Anisotropic Adaptation: Metrics and Meshes

by

Douglas Pagnutti

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

The Faculty of Graduate Studies

(Mechanical Engineering)

The University of British Columbia

February 20, 2008

Abstract

We present a method for anisotropic mesh refinement to high-order numerical solutions. We accomplish this by assigning metrics to vertices that approximate the error in that region. To choose values for each metric, we first reconstruct an error equation from the leading order terms of the Taylor expansion. Then, we use a Fourier approximation to choose the metric associated with that vertex. After assigning a metric to each vertex, we refine the mesh anisotropically using three mesh operations. The three mesh operations we use are *swapping* to maximize quality, *inserting* at approximate circumcenters to decrease cell size, and *vertex removal* to eliminate small edges. Because there are no guarantees on the results of these modification tools, we use them iteratively to produce a quasi-optimal mesh. We present examples demonstrating that our anisotropic refinement algorithm improves solution accuracy for both second and third order solutions compared with uniform refinement and isotropic refinement. We also analyze the effect of using second derivatives for refining third order solutions.

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost I'd like to thank my supervisor, Dr. Carl Ollivier-Gooch, whose time and patience was invaluable from inception to completion of this thesis. I'd also like to thank my lab-mates Chris, Serge and Amir. Without your help with my numerous coding problems, chances are I'd never have finished. I consider myself exceptionally lucky to have worked with all three of you.

I'd also like to thank my wife Danielle, for enduring so many countless delays and providing a limitless supply of love and support. You not only helped me throughout my degree, but made sure the whole process was enjoyable.

Finally I'd like to thank my family for their endless encouragement. It always helps to have your own personal cheering squad.

Dedication

To my mom, whose strength has been an inspiration.

Co-Authorship Statement

All of the research and data analysis contained within this thesis were accomplished by Douglas Pagnutti. The manuscript presentation was also done by Douglas Pagnutti with significant contributions from Dr. Carl Ollivier-Gooch.

Introduction

Partial differential equations (PDEs) are one of the most useful tools by which we describe our environment. These equations, that relate functions and their derivatives, allow scientists and engineers to model how things behave and thus better predict how they will behave in the future. Unfortunately, predicting such behaviour requires not just deriving the governing equations, but also solving them accurately and efficiently. For some PDEs, it is possible to derive analytic solutions that are easy to use and infinitely accurate. In other cases, it may be possible to approximate the solution (or key properties of the solution) with simple curves and then to choose parameters for those curves based on experimental analysis. This second method, however, has significant limitations in terms of accuracy and cost. A third method of solving PDEs — and one which has risen in popularity due to increasing computer power — is numerical analysis. The key to numerical analysis is to break up a large and complicated problem into many small, simple problems that can be processed through a computer. While such methods have recently seen dramatic improvements in efficiency, there is still a strong desire in industry for numerical analysis software that can produce more accurate solutions more quickly.

Computational Fluid Dynamics

One particular set of PDEs that is a focus of numerical analysis is those governing fluid behaviour. These PDEs are often too difficult for analytic solutions and the dynamic and multi-scale nature of the solutions can make experimental analysis difficult and costly. Because of this, an entire field of study has been created to produce numerical solution for problems involving fluids. This field of study is referred to as computational fluid dynamics

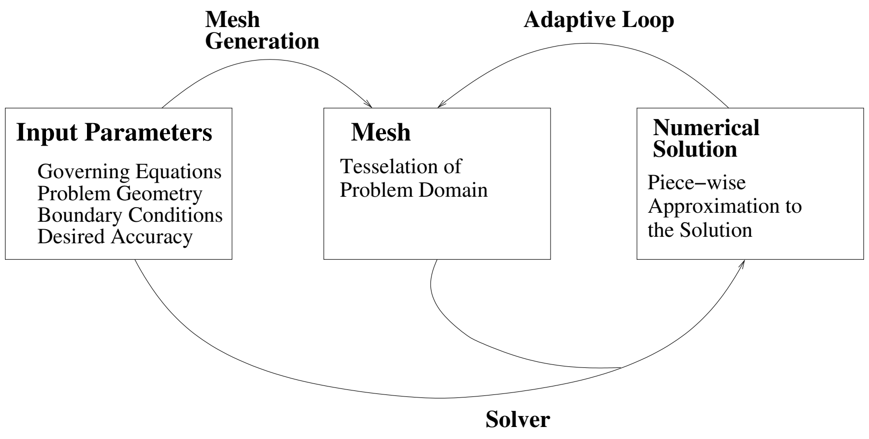


Figure 1: Illustration of the CFD process with Adaptive Loop.

(CFD).

The goal of CFD is simple: to produce accurate solutions in short amounts of time. This thesis contributes to that goal by combining two recent developments in CFD: anisotropic mesh adaptation and high-order accurate methods (of order greater than two).

