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta}$ and $\frac{\partial}{\partial y} = \xi_y \frac{\partial}{\partial \xi} + \eta_y \frac{\partial}{\partial \eta}$) reveals the following relationships:

$$\xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta} = \frac{1}{J}\left(y_\eta \frac{\partial}{\partial \xi} - y_\xi \frac{\partial}{\partial \eta} \right) \tag{A.7}$$

$$\xi_y \frac{\partial}{\partial \xi} + \eta_y \frac{\partial}{\partial \eta} = \frac{1}{J}\left(x_\xi \frac{\partial}{\partial \eta} - x_\eta \frac{\partial}{\partial \xi} \right) \tag{A.8}$$

Combining equations (A.7) and (A.8) into matrix form (shown as equation (A.9)) allows for the relationships between the grid metrics and the inverse grid metrics to be easily seen via inspection.

$$\begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} y_\eta & -y_\xi \\ -x_\eta & x_\xi \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \tag{A.9}$$

$$\xi_x = \frac{y_\eta}{J}$$

$$\eta_x = -\frac{y_\xi}{J}$$

$$\xi_y = -\frac{x_\eta}{J}$$

$$\eta_y = \frac{x_\xi}{J}$$

$$J = x_\xi y_\eta - x_\eta y_\xi$$

It is also necessary to define the inverse of this procedure. The partial derivatives of the physical variables, which are $\frac{\partial}{\partial x} = \left(\frac{\partial \xi}{\partial x}\right)\left(\frac{\partial}{\partial \xi}\right) + \left(\frac{\partial \eta}{\partial x}\right)\left(\frac{\partial}{\partial \eta}\right)$ and $\frac{\partial}{\partial y} = \left(\frac{\partial \xi}{\partial y}\right)\left(\frac{\partial}{\partial \xi}\right) + \left(\frac{\partial \eta}{\partial y}\right)\left(\frac{\partial}{\partial \eta}\right)$ can be written as the following system:

$$\begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{A.10}$$

Applying Cramer's rule to equation (A.10) gives the equations for the partial derivatives in computational space shown as equation (A.11) and equation (A.12).

$$\frac{\partial}{\partial \xi} = \frac{\begin{vmatrix} \frac{\partial}{\partial x} & \eta_x \\ \frac{\partial}{\partial y} & \eta_y \end{vmatrix}}{\begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix}} \tag{A.11}$$

$$\frac{\partial}{\partial \eta} = \frac{\begin{vmatrix} \xi_x & \frac{\partial}{\partial x} \\ \xi_y & \frac{\partial}{\partial y} \end{vmatrix}}{\begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix}} \tag{A.12}$$

The value in the denominators of (A.11) and (A.12) is now the inverse-Jacobian, which will be denoted with a script J ($\mathscr{J}$):

$$\mathscr{J} = \begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix} = \xi_x \eta_y - \xi_y \eta_x \tag{A.13}$$

The equations for the partial derivatives in computational space above can now be written as follows:

$$\frac{\partial}{\partial \xi} = \frac{1}{\mathscr{J}} \left(\eta_y \frac{\partial}{\partial x} - \eta_x \frac{\partial}{\partial y} \right) \tag{A.14}$$

$$\frac{\partial}{\partial \eta} = \frac{1}{\mathscr{J}} \left(\xi_x \frac{\partial}{\partial y} - \xi_y \frac{\partial}{\partial x} \right) \tag{A.15}$$

Comparing equations (A.14) and (A.15) to the equations for partial derivatives in computational space, i.e. $\frac{\partial}{\partial \xi} = \left(\frac{\partial x}{\partial \xi} \right) \left(\frac{\partial}{\partial x} \right) + \left(\frac{\partial y}{\partial \xi} \right) \left(\frac{\partial}{\partial y} \right)$ and

$\frac{\partial}{\partial \eta} = \left(\frac{\partial x}{\partial \eta} \right) \left(\frac{\partial}{\partial x} \right) + \left(\frac{\partial y}{\partial \eta} \right) \left(\frac{\partial}{\partial y} \right)$ reveals the following relationships:

$$x_\xi\left(\frac{\partial}{\partial x}\right)+y_\xi\left(\frac{\partial}{\partial y}\right)=\frac{1}{\mathcal{J}}\left(\eta_y\frac{\partial}{\partial x}-\eta_x\frac{\partial}{\partial y}\right) \tag{A.16}$$

$$x_\eta\left(\frac{\partial}{\partial x}\right)+y_\eta\left(\frac{\partial}{\partial y}\right)=\frac{1}{\mathcal{J}}\left(\xi_x\frac{\partial}{\partial y}-\xi_y\frac{\partial}{\partial x}\right) \tag{A.17}$$

Combining equations (A.16) and (A.17) into matrix form (shown as equation (A.18)) allows for the following relationships between the grid metrics and the inverse grid metrics (this time utilizing the inverse Jacobian) to again be easily seen via inspection.

$$\begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix}\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}=\frac{1}{\mathcal{J}}\begin{bmatrix} \eta_y & -\eta_x \\ -\xi_y & \xi_x \end{bmatrix}\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{A.18}$$

$$x_\xi=\frac{\eta_y}{\mathcal{J}}$$

$$y_\xi=-\frac{\eta_x}{\mathcal{J}}$$

$$x_\eta=-\frac{\xi_y}{\mathcal{J}}$$

$$y_\eta=\frac{\xi_x}{\mathcal{J}}$$

$$\mathcal{J}=\xi_x\eta_y-\xi_y\eta_x$$

Appendix B – Derivation of the Winslow Equations

Elliptic Smoothing can be used to generate and improve meshes in physical space. The equations that are used for the smoothing process are derived from the relationship that exists between a mesh in physical space and computational space. The Winslow equations were originally applied to structured meshes. A very simple structured mesh, in both physical and computational space, can be seen below.