The CFD process can be thought of as an interaction between three objects: the problem model, the mesh, and the numerical approximation. These three objects are illustrated by the boxes in Figure 1. To generate a solution, the standard approach is to model the problem, create a mesh for the modeled domain, and then approximate the modeled equations on the mesh. These are illustrated by the Mesh Generation and Solver arrows of Figure 1.

The first step in generating a CFD solution, and that which should require the most user input, is to specify the problem being solved. At a minimum, this includes the governing equations, the problem geometry and the boundary conditions. While this seems relatively straightforward, it is often the first source of error in CFD, especially for flows that involve complex physical phenomena such as turbulence and combustion. Examination of these sources of error is well beyond the scope of this thesis and so we will assume that the goal of CFD is strictly to solve the modeled problem. By making this assumption, we can treat solution error independently from modeling error. Ultimately, it should be the engineer's goal to correctly model the problem and the CFD program's goal to provide an accurate solution for that model.

Since infinite accuracy in infinitesimally small time is not yet possible, some further information must be given to determine how accurate the solution should be or how long it will take to compute. Currently, the most common way of doing this is to specify how finely the problem domain should be divided, or in other words, how small the elements of the mesh should be. The more a domain is divided, and thus the finer the mesh, the more accurate a solution should be and the more time that solution will take to compute. While this relationship between cell size and solution accuracy is generally true, it cannot provide a quantifiable measure of accuracy since local differences in element shape and size can dramatically change the solution. One approach for overcoming this difficulty is to repeat the problem with increasingly fine meshes until the variation from one solution to another is within acceptable tolerance. This is very inefficient and time consuming and one of the reasons why automated adaptive meshing shows so much promise.

Once the geometry and mesh cell size have been specified, the next step in generating CFD solutions is to create a mesh on which the solution is to be approximated. From an engineer's perspective, this is arguably one of the most frustrating and least understood parts of generating CFD solutions. It can also be very time consuming since it may require a large amount of user input. Essentially, a mesh is nothing more than a tessellation of the problem domain with shapes that can be recognized by the solver. There are generally two different types of meshes: structured and unstructured. Structured meshes consist of cells whose relation to one another, or connectivity, is implied by their numbering. For example, in structured meshes, cell (i, j) is always topologically to the left of cell $(i+1, j)$. Conversely, unstructured meshes require connectivity to be declared explicitly. The implicit connectivity of structured meshes allows for easier processing by the solver but it places restrictions on the topology of the mesh. Because unstructured meshes do not have this restriction, they are much better suited to meshing arbitrary geometries. This makes unstructured meshes more widely applicable and potentially reduces the amount of time and effort required by the user to produce an acceptable mesh.

As mentioned earlier, the size and shape of cells in the mesh can have a dramatic effect on solution accuracy. For solutions that are isotropic, it is generally preferable to have cells that are also isotropic (ideally, equilateral) and whose size is such that the error per cell is equidistributed. That is why the majority of mesh generation software tends to focus on creating cells with unitary aspect ratios and varying length scales. When the solution is anisotropic, however, the desired cell size and shape is less obvious. Rippa [34] concluded that, for linear interpolation of smooth functions, triangles should be long in directions where the magnitude of the second directional derivative is small and thin where the magnitude of the second directional derivative is large. It is intuitive that there should be analogous results for higher order interpolations. In Chapter , we combine size and quality measures to define a quality space for the cells of our mesh. Because there is no consensus on which specific triangle measure is preferable[1], our quality space is created from widely-used measures that are suited to our meshing approach: triangle circumradius and minimum edge length. Our hypothesis is that triangles within certain bounds in this quality space will produce more accurate solutions.

The final step in generating CFD solutions, and the most computationally intensive part, is performing the actual approximation. To do this, the governing equations are first discretized, typically using one of three common approaches: finite differences, finite elements, or finite volumes. While all of these methods can be analyzed using Taylor expansions of the solution, and are therefore compatible with the results in Chapter , it is still important to recognize the differences between the three methods.

One discretization method is the finite-difference method. This method discretizes equations by replacing the differential operator of the PDE with an equivalent difference operator. For example, if $\mathcal{L}(f) = \frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$, then an appropriate difference operator would be $\tilde{\mathcal{L}}(f) = \frac{f(x+h)-f(x)}{h}$ where h is now a finite value. If we assume that $f(x)$ is known at

[1]For a list of triangle quality measures, and their asymptotic behaviour, see [32].

a series of discrete locations $f(x+h)$, then we can approximate the solution at $f(x)$ using the difference formulation of the original PDEs.

Finite-element discretizations are similar to finite-difference ones in the sense that the solution f is assumed to be known at a discrete set of locations in the mesh. The solution between these locations f_{app} is then assumed to be a polynomial of a certain degree. Since this polynomial does not necessarily satisfy the partial differential equations $\mathcal{L}(f)=0$, there will be some error termed the residual $\mathcal{L}(f_{\text{app}})=0$. By convolving this residual with a set of weight functions, it is possible to choose polynomial coefficients that will minimize the residual and thus better approximate the actual solution.

Finite-volume discretizations are arguably the type of discretization with which engineers are most comfortable because it is identical to the control volume approach used to analyze thermodynamic and fluid dynamic systems. Instead of trying to calculate quantities at specific locations within a system, the average of those quantities is calculated by summing fluxes at the boundaries and applying them to the conservation form of the PDEs. To produce a finite-volume approximation for a set of PDEs, each cell in the mesh is considered its own control volume and the average of the solution within every cell is assumed to be known. Then, the flux through each cell boundary is approximated and used to update the cell average. This type of discretization is used throughout this thesis.

After discretizing the equations, it is then necessary to update the solution until either a desired time is reached or the solution no longer changes between iterations. For steady-state problems, when the solution no longer changes it is said to be converged. As the number of cells in a mesh increases, the required time to reach convergence likewise increases. Further complicating matters, some discretizations are only stable if the solution is advanced slowly, and this can dramatically increase computation time.

To address these instabilities, researchers have developed more complicated discretizations that take into account convection within the PDEs. For example, Roe's scheme [36] decomposes fluxes into pieces that are associated with certain physical characteristics and then only uses the cells that are upwind for each piece; this upwinding prevents aphysical propagation of information and improves stability.

Another way of improving stability, especially for steady-state problems, is through implicit time-stepping. This is done by assuming the solution is known at the current iteration as well as at a future iteration in surrounding cells. By doing this for every cell, we can create a large linear system that will generally be stable for larger time steps (assuming the spatial discretization is stable). While this allows for a converged solution to be found in fewer iterations, it produces a large linear system that can be very costly to solve. Fortunately, there have been many advances in numerical linear algebra that accelerate this process. Methods like GMRES [38] have been shown to substantially reduce computation time [23]. These linear algebra techniques are important to our work because the efficacy of these matrix acceleration methods is largely dependent on how well conditioned the matrix is. This matrix conditioning (a function of the eigensystem of the matrix) is highly dependent on the shape and connectivity of the cells in the mesh. Using a mesh with highly anisotropic cells that vary in shape and size may reduce (or improve) the effect of matrix acceleration methods. Although matrix conditioning is not explicitly discussed in this thesis, this is clearly one of the issues that must be addressed before anisotropic adaptation is more widely accepted in the CFD community.

Local Reconstruction and Error Estimation

All solution-based adaptation is based on the concept that the ideal mesh is one which equidistributes the error. However, without knowing the exact solution, it is impossible to know the exact error. The error must therefore be estimated from the approximate solution. Researchers typically use one of three main approaches to error estimation: interpolation-based, adjoint-based, and feature-based error estimation.

The error estimator that we use in Chapter is based on the interpolation of the solution between known values. Regardless of whether the solver is finite-difference, finite-volume or finite-element, the general approach for CFD is to assume that the solution is known exactly at a set of locations and then to use those values to interpolate the solution at some other location. The exact way that this is done may vary, but nearly all solvers use Taylor expansions in one form or another.[2] We can illustrate this with a one dimensional example.

Assume we know a solution $f(x)$ has an infinite Taylor expansion with a finite radius of convergence in the neighbourhood of some location a

$$f(x) = f(a) + \sum_{i=1}^{\infty} \frac{1}{i!} \frac{d^i f}{dx^i}(a)(x-a)^i$$

For a p order solver, we would approximate this infinite sum using only terms up to degree $p-1$

$$f(x) \approx f(a) + \sum_{i=1}^{p-1} \frac{1}{i!} \frac{d^i f}{dx^i}(a)(x-a)^i$$

The difference between the actual solution $f_{exact}(x)$ and our approximation $f_{approx}(x)$ is therefore

$$f_{exact}(x) - f_{approx}(x) = \sum_{i=p}^{\infty} \frac{1}{i!} \frac{d^i f}{dx^i}(a)(x-a)^i$$

As the difference between x and a decreases, the difference between the exact solution and our approximation is

$$f_{exact}(x) - f_{approx}(x) \approx \frac{1}{p!} \frac{d^p f}{dx^p}(a)(x-a)^p$$

Thus, if we were to use the known quantities at a to approximate the solution a small distance of h away, the error in that approximation would depend on both h and p.

$$\text{Error}(h,p) \approx \frac{1}{p!} \frac{d^p f}{dx^p}(a)(h)^p$$

In finite-volume solvers, this approximation is used to determine the flux at cell boundaries

[2]Some solvers rely on other expansions, such as the Fourier series, however those solvers are in the minority and are not discussed in this thesis.

and from that an update to the average solution within the cell. For steady-state problems, when the solution is converged, the error can be directly related to the size h that separates the approximation points as well as the order p of the solver. If the mesh spacing h is decreased everywhere, then the accuracy of the solution should increase by a factor of h^p. Hence the solution is order p accurate. Unfortunately, for most problems of interest, the p^{th} derivatives of f can vary greatly throughout the mesh. This leads to approximations that are more accurate in some locations than in others. Consequently, decreasing h where the p derivatives are high improves the solution much more than the same decrease in other areas.