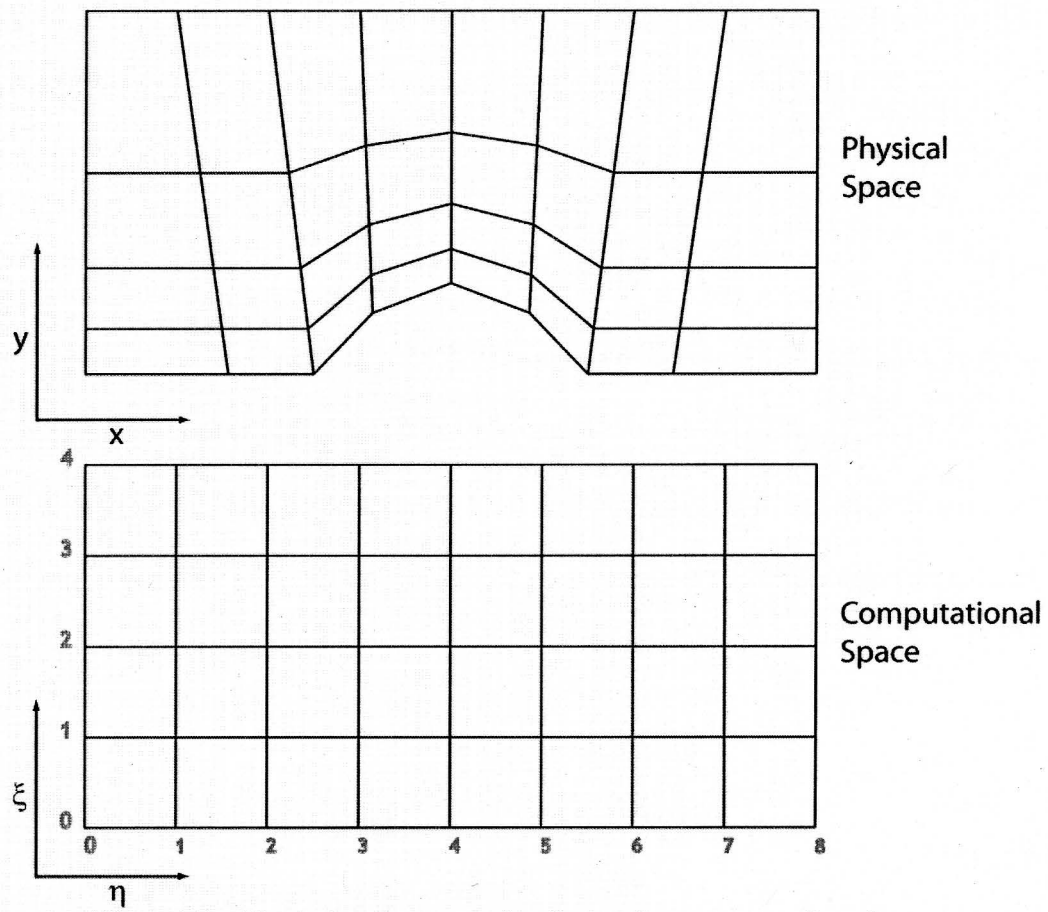


Figure 149 – Physical and computational space for a structured mesh.

The coordinates in two dimensional physical space are represented as (x,y) and the coordinates in two-dimensional computational space are represented as (ξ,η). The mapping between these two domains is defined with the transformations shown as equations (B.1) and (B.2) [11].

$$\begin{aligned} \xi &= \xi(x,y) \\ \eta &= \eta(x,y) \end{aligned} \tag{B.1}$$

$$\begin{aligned} x &= x(\xi,\eta) \\ y &= y(\xi,\eta) \end{aligned} \tag{B.2}$$

Elliptic smoothing utilizes elliptic partial differential equations in the form of Poisson's equation (B.3) or the homogeneous Laplace's equation (B.4) [19].

$$\nabla^2\phi = f(x,y) \tag{B.3}$$

$$\nabla^2\phi = 0 \tag{B.4}$$

The elliptic smoothing equations are constructed by having a Laplacian operator with respect to physical coordinates act on the computational coordinates. The result is then set equal to zero if the Laplace form is desired and set equal to a forcing function if the Poisson form is desired. The elliptic smoothing equations are conventionally expressed in one of the two forms shown below:

P-Q Form:

$$\begin{aligned} \nabla^2\xi &= \frac{\partial^2\xi}{\partial x^2} + \frac{\partial^2\xi}{\partial y^2} = P \\ \nabla^2\eta &= \frac{\partial^2\eta}{\partial x^2} + \frac{\partial^2\eta}{\partial y^2} = Q \end{aligned} \tag{B.5}$$

Φ–Ψ Form:

$$\nabla^2\xi = \frac{\partial^2\xi}{\partial x^2} + \frac{\partial^2\xi}{\partial y^2} = (\nabla\xi \bullet \nabla\xi)\Phi$$
$$\nabla^2\eta = \frac{\partial^2\eta}{\partial x^2} + \frac{\partial^2\eta}{\partial y^2} = (\nabla\eta \bullet \nabla\eta)\Psi \qquad \text{(B.6)}$$

These equations are currently cast in physical space and represent the smooth variation of the computational coordinates in physical space. When the forcing functions are set equal to zero, the equations convert to Laplace's equations, which have the property that the average value over a spherical surface is equal to the value at the center of the sphere. This has the effect of causing a node to move to the centroid of the nodes to which it is connected. Solutions to Laplace's equations also satisfy the maximum/minimum principle, which states that the dependent variables on the interior of the domain are bounded by the values on the boundary of the domain. This ensures that the interior grid lines do not cross if the boundaries of the domain are chosen to be constant ξ and constant η gridlines [11] as seen in Figure 149.

It was noted that the equations in the form shown in (B.5) and (B.6) represent a smooth variation of the computational coordinates in physical space. However, the computational coordinates (ξ and η) are generally known and it is the values of the physical coordinates (x and y) that are desired. For the equations to be of practical value, they are transformed to computational space (i.e. the partial derivatives will be with respect to computational space). The final transformed equations are referred to as the Winslow Equations, the derivation of which is shown below.