This is the principle behind interpolation-based refinement. If the p^{th} derivatives are somehow reconstructed, then an error distribution can be approximated. When the mesh is refined, instead of uniformly decreasing h, elements of the mesh can be sized to match the error distribution. Ideally, this would increase the accuracy of the solver, in the sense of minimizing the solution error for a given number of unknowns. Most research into adaptive anisotropic refinement has been for second order solvers. In this case, the interpolation error can be related to the Hessian H. In two dimensions,

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

If this matrix is positive-definite and invertible, then the ideal cell can be found by linearly transforming an isotropic cell by H^{-1}. This is roughly the approach used by [8, 10, 12, 13, 15]. For third order solvers and higher, the derivatives can be grouped into a higher rank tensor; however, the shape of the ideal cell in these cases is unclear. Previous to the current thesis, no papers have been published that extend interpolation based refinement to higher order solvers and this is the principal focus of Chapter .

Unfortunately, even if the mesh perfectly equidistributes the interpolation-error estimate, p order accuracy would rarely be achieved. This is primarily due to the presence of convected error. Interpolation based error estimation assumes that the solution at surrounding locations are all exact and thus the only source of error comes from the higher-order terms of the Taylor expansion. However, if the values used in the Taylor approximation themselves contain error, that error will be passed on to the interpolated value which will in turn affect other Taylor expansion terms and so on. The net result is that small errors from one location of the mesh can have a big effect in locations that are relatively far away. This is particularly a problem for PDEs that contain strong convective terms such as the Euler equations where a strong feature like a shock might be refined in the wrong location.

An approach that several research groups (including, for example, [41, 3, 40, 42]) have used to address this issue is adjoint-based error estimation. Adjoint-based error estimation involves solving both the primal PDEs as well as the adjoint PDEs. Then, the error in a functional is directly related to local residual errors through the adjoint variables. This provides a more accurate analysis of where the functional is most affected by errors in the solution and hence a solid basis for where the mesh should be refined. Choosing a functional is both an advantage and a disadvantage for using this method. For situations where the only desired output is the functional, adjoint-based methods are ideal because they do not refine areas of the mesh where the solution has no effect on that functional. For example, if the only component of the solution analyzed is drag on an airfoil, then the meshes created by adjoint-based refinement will provide better drag predictions than other meshes of similar size. However, if that same mesh is used to predict lift, or any other component of the solution, it may not provide sufficient accuracy. Another disadvantage of this method is that solving the adjoint problem can be complicated and time consuming.

The third error estimator that some researchers use is feature-based error estimation (an excellent example of this is [18]). Feature-based error estimation explicitly detects and analyzes flow features and refines the mesh according to those flow features. This process relies on *a priori* knowledge of what the solution will be composed of (shocks, wakes, boundary layers, etc...) as well as a reliable feature-detection system. The advantage of explicitly detecting features is that it allows for more specific refinement criteria such as orthogonality in the boundary layers, vertex alignment with shocks and other mesh adaptations. Unfortunately, the feature detectors must be created and tuned for each particular problem and the actual refinement criteria are much more heuristic; for example, it is unclear how much one feature should be refined relative to another. Another drawback is that feature detection is often based on the same local parameters that interpolation-based error measures use. That means that convected error is ignored and it is possible to detect features in the wrong location.

We have chosen to base our error estimation on the local reconstruction error for a number of reasons. First, since this error measure relates directly to the order of the discretization, the extension to high-order solutions is intuitive. Second, local error estimation is relatively cheap compared to adjoint-based error estimation. Finally, current adjoint methods provide an ideal size distribution for cells but not an ideal shape distribution and so researchers such as Venditti and Darmofal [42] rely on local error reconstruction to define the local anisotropy.

Communicating the Anisotropy and Creating an Anisotropic Mesh

Once the desired anisotropy has been defined, it is necessary to communicate that anisotropy to the meshing program and create an anisotropic mesh. The most standard method of defining anisotropy is through a metric [6, 7]. A metric is a function that defines a scalar distance between points in a domain. Like the standard distance measure in Cartesian space, metric functions are required to be positive-definite, symmetric and they must also satisfy the triangle inequality. This type of function is ideal for creating meshes because geometric properties based on lengths have equivalent properties according to the metric function. In particular, most research groups use the family of metrics created by linearly transforming the standard Cartesian distance measure, see [16, 15, 8] among others. We also use these types of metrics by assigning a symmetric positive-definite matrix to each vertex. The choice of these matrices is discussed in Chapter .

Unfortunately, variations in the error also require variations in the metric and this is where researchers have developed different approaches. The key problem with a varying metric is that, even though the metric is valid point-wise, it may no longer globally satisfy the triangle inequality. Courty et al. [12] linearly interpolate the metric and deal with the triangle inequality heuristically. Others, such as Labelle and Shewchuk [16] assume the metric is only valid within the Voronoi cell of each vertex. This leads to very complicated Voronoi diagrams that require certain restrictions on the metric in order for the Voronoi diagram to dualize to a valid triangulation.

As part of the anisotropic meshing approach in Chapter , we have chosen to approximate the metric for the interior of triangles using the average metric from the three vertices. For triangle edges, we average the length that would be calculated from the two triangles. These simple approximations allows for quick estimates of triangle quality while imposing no restrictions on metric variation.

After the metric is defined, it is necessary to create a mesh that is roughly isotropic when measured by the metric. To create two-dimensional isotropic unstructured meshes, there are three principal approaches: quad-tree, advancing-front, and Delaunay meshing.

Quad-tree meshing [43, 9] discretizes the domain by first creating a square that contains the entire domain. Then, that square is recursively split until all the squares satisfy some assigned size property and the boundary is suitably discretized. The difficulty with anisotropic quad-tree meshing is that the previously-mentioned squares must somehow be divided into rectangles with varying aspect ratios and directions.

In the advancing front method [17], the meshing procedure begins with a set of seed vertices (usually along the boundaries) connected into a closed loop called a front. Then, a decision is made as to where to insert the next vertex so that it will create a high quality triangle.

The front is thus updated, and the process repeats. The main drawback of this method is that when different meshing fronts meet, some heuristic approach is required to unify the fronts without creating poor-quality triangles. For anisotropic adaptation, this method requires removing large tracts of the mesh and then re-meshing these areas according to the metric.

The meshing approach that we have modified for anisotropy in Chapter is Delaunay meshing. A Delaunay triangulation of vertices is a triangulation where the interior of every triangle's circumcircle is empty. This triangulation always exists, and is equivalent in two dimensions to the triangulation that maximizes the minimum angle in the mesh. Delaunay mesh refinement takes a pre-existing Delaunay triangulation of vertices and then inserts vertices at the circumcenter of any triangle with a small angle or a large circumradius. By repeatedly inserting vertices at circumcenters it is possible to create meshes with guaranteed quality. The exact guarantee depends on how the boundaries are handled, but in general, the smallest angle in the mesh is somewhere around thirty degrees[11, 37, 39, 28]. The reason that Delaunay meshing extends well to anisotropic meshing is because of its simple approach of choosing the ideal location to insert a vertex given a valid mesh. If the insertion location is chosen to produce anisotropic triangles according to the metric, then a high-quality anisotropic mesh can be created without any expensive local maximization routines.

The Anisotropic Mesh Adaptation Loop

Good error estimation and fast, high-quality anisotropic meshing algorithms are the two key ingredients for successful anisotropic mesh adaptation. Essentially, anisotropic adaptive meshing is similar to adding a feedback loop within a control system. In particular, we are trying to minimize an error function by controlling the mesh. What makes this problem so difficult is that the error function isn't necessarily known, and the mesh has an enormous number of dependent variables.

Despite the complications, this feedback loop can improve the effectiveness of CFD in several ways. Ideally, users are not required to spend inordinate amounts of time trying to create meshes that may or may not be well-suited to the problem. Furthermore, automatically created meshes provide higher accuracy with fewer cells than the isotropic, uniform meshes. Ultimately, increasing the number of cells with each adaptation instead of converging the mesh and solution with a fixed number of cells allows CFD users to gage whether or not the solution is converging to an infinitely-fine mesh solution and thus better estimate the overall accuracy of the solver.

In Chapter we examine the first difficulty in adaptive anisotropic mesh adaptation: estimating error and converting it to a form that can be used by the mesh generator. It is here that the added difficulties of adapting to higher-order solutions and multi-variable problems are solved. The former by taking a Fourier transform of an estimated error function, and the latter by choosing a standard norm of the variables.

In Chapter we use the metric derived in Chapter to refine a pre-existing mesh so that it appears isotropic when measured by the metric. The approach we use is to define a quality space based on approximate circumradii and minimum edge lengths as measured by the metric. Then, vertices are inserted at approximate circumcenters for triangles whose circumradii are above a certain bounds and vertices are removed for edges that are too small. Several of these insertion and removal iterations are performed on the mesh, leaving the vast majority of triangles within the desired quality bounds.

These two chapters deal with incredibly different aspects of CFD and thus warrant individual attention. However, the two Chapters are also strongly coupled because it is useless to have anisotropic error estimation without being able to create anisotropic meshes. Likewise, high-quality anisotropic meshes cannot be created without first defining what that anisotropy should be. It is the combination of these two articles, and hence this thesis, that provides the complete view of two-dimensional anisotropic mesh adaptation and how it can be used to improve CFD.