The chain rule for differentiation in multiple variables states that if $f = f(\xi,\eta)$ and $\xi = \xi(x,y)$ and $\eta = \eta(x,y)$ then the derivative for f is $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial f}{\partial \eta}\frac{\partial \eta}{\partial x}$. This can be rewritten in operator form as $\frac{\partial}{\partial x}(f) = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi}(f) + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta}(f)$.

Now, consider the case where $f = \frac{\partial \xi}{\partial x}$. This will give the equation for the second derivative:

$$\frac{\partial}{\partial x}\left(\frac{\partial \xi}{\partial x}\right) = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi}\left(\frac{\partial \xi}{\partial x}\right) + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta}\left(\frac{\partial \xi}{\partial x}\right)$$

Or, in condensed notation:

$$\xi_{xx} = \xi_x \xi_{x\xi} + \eta_x \xi_{x\eta}$$

$$\xi_{xx} = \xi_x \frac{\partial}{\partial \xi}(\xi_x) + \eta_x \frac{\partial}{\partial \eta}(\xi_x)$$

$$\xi_{xx} = J^{-1} y_\eta \frac{\partial}{\partial \xi}\left(J^{-1} y_\eta\right)\xi_x - J^{-1} y_\xi \frac{\partial}{\partial \eta}\left(J^{-1} y_\eta\right)$$

$$\xi_{xx} = J^{-1} y_\eta \left[-J^{-2} J_\xi y_\eta + J^{-1} y_{\eta\xi}\right] - J^{-1} y_\xi \left[-J^{-2} J_\eta y_\eta + J^{-1} y_{\eta\eta}\right]$$

$$\xi_{xx} = \frac{y_\eta}{J^3}\left(J y_{\eta\xi} - J_\xi y_\eta\right) - \frac{y_\xi}{J^3}\left(J y_{\eta\eta} - J_\eta y_\eta\right) \qquad \text{(B.7)}$$

Likewise:

$$\frac{\partial}{\partial x}\left(\frac{\partial \eta}{\partial x}\right)=\frac{\partial^2 \eta}{\partial x \partial \xi}\frac{\partial \xi}{\partial x}+\frac{\partial^2 \eta}{\partial x \partial \eta}\frac{\partial \eta}{\partial x}=\eta_{\xi x}\xi_x+\eta_{\eta x}\eta_x$$

$$\eta_{xx}=\xi_x\frac{\partial}{\partial \xi}(\eta_x)+\eta_x\frac{\partial}{\partial \eta}(\eta_x)$$

$$\eta_{xx}=J^{-1}y_\eta\frac{\partial}{\partial \xi}\left(-J^{-1}y_\xi\right)\xi_x-J^{-1}y_\xi\frac{\partial}{\partial \eta}\left(-J^{-1}y_\xi\right)$$

$$\eta_{xx}=J^{-1}y_\eta\left[J^{-2}J_\xi y_\xi-J^{-1}y_{\xi\xi}\right]-J^{-1}y_\xi\left[J^{-2}J_\eta y_\eta-J^{-1}y_{\xi\eta}\right]$$

$$\eta_{xx}=\frac{y_\eta}{J^3}\left(-Jy_{\xi\xi}+J_\xi y_\eta\right)-\frac{y_\xi}{J^3}\left(-Jy_{\xi\eta}+J_\eta y_\eta\right) \qquad \text{(B.8)}$$

The derivatives of the Jacobian are:

$$J_\xi=x_\xi y_{\eta\xi}+y_\eta x_{\xi\xi}-x_\eta y_{\xi\xi}-y_\xi x_{\eta\xi}$$
$$J_n=x_\xi y_{\eta\eta}+y_\eta x_{\eta\xi}-x_\eta y_{\xi\eta}-y_\xi x_{\eta\eta}$$

Substituting the values of the Jacobian and the Jacobian derivatives into (B.7) gives:

$$\xi_{xx}=\frac{y_\eta}{J^3}\left(x_\xi y_\eta y_{\eta\xi}-x_\eta y_\xi y_{\eta\xi}-x_\xi y_{\eta\xi}y_\eta-y_\eta{}^2x_{\xi\xi}+x_\eta y_{\xi\xi}y_\eta+y_\xi x_{\eta\xi}y_\eta\right)$$
$$-\frac{y_\xi}{J^3}\left(x_\xi y_\eta y_{\eta\eta}-x_\eta y_\xi y_{\eta\eta}-x_\xi y_{\eta\eta}y_\eta-y_\eta x_{\eta\xi}y_\eta+x_\eta y_{\xi\eta}y_\eta+y_\xi x_{\eta\eta}y_\eta\right)$$

Rearranging and combining terms gives:

$$\xi_{xx}=\frac{1}{J^3}\left[x_\eta\left(y_\eta{}^2y_{\xi\xi}+y_\xi{}^2y_{\eta\eta}-2y_\xi y_\eta y_{\xi\eta}\right)-y_\eta\left(y_\eta{}^2x_{\xi\xi}+y_\xi{}^2x_{\eta\eta}-2y_\xi y_\eta x_{\eta\xi}\right)\right]$$

Following the same steps for the other terms gives:

$$\xi_{yy}=\frac{1}{J^3}\left[-y_\eta\left(x_\eta{}^2x_{\xi\xi}+x_\xi{}^2x_{\eta\eta}-2x_\xi x_\eta x_{\xi\eta}\right)+x_\eta\left(x_\eta{}^2y_{\xi\xi}+x_\xi{}^2y_{\eta\eta}-2x_\xi x_\eta y_{\eta\xi}\right)\right]$$

$$\eta_{xx}=-\frac{1}{J^3}\left[x_\xi\left(y_\eta{}^2y_{\xi\xi}+y_\xi{}^2y_{\eta\eta}-2y_\xi y_\eta y_{\xi\eta}\right)-y_\xi\left(y_\eta{}^2x_{\xi\xi}+y_\xi{}^2x_{\eta\eta}-2y_\xi y_\eta x_{\eta\xi}\right)\right]$$

$$\eta_{yy} = -\frac{1}{J^3}\left[-y_\xi\left(x_\eta{}^2 x_{\xi\xi} + x_\xi{}^2 x_{\eta\eta} - 2x_\xi x_\eta x_{\xi\eta}\right) + x_\xi\left(x_\eta{}^2 y_{\xi\xi} + x_\xi{}^2 y_{\eta\eta} - 2x_\xi x_\eta y_{\eta\xi}\right)\right]$$