A Generalized Framework for High Order Anisotropic Mesh Adaptation

Introduction

There is, in general, an inverse relationship between the accuracy of CFD solutions and the time required to produce them. Increases in computational resources may have significantly improved this relationship, but there is still a strong desire in industry for more efficient CFD. Consequently, researchers have developed a variety of improvements to the CFD process. Two such improvements are anisotropic adaptation and higher-order methods.

The first improvement, anisotropic adaptation, is the process of using the solution on an initial mesh to produce a better, anisotropic mesh. Since the new mesh is better suited to the solution, the accuracy should increase. If the cycle is repeated, the mesh should converge to one which is optimal for the problem being examined. There are multiple advantages to adding this feedback loop in a CFD algorithm. First, it significantly reduces the dependence of the final solution on the mesh and should thus increase confidence in the results. Second, to achieve the same accuracy on a uniformly refined grid might require so many cells that the computation becomes unfeasible. Another advantage, and one which should not be overlooked, is that creating a mesh is often the most frustrating and user-intensive part of CFD. By automating this process, we substantially increase the usability of CFD programs.

The second improvement, high-order methods, are another recent improvement to CFD. Most current CFD code creates a piecewise linear reconstruction of the solution. If the solution is smooth, the accuracy of that reconstruction is proportional to the size of the mesh squared. By using higher order polynomials to reconstruct the solution, we can increase the order of accuracy. For example, if the solution is reconstructed with a quadratic polynomial, the accuracy will be proportional to the size of the mesh cells cubed (third-order). While this incurs an increase in computational cost per cell, the increase in order of accuracy means that far fewer cells are required and there is a potential net gain in efficiency.

Combining these two improvements is a non-trivial task. The most common approach to anisotropic adaptation is to refine based on the Hessian of a solution variable[8, 10, 12, 13, 15]. The reasoning for this is that the error on a piece-wise linear interpolation of a smooth function is bounded by a quadratic term involving the second derivatives (the Hessian) of the function. When higher-order interpolation is used, as it is with higher-order methods, this reasoning is no longer valid. At the time of writing, we are the only authors to extend Hessian-based anisotropic refinement to higher order methods.

To accomplish this, we first model the local error with a polynomial from the Taylor expansion. This polynomial is then used to construct a metric function through a Fourier transform. By taking this approach, our metric derivation can be applied to any order reconstruction and any number of variables.

We begin this article by first examining how the local error can be approximated using Taylor expansions, and how this compares to other methods of error estimation. Then we show how an appropriate metric function can be chosen irrespective of the number of variables and the order of the error. We then show the benefits of this metric function for three different two dimensional flows over a simple airfoil. The first test case compares adapted refinement and uniform refinement for second and third order solutions of subsonic inviscid flow. The second test case examines how the metric performs in the presence of discontinuities in the solution. The third and final test case examines the benefits of anisotropic adaptation versus isotropic adaptation for viscous flow. This final test case is also used to examine the differences between second and third order metrics for third order solutions.

Error Estimation

Analytic *a priori* error analysis for CFD solvers — whether finite volume, finite element, or finite difference — is based on Taylor series analysis. While the details differ between families of schemes, in all cases the analysis assumes the solution to the PDE(s) has a valid, converging Taylor expansion at every point in the domain. If the scheme is of order p, we approximate the solution $f(x)$ around some location a by the Taylor expansion terms up to degree $p-1$. For example, in a one-dimensional, third order scheme, we would approximate $f(x)$ around a as

$$f(x) \approx f(a) + \frac{df}{dx}(a)(x-a) + \frac{1}{2}\frac{d^2 f}{dx^2}(a)(x-a)^2 \tag{1}$$

The difference between the exact function $f_{\text{exact}}(x)$ and the approximate function $f_{\text{app}}(x)$ is therefore the remaining terms from the Taylor expansion. Continuing the example in Equation 1,

$$f_{\text{exact}}(x) - f_{\text{app}}(x) = \frac{1}{3!}\frac{d^3 f}{dx^3}(a)(x-a)^3 + \frac{1}{4!}\frac{d^4 f}{dx^4}(a)(x-a)^4 + ... \tag{2}$$

For small areas around a, the leading term in this error is a polynomial of degree p and only p. We exploit this characteristic in our error indicator.

The accuracy of the solver depends on how well it can evaluate $f(x)$ at some other location, say b. The consequence of this is that, if the distance between points in the mesh scales with h, the error in the solution should scale with h^p. This is illustrated by continuing the example in Equation 2,

$$\begin{aligned} f_{\text{exact}}(b) - f_{\text{app}}(b) &\approx \frac{1}{3!}\frac{d^3 f}{dx^3}(a)(b-a)^3 \\ \text{Error}(h) &\approx \frac{1}{3!}\frac{d^3 f}{dx^3}(a)h^3 \end{aligned} \tag{3}$$

In practice, solvers combine these approximations at a large number of discrete points to approximate the overall solution $f(x)$. While the precise way in which this is done varies from one solver to another, any Taylor-based scheme will have an error analogous to Equation 3. The generality of this derivation is one of the principal reasons engineers use this type of error estimation. However, other methods for estimating error exist, including adjoint methods and feature-based methods.

When we derived the reconstruction error, we assumed that $f_{\text{app}}(a)$ was known exactly. This assumption is not necessarily true, and any errors in the calculation of $f_{\text{app}}(a)$ will be convected to the solution at b. Since this process is repeated from cell to cell small errors can be convected and cause large errors far away from their source. Adjoint-based error measures [3, 41] attempt to capture this convected error. To do this, they solve both the original PDEs and the adjoint PDEs for some desired functional. These two solutions can then be compared to determine where that functional is most dependent on the solution and thus refine the mesh accordingly. The advantage of this error measure is that it gives a much more global estimate of the error, which is especially important for PDEs with substantial convection components. Unfortunately, solving the adjoint PDEs can be a very time consuming process. The meshes that are created by this method are also strongly dependent on the chosen functional. If, for example, a mesh is created to increase the accuracy of the aerodynamic drag, it might have a less accurate prediction of lift than an mesh optimized for lift accuracy.

Another method of estimating error, feature-based error estimation [18], explicitly detects and analyzes flow features. Mesh refinement is then done to resolve those flow features according to some pre-defined set of rules. This is especially useful for imposing more strict mesh refinement criteria such as mesh orthogonality in boundary layers and vertex alignment with shocks. Unfortunately, this method has much less of a theoretical basis for how one flow feature should be refined relative to another and the feature detectors must be tuned for each type of flow.

While both of these methods have their respective advantages and disadvantages, we have chosen to focus on local reconstruction error because it provides an error measure that is directly dependent on the solution vector instead of a particular functional. This is also the approach used by the majority of researchers [8, 10, 13, 12, 15].

Other sources of error such as modeling error and numerical roundoff error are not addressed in this paper. It is assumed that the desired solution is the solution to the modeled problem, which may differ from the actual problem. In this way modeling error can be addressed independently. Also, numerical roundoff error is assumed to be much smaller than the previously described sources of error and so we will neglect it.

Calculating the Metric

A Metric is a function $d : X \times X \to \mathbb{R}$ such that it satisfies the following conditions

$$\begin{aligned} d(a,b) &> 0 \iff a \neq b \\ d(a,a) &= 0 \\ d(a,b) &= d(b,a) \\ d(a,c) &\leq d(a,b) + d(b,c) \end{aligned} \tag{4}$$

More simply put, a metric defines the distance between points in a way similar to the standard Cartesian distance metric $d_{\text{std}}(a,b) = \sqrt{(a-b)^T(a-b)}$. Because of these properties, any geometry that can be defined in terms of distances has equivalent properties according to the metric. The metric that we use for anisotropic adaptation, $d_{\text{ani}}(a,b)$, is an invertible linear transformation L of the standard distance measure $d_{\text{std}}(a,b)$.

$$\begin{aligned} d_{\text{std}}(a,b) &= \sqrt{(a-b)^T(a-b)} \\ d_{\text{ani}}(a,b) &= \sqrt{(L(a-b))^T\,(L(a-b))} \\ d_{\text{ani}}(a,b) &= \sqrt{(a-b(L^TL)(a-b)} \\ d_{\text{ani}}(a,b) &= \sqrt{(a-b)^TM(a-b)} \end{aligned} \tag{5}$$

The metric $d_{\text{ani}}(a,b)$ can thus be defined by the symmetric positive definite matrix $M = L^T L$.

To calculate the values of M from Equation 5 for a solution of order p we first assume that the error is equal to the p degree terms of the Taylor expansion, as discussed in Section . Also, it is intuitive to assume that errors which are positive are equivalent to errors which are negative. Therefore the error measure is modified to be the absolute value of the p order polynomial from the Taylor expansion. For example, the error for a function $f(x,y)$ at a distance $(\Delta x, \Delta y)$ from the reference point of the Taylor series expansion in a two-dimensional, third order solution is

$$\text{Err}(\Delta x, \Delta y) \approx \left| \frac{1}{6}\frac{d^3 f}{dx^3}(\Delta x)^3 + \frac{1}{2}\frac{d^3 f}{dx^2 y}(\Delta x)^2(\Delta y) + \frac{1}{2}\frac{d^3 f}{dxy^2}(\Delta x)(\Delta y)^2 + \frac{1}{6}\frac{d^3 f}{dy^3}(\Delta y)^3 \right|$$

For solutions with multiple variables, it is necessary to take a scalar measure of the individual variable errors. Our approach is to use one of the standard vector norms. Another method of joining error variables is the concept of maximum contained error-ellipse as proposed by Huang[15]. However, it is unclear how this should be constructed for more than two variables and it is significantly more complicated to compute. For the remainder of this paper we will use the L_1 norm, or the average of the polynomials[3].