Recall that the Laplacian of the computational coordinates are:

$$\nabla^2\xi = \xi_{xx} + \xi_{yy}$$
$$\nabla^2\xi = \frac{1}{J^3}\left\{-y_\eta\left[\left(x_\eta{}^2 + y_\eta{}^2\right)x_{\xi\xi} - 2\left(x_\xi x_\eta + y_\xi y_\eta\right)x_{\xi\eta} + \left(x_\xi{}^2 + y_\xi{}^2\right)x_{\eta\eta}\right]\right.$$
$$\left. + x_\eta\left[\left(x_\eta{}^2 + y_\eta{}^2\right)y_{\xi\xi} - 2\left(x_\xi x_\eta + y_\xi y_\eta\right)y_{\xi\eta} + \left(x_\xi{}^2 + y_\xi{}^2\right)y_{\eta\eta}\right]\right\}$$

$$\nabla^2\eta = \eta_{xx} + \eta_{yy}$$
$$\nabla^2\eta = -\frac{1}{J^3}\left\{-y_\xi\left[\left(x_\eta{}^2 + y_\eta{}^2\right)x_{\xi\xi} - 2\left(x_\xi x_\eta + y_\xi y_\eta\right)x_{\xi\eta} + \left(x_\xi{}^2 + y_\xi{}^2\right)x_{\eta\eta}\right]\right.$$
$$\left. + x_\xi\left[\left(x_\eta{}^2 + y_\eta{}^2\right)y_{\xi\xi} - 2\left(x_\xi x_\eta + y_\xi y_\eta\right)y_{\xi\eta} + \left(x_\xi{}^2 + y_\xi{}^2\right)y_{\eta\eta}\right]\right\}$$

Breaking out the coefficients gives the Laplacians in computational coordinates in the following form:

$$\nabla^2\xi = \frac{1}{J^3}\left\{-y_\eta\left[\alpha x_{\xi\xi} - 2\beta x_{\eta\xi} + \gamma x_{\eta\eta}\right] + x_\eta\left[\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right]\right\}$$
$$\nabla^2\eta = -\frac{1}{J^3}\left\{-y_\xi\left[\alpha x_{\xi\xi} - 2\beta x_{\eta\xi} + \gamma x_{\eta\eta}\right] + x_\xi\left[\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right]\right\} \qquad \text{(B.9)}$$

where
$$\alpha = x_\eta{}^2 + y_\eta{}^2$$
$$\beta = x_\xi x_\eta + y_\xi y_\eta$$
$$\gamma = x_\xi{}^2 + y_\xi{}^2$$

Now we define an operator $G(\bullet)$ as:

$$G(\bullet) = \alpha\frac{\partial^2(\bullet)}{\partial\xi^2} - 2\beta\frac{\partial^2(\bullet)}{\partial\xi\partial\eta} + \gamma\frac{\partial^2(\bullet)}{\partial\eta^2} \qquad \text{(B.10)}$$

Using the operator G, equations (B.9) now become:

$$\nabla^2\xi = \frac{1}{J^3}\left(-y_\eta G(x) + x_\eta G(y)\right) = -\frac{1}{J^2}\left(\xi_x G(x) + \xi_y G(y)\right)$$
$$\nabla^2\eta = -\frac{1}{J^3}\left(-y_\xi G(x) + x_\xi G(y)\right) = \frac{1}{J^2}\left(\eta_x G(x) + \eta_y G(y)\right)$$

Define a vector $\vec{r} = \begin{bmatrix} x \\ y \end{bmatrix}$ and these become:

$$\nabla^2\xi = -\frac{1}{J^2}\left(\nabla\xi \cdot G(\vec{r})\right) \text{ and } \nabla^2\eta = -\frac{1}{J^2}\left(\nabla\eta \cdot G(\vec{r})\right)$$

Rearranging the above equations slightly gives:

$$\left(\xi_x G(x) + \xi_y G(y)\right) = -J^2\nabla^2\xi$$
$$\left(\eta_x G(x) + \eta_y G(y)\right) = -J^2\nabla^2\eta$$

In Matrix form this is:

$$\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = -J^2\begin{bmatrix} \nabla^2\xi \\ \nabla^2\eta \end{bmatrix}$$

Solving for G(x) and G(y) gives:

$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = -J^2\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}^{-1}\begin{bmatrix} \nabla^2\xi \\ \nabla^2\eta \end{bmatrix}$$
$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = \frac{-J^2}{\xi_x\eta_y - \xi_y\eta_x}\begin{bmatrix} \eta_y & -\xi_y \\ -\eta_x & \xi_x \end{bmatrix}\begin{bmatrix} \nabla^2\xi \\ \nabla^2\eta \end{bmatrix}$$

Using the inverse-Jacobian, derived in Appendix A and shown as (A.13), this becomes:

$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = \frac{-J^2}{\mathcal{J}}\begin{bmatrix} \eta_y\nabla^2\xi - \xi_y\nabla^2\eta \\ -\eta_x\nabla^2\xi + \xi_x\nabla^2\eta \end{bmatrix}$$

$$G(x) = -J^2\left[\frac{\eta_y}{\mathcal{J}}\nabla^2\xi - \frac{\xi_y}{\mathcal{J}}\nabla^2\eta\right] = -J^2\left[x_\xi\nabla^2\xi + x_\eta\nabla^2\eta\right] \qquad \text{(B.11)}$$

$$G(y) = -J^2\left[-\frac{\eta_x}{\mathcal{J}}\nabla^2\xi + \frac{\xi_x}{\mathcal{J}}\nabla^2\eta\right] = -J^2\left[y_\xi\nabla^2\xi + y_\eta\nabla^2\eta\right] \tag{B.12}$$

B.1 – Zero Forcing Function

The desired conditions for smooth interior points using the homogeneous Laplace's Equation is $\nabla^2\xi = 0$ and $\nabla^2\eta = 0$. If $\nabla^2\eta$ and $\nabla^2\xi$ are set to zero in equation (B.11) and equation (B.12) then it follows that G(x) = 0 and G(y) = 0. Using the definition for G(x) and G(y) shown in equation (B.10) gives the final form of the Winslow equations with no Forcing Function:

$$G(x) = 0$$

$$G(y) = 0$$

$$\begin{aligned} \alpha\frac{\partial^2 x}{\partial\xi^2} - 2\beta\frac{\partial^2 x}{\partial\xi\eta} + \gamma\frac{\partial^2 x}{\partial\eta^2} &= 0 \\ \alpha\frac{\partial^2 y}{\partial\xi^2} - 2\beta\frac{\partial^2 y}{\partial\xi\eta} + \gamma\frac{\partial^2 y}{\partial\eta^2} &= 0 \end{aligned} \tag{B.13}$$

where:

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

B.2 – Winslow Equations with Forcing Functions

Consider Equations (B.5) and (B.6) with the forcing functions (P and Q) intact (i.e. the nonhomogeneous Poisson's Equation form):

$$\nabla^2\xi = P(x,y)$$
$$\nabla^2\eta = Q(x,y)$$

The Laplacians of the computational coordinates can now again be related to G(x) and G(y) by examining equation (B.11) and equation (B.12). (Note that, at this point, no assumptions were made as to whether or not the equations were in the Laplace or Poisson form.)

$$G(x) = -J^2\left[x_\xi \nabla^2\xi + x_\eta \nabla^2\eta\right]$$
$$G(y) = -J^2\left[y_\xi \nabla^2\xi + y_\eta \nabla^2\eta\right]$$

Applying the forcing functions gives:

$$G(x) = -J^2\left[x_\xi P + x_\eta Q\right]$$
$$G(y) = -J^2\left[y_\xi P + y_\eta Q\right]$$

Thomas and Middlecoff [20] proposed the following form for the forcing functions:

$$\begin{aligned} P(x,y) &= \Phi(\xi,\eta)(\nabla\xi\bullet\nabla\xi) \\ Q(x,y) &= \Psi(\xi,\eta)(\nabla\eta\bullet\nabla\eta) \end{aligned} \tag{B.14}$$

Using the Thomas-Middlecoff form (a.k.a. the Φ - Ψ form) we get:

$$G(x) = -J^2\left[x_\xi \Phi(\nabla\xi\bullet\nabla\xi) + x_\eta \Psi(\nabla\eta\bullet\nabla\eta)\right]$$
$$G(y) = -J^2\left[y_\xi \Phi(\nabla\xi\bullet\nabla\xi) + y_\eta \Psi(\nabla\eta\bullet\nabla\eta)\right]$$

Employing the definition of the G operator from equation (B.10) results in:

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = -J^2\left[x_\xi \Phi(\nabla\xi\bullet\nabla\xi) + x_\eta \Psi(\nabla\eta\bullet\nabla\eta)\right]$$
$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = -J^2\left[y_\xi \Phi(\nabla\xi\bullet\nabla\xi) + y_\eta \Psi(\nabla\eta\bullet\nabla\eta)\right]$$

$$\begin{aligned} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} + J^2 x_\xi \Phi(\nabla\xi\bullet\nabla\xi) + J^2 x_\eta \Psi(\nabla\eta\bullet\nabla\eta) = 0 \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} + J^2 y_\xi \Phi(\nabla\xi\bullet\nabla\xi) + J^2 y_\eta \Psi(\nabla\eta\bullet\nabla\eta) = 0 \end{aligned} \tag{B.15}$$

Applying the dot product to the computational coordinates gives:

$$\nabla\xi\bullet\nabla\xi = \xi_x\xi_x + \xi_y\xi_y = \frac{y_\eta}{J}\frac{y_\eta}{J} + \frac{-x_\eta}{J}\frac{-x_\eta}{J} = \frac{1}{J^2}\left(y_\eta{}^2 + x_\eta{}^2\right)$$

$$\nabla\eta\bullet\nabla\eta = \eta_x\eta_x + \eta_y\eta_y = \frac{-y_\xi}{J}\frac{-y_\xi}{J} + \frac{x_\xi}{J}\frac{x_\xi}{J} = \frac{1}{J^2}\left(y_\xi{}^2 + x_\xi{}^2\right)$$

This is now plugged into equation (B.15).

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

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

Rearranging the equations above results in the final form of the Winslow Equations using forcing functions in the Thomas-Middlecoff form:

$$\alpha\left(x_{\xi\xi} + \Phi x_\xi\right) - 2\beta x_{\xi\eta} + \gamma\left(x_{\eta\eta} + \Psi x_\eta\right) = 0$$
$$\alpha\left(y_{\xi\xi} + \Phi y_\xi\right) - 2\beta y_{\xi\eta} + \gamma\left(y_{\eta\eta} + \Psi y_\eta\right) = 0 \qquad \text{(B.16)}$$

where, again:

$$\alpha = x_\eta{}^2 + y_\eta{}^2$$
$$\beta = x_\xi x_\eta + y_\xi y_\eta$$
$$\gamma = x_\xi{}^2 + y_\xi{}^2$$

Appendix C – Extension to 3D

C.1 – 2D vs. 3D

As mentioned in the background section, because much of the work performed for this dissertation was proof-of-concept level work, it made sense to test the theories and work out the details of the algorithms in a 2 dimensional environment where the results could be viewed more quickly and efficiently. When possible, concepts were explored in limited cases in 3 dimensions and in all cases it was the desire of the author to design the concepts and algorithms in such a way that they would be extensible to 3D in the future. It is the intent of the author to continue this research and fully extend the concepts to 3D in post-doctoral work. Some of the concepts that were explored in 3D are described in this section.

C.2 – Gradients in 3D

Consider a three dimensional domain broken up into an unstructured mesh comprised of tetrahedra similar to the one shown below in Figure 150. Each tetrahedron becomes a control volume and the gradient within the control volume (which is treated as constant throughout the control volume) can be found by manipulating the Divergence Theorem (equation (2.1)).

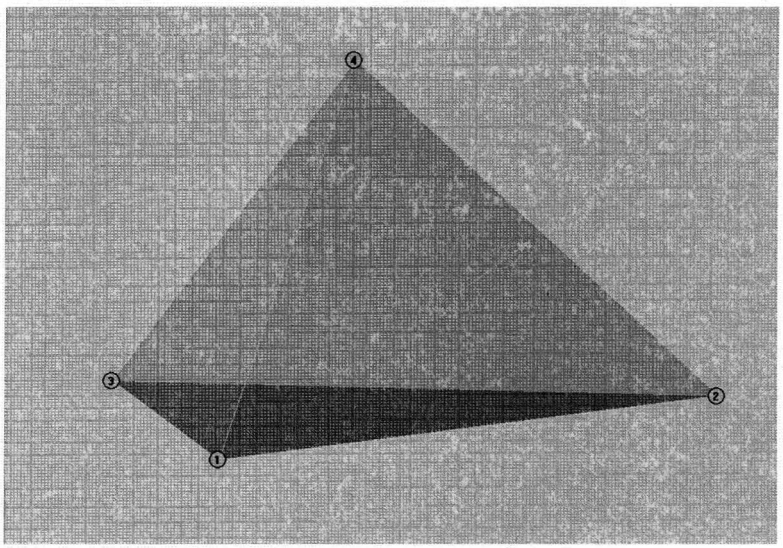


Figure 150 – Tetrahedral control volume.

In three dimensions, the gradient of a scalar can be written: $\nabla\phi = \begin{bmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \\ \frac{\partial\phi}{\partial z} \end{bmatrix}$

The x component of the gradient will be used for illustrative purposes. The other components are derived in similar fashion. The average gradient can be calculated as:

$$\overline{\frac{\partial\phi}{\partial x}} = \frac{1}{V}\iiint \frac{\partial\phi}{\partial x} dV \qquad (C.1)$$

The divergence of a vector $\left(\nabla \cdot \vec{F}\right)$ can be related to a single value of the gradient (i.e., the gradient in a particular coordinate direction) using the same logic that was described in the 2D gradient section (Section 2.2).

$$\vec{F} = \begin{bmatrix} \phi \\ 0 \\ 0 \end{bmatrix} \Rightarrow \nabla \cdot \vec{F} = \frac{\partial \phi}{\partial x} + 0 + 0 = \frac{\partial \phi}{\partial x}$$

$$\underbrace{\iiint \nabla \cdot \vec{F}\, dV = \oiint \vec{F} \cdot \vec{n} dS}_{\text{DivergenceThm}} = \oiint \begin{bmatrix} \phi \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} dS = \oiint \phi\, n_x\, dS$$

Using the logic described above, the average x-direction gradient in a control volume can be calculated as:

$$\overline{\frac{\partial \phi}{\partial x}} = \frac{1}{V} \iiint \frac{\partial \phi}{\partial x} dV = \frac{1}{V} \oiint \phi\, n_x\, dS$$

Dropping the overbar for simplicity with the understanding that the gradient is treated as constant within the control volume, the equation for the 3D gradient becomes:

$$\nabla \phi = \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{1}{V} \oiint \phi\, n_x\, dS \\ \frac{1}{V} \oiint \phi\, n_y\, dS \\ \frac{1}{V} \oiint \phi\, n_z\, dS \end{bmatrix} \tag{C.2}$$

The surface integral can be discretized by examining and summing the scalar for which the gradient is desired on each of the 4 faces of the tetrahedron:

$$\oiint \phi\, n_x\, dS \approx \sum_{i=1}^{4} \bar{\phi}_i \hat{n}_{xi} |\Gamma_i| \tag{C.3}$$

Where: $\overline{\phi_i}$ = the average value of the scalar over the face, this can be calculated as the average of the value at each of the 3 nodes that makes up the face or as the average of the value at the two cell centers on either side of the face.

$\hat{n}_{xi}$ = the x component of the surface unit-normal vector for face i

$|\Gamma_i|$ = the magnitude of the area of face i

The area of a triangle in 3D space can be calculated as the cross produce of two of its edges. A tetrahedron (tet) is defined to have a positive volume if the surface normals are outward-facing. The areas of each of the surfaces will be defined to be positive if the normal vectors are coming out of the tet as well. Look at the base of the tet shown in Figure 150 and re-shown in Figure 151 below. The numbering convention that will be used for the faces is that the face number will correspond to the node opposite it. The base (face 4) is formed by nodes 1, 2 and 3. The area of the face is calculated using equation (C.4).

$$\vec{\Gamma}_4 = \vec{u} \times \vec{v} \tag{C.4}$$

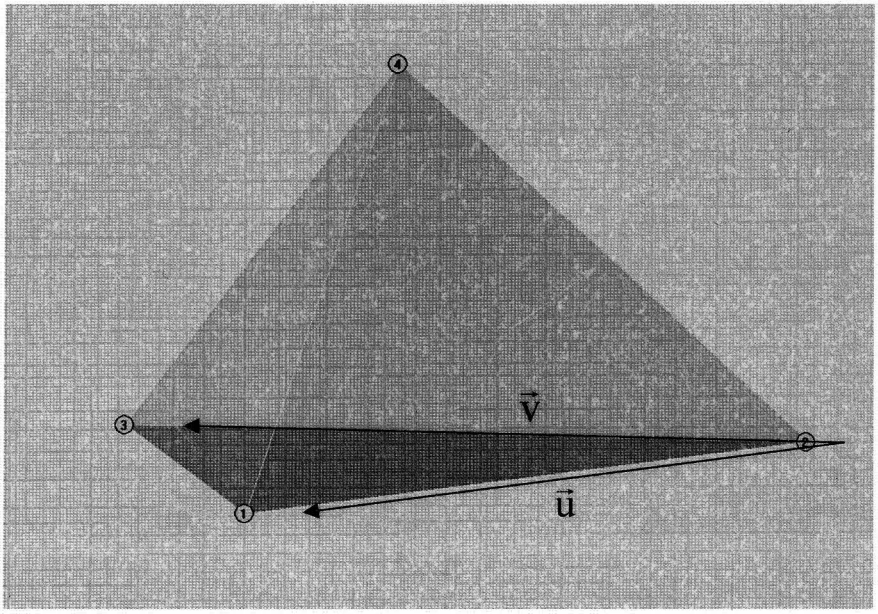


Figure 151 – Vectors for calculating the area of face 4.

The right hand rule will confirm that face 4 has a positive area (outward facing normal). The coordinates of the nodes are denoted using the following convention: Coordinates of node 1 are (x_1, y_1, z_1) etc.

$$\vec{\Gamma}_4 = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \end{vmatrix} \tag{C.5}$$

$$\vec{\Gamma}_4 = \begin{bmatrix} (y_1 - y_2)(z_3 - z_2) - (y_3 - y_2)(z_1 - z_2) \\ -(x_1 - x_2)(z_3 - z_2) + (x_3 - x_2)(z_1 - z_2) \\ (x_1 - x_2)(y_3 - y_2) - (x_3 - x_2)(y_1 - y_2) \end{bmatrix} = \begin{bmatrix} \Gamma_x \\ \Gamma_y \\ \Gamma_z \end{bmatrix}_4 \tag{C.6}$$

The magnitude of the area is $\left|\vec{\Gamma}\right| = \sqrt{\Gamma_x{}^2 + \Gamma_y{}^2 + \Gamma_y{}^2}$ and the area vector is in the same direction as the unit normal vector, which is defined by equation (C.7):

$$\vec{n} = \begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_y \end{bmatrix} = \begin{bmatrix} \Gamma_x \\ \Gamma_y \\ \Gamma_y \end{bmatrix} \frac{1}{\sqrt{\Gamma_x{}^2 + \Gamma_y{}^2 + \Gamma_y{}^2}} \tag{C.7}$$

For face i, this gives the following relationships:

$$\hat{n}_{xi}|\Gamma_i| = \Gamma_{xi}$$
$$\hat{n}_{yi}|\Gamma_i| = \Gamma_{yi}$$
$$\hat{n}_{zi}|\Gamma_i| = \Gamma_{zi}$$

These relationships can be used to simplify equation (C.3) as follows.

$$\iint \phi \vec{n}_x \, dS \approx \sum_{i=1}^{4} \bar{\phi}_i \hat{n}_{xi} |\Gamma_i| = \sum_{i=1}^{4} \bar{\phi}_i \Gamma_{xi}$$

And, using the differential form of the gradient equation seen in (C.2) we get

$$\frac{\partial \phi}{\partial x} = \frac{1}{V} \iint \phi \vec{n}_x \, dS \approx \frac{1}{V} \sum_{i=1}^{4} \bar{\phi}_i \Gamma_{xi}$$

Now the area components (using face 4) are calculated as:

$$\Gamma_{x4} = (y_1 - y_2)(z_3 - z_2) - (y_3 - y_2)(z_1 - z_2)$$
$$\Gamma_{y4} = -(x_1 - x_2)(z_3 - z_2) + (x_3 - x_2)(z_1 - z_2)$$
$$\Gamma_{z4} = (x_1 - x_2)(y_3 - y_2) - (x_3 - x_2)(y_1 - y_2)$$

The final form of the three dimensional gradient is seen below as equation (C.8).

$$\nabla \phi = \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{1}{V} \sum_{i=1}^{4} \phi_i \Gamma_{xi} \\ \frac{1}{V} \sum_{i=1}^{4} \phi_i \Gamma_{yi} \\ \frac{1}{V} \sum_{i=1}^{4} \phi_i \Gamma_{zi} \end{bmatrix} \qquad \text{(C.8)}$$

C.3 – Extending Reflection for Ghost Points to 3D

Placing a ghost point (P_G) requires that a coordinate location (in this case the location is the coordinates of P_A, which is the average of all interior nodes connected to a surface point, P_S) on the interior of the domain be reflected outside the region of the proper domain. The technique that is used to place P_G in two dimensions is described in Chapter 4. The following logic extends the technique to three dimensions

In order to extend the reflection technique for placing a ghost point in three dimensions, the Householder matrix can be employed. The Householder matrix (or reflector) is a transformation matrix that reflects a vector through the plane for which $\vec{n}$ is the unit normal. In the context of placing a ghost point, $\vec{n}$ would be the normal out of the outer boundary surface at the surface point P_S. The Householder matrix is defined as $H = I - 2\vec{n}\vec{n}^T$ and is applied to a general vector $\vec{a}$ as follows:

$$H\vec{a} = \left[I - 2\vec{n}\vec{n}^T\right]\vec{a} \tag{C.9}$$

Appling this to P_A, which is the coordinate vector of the average interior point, gives:

$$P_G = [H]P_A = \left[I - 2\vec{n}\vec{n}^T\right]P_A$$
$$P_G = P_A - 2\vec{n}\vec{n}^T P_A$$

Noting that $\vec{n}^T P_A$ is a scalar, the equation is rearranged to get:

$$P_G = [H]P_A = \left[I - 2\vec{n}\vec{n}^T\right]P_A$$
$$P_G = P_A - 2\left(\vec{n}^T P_A\right)\vec{n}$$

A graphical interpretation of a Householder reflector working on the vector $\vec{a}$ is shown on Figure 152 below.

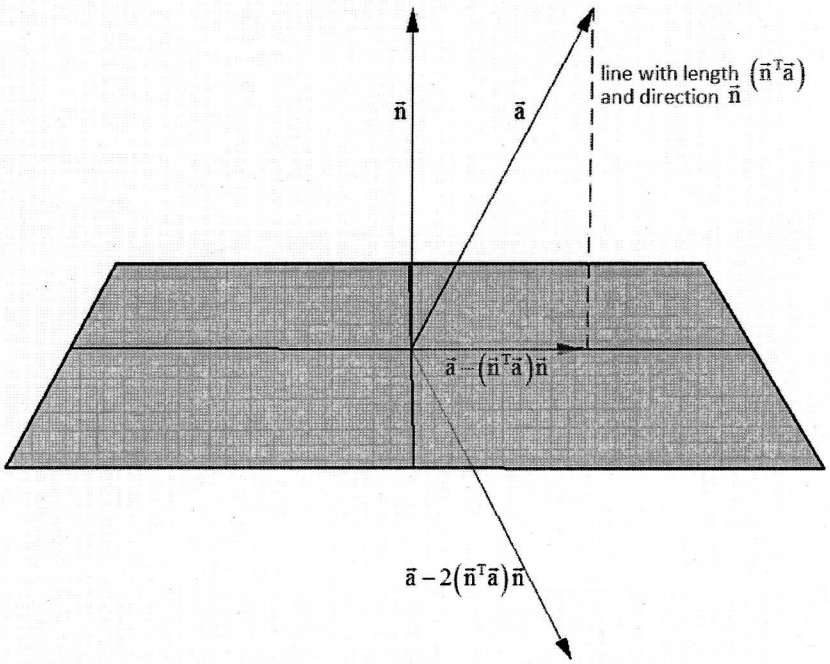


Figure 152 – Graphical interpretation of the Householder Reflector.

Appendix D – Closest Point Proof

It was stated without proof in section 4.5 that the closest point on a circle with radius r is the point of intersection between the circle and a ray originating from the origin and passing through point p. This can be proven as follows.

The distance, d, between point p and a given point on the circle is:

$$\text{dist}^2 = (x_p - x)^2 + (y_p - y)^2$$

Defining d as dist^2 and recalling that $x^2 + y^2 = r^2$, the distance equation is solved for x:

$$d = (x_p - x)^2 + (y_p - y)^2$$

$$\begin{aligned}
d &= (x_p - x)(x_p - x) + (y_p - y)(y_p - y) \\
d &= x_p{}^2 - 2xx_p + x^2 + y_p{}^2 - 2yy_p + y^2 \\
d &= x_p{}^2 + y_p{}^2 - 2(xx_p + yy_p) + x^2 + y^2 \\
d &= x_p{}^2 + y_p{}^2 - 2(xx_p + yy_p) + r^2 \\
d &= -2xx_p - 2yy_p + x_p{}^2 + y_p{}^2 + r^2 \\
d &= -2x_p x - 2y_p\sqrt{r^2 - x^2} + x_p{}^2 + y_p{}^2 + r^2
\end{aligned}$$

Notice that when d is at a minimum (actually, the absolute value of d), dist^2 will also be at a minimum. The minimum can be found by taking the derivative and setting it equal to zero.

$$\frac{d}{dx}d = \frac{d}{dx}\left[-2x_p x - 2y_p\sqrt{r^2 - x^2} + x_p{}^2 + y_p{}^2 + r^2\right] = 0$$

$$-2x_p - 2y_p \frac{1}{2}\left(r^2 - x^2\right)^{-\frac{1}{2}}(-2x) = 0$$

$$\frac{2y_p x}{\sqrt{r^2 - x^2}} = 2x_p$$

$$\frac{x}{\sqrt{r^2 - x^2}} = \frac{x_p}{y_p}$$

Recalling that $m = \frac{y_p}{x_p}$ gives:

$$\frac{x}{\sqrt{r^2 - x^2}} = \frac{1}{m}$$

$$\frac{x^2}{r^2 - x^2} = \frac{1}{m^2}$$

$$x^2 = \frac{1}{m^2} r^2 - \frac{1}{m^2} x^2$$

$$x^2 + \frac{1}{m^2} x^2 = \frac{r^2}{m^2}$$

$$x^2 \left(1 + \frac{1}{m^2}\right) = \frac{r^2}{m^2}$$

$$x^2 = \frac{r^2}{m^2\left(1 + \frac{1}{m^2}\right)} = \frac{r^2}{\left(1 + m^2\right)}$$

Thus, it has been proven that $x = \sqrt{\frac{r^2}{1 + m^2}}$.

Vita

James Steven Masters was born in Denver, Colorado on July 24th, 1975. He is a Kung Fu fighter (he's fast as lightnin') and has mad disco skills. In his youth, he made it through the Jedi trials but was not knighted because he couldn't bear to part with his padawan braid. He never missed new-comic-book day growing up and still knows the secret identities of all the X-men and Avengers. He had a childhood much too happy to consider a career as an avant-garde artist and graduated in 1993 from Greeley West (go Spartans!). He spent three years in an Airborne Ranger Battalion where he graduated from Ranger School, Stinger School, Assault Climber School and Jungle Warfare School. He once shot down a drone with a Stinger missile.

After serving in the Rangers, he went back to Colorado where he served in the Army Reserves, entered the Mechanical Engineering program at Colorado State University and met his beautiful wife on a blind date. He graduated from Colorado State with Honors, Cum Laude. After graduation, he spent the summer with his wife working in orphanages in Romania and putting on youth camps through a Romanian Bible school. Upon his return to the States, he entered grad school at Georgia Tech where he received a Masters of Science in Aerospace Engineering. His favorite color used to be green but now it is orange.

He is a seasoned triathlete who has finished long and ultra distance races. He works by day as an engineer and has the ultimate nerd-street-cred title of Engineer/Scientist. No one knows what he does by night, but the streets of his city are curiously free of crime.

He knows the meaning to all of the verses in the Beatles song "I Am the Walrus." He's a master of disguise, has friends in every town and village from here to the Sudan, he speaks a dozen languages and knows all the local customs; he'll blend in, disappear, you'll never find him.

He has written and presented multiple papers at AIAA conferences. He currently works as a computational fluid dynamisist and the letters on his (non-vanity) license plate are CFD. He was team lead for the team that won the 2001 ASME National Human Powered Vehicle Competition. His inimitable Lego™ sculptures have brought onlookers to tears and he has been known to rock the Kasbah. He can both lead a horse to water and make it drink. He has always dreamed of climbing Mount McKinley and earning a PhD (time to start packing for Alaska).

[Editor's Note: not all of the claims in the above account were rigorously fact checked.]