Using the previously stated assumptions, we can therefore write a general equation for the error of a p order solution in d dimensions, with n variables over a vector $(\Delta x_1, ..., \Delta x_d)$.

$$\text{Err}(\Delta x_1, ..., \Delta x_d) \approx \frac{1}{n}\sum_{i=1}^{n} |P_i(\Delta x_1, \Delta x_2, ..., \Delta x_d)| \tag{6}$$

where P_i is a polynomial with terms of degree p and only p. We then re-write Equation 6 in d dimensional spherical coordinates.

$$\text{Err}(\Delta r, \theta_1, ..., \theta_{d-1}) \approx (\Delta r)^p f(\theta_1, ..., \theta_{d-1}) \tag{7}$$

[3]When running the test cases in Section , L_2 and L_∞ norms were found to refine almost exclusively based on flow velocity (ideal for drag predictions) whereas the L_1 norm better accounted for pressure (and better predicted lift).

We can always factor out $(\Delta r)^p$ in this equation because all the terms of each polynomial P_i are of degree p. In general, $f(\theta_1, \theta_2, ..., \theta_{d-1})$ is a complicated, piece-wise smooth function that captures the anisotropy of the error. We also perform a similar procedure for the metric function from Equation 5.

$$d(\Delta x_1, ..., \Delta x_d) = \sqrt{(\Delta x_1, ..., \Delta x_d)^T M (\Delta x_1, ..., \Delta x_d)} \tag{8}$$

Re-writing in d dimensional spherical coordinates

$$d(\Delta r, \Delta\theta_1, ..., \Delta\theta_{d-1}) \quad = \quad (\Delta r)\sqrt{g(\theta_1, \theta_2, ..., \theta_{d-1})} \tag{9}$$

If we consider an edge of length Δr, then the error for a p order solution will be proportional to $(\Delta r)^p$. This should also be the case when Δr is measured in the metric. This ensures the metric length and error will have the same scaling as Δr changes. Thus we want to choose d such that

$$(d(\Delta r, \Delta\theta_1, ..., \Delta\theta_{d-1}))^p = \mathrm{Err}(\Delta r, \theta_1, ..., \theta_{d-1}) \tag{10}$$

Combining equations 7, 9 and 10,

$$\begin{aligned}(\Delta r)^p f(\theta_1, \theta_2, ..., \theta_{d-1}) &\approx \left((\Delta r)\sqrt{g(\theta_1, \theta_2, ..., \theta_{d-1})}\right)^p \\ (f(\theta_1, \theta_2, ..., \theta_{d-1}))^{\frac{2}{p}} &\approx g(\theta_1, \theta_2, ..., \theta_{d-1})\end{aligned}$$

If we choose values of M such that $g(\theta_1, \theta_2, ..., \theta_{d-1}) \approx (f(\theta_1, ..., \theta_{d-1}))^{\frac{2}{p}}$ then the metric is a good approximation to the error. Because f and g are both functions of an angle θ, we will use Fourier series to find an appropriate g. Repeating Equation 9 for two dimensions using the metric $d_{\text{ani}}(x, y)$ from Equation 5.

$$\begin{aligned}d_{\text{ani}}(1, \theta) &= \sqrt{g(\theta)} \\ g(\theta) &= d_{ani}(1, \theta)^2\end{aligned}$$

$$
\begin{aligned}
g(\theta) &= (\sin\theta, \cos\theta)^T M (\sin\theta, \cos\theta) \\
g(\theta) &= M_{1,1}\sin^2\theta + 2M_{1,2}\sin\theta\cos\theta + M_{2,2}\cos^2\theta \\
g(\theta) &= \frac{1}{2}(M_{1,1}+M_{2,2}) + \frac{1}{2}(M_{1,1}-M_{2,2})\cos 2\theta + M_{1,2}\sin 2\theta
\end{aligned}
$$

If we also do a Fourier transform of $f(\theta)$ around the unit circle we get

$$
\begin{aligned}
(f(\theta))^{\frac{2}{p}} &= \frac{1}{2}a_0 + a_1\cos\theta + b_1\sin\theta + a_2\cos 2\theta + b_2\sin 2\theta + ... \\
a_i &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\cos i\theta d\theta \\
b_i &= \frac{1}{\pi}\int_0^{2\pi} f(\theta)\sin i\theta d\theta
\end{aligned} \tag{11}
$$

Now, the original polynomials P_i from Equation 6 that are summed to find f are all either symmetric or all antisymmetric in x and y. When we write the L_1-norm of the P_i in polar coordinates, this gives a function $f(\theta)$ that is periodic with period π, not 2π; this is clearly illustrated in Figure 2. This in turn implies that the odd terms in the Fourier series (most importantly a_1 and b_1) must be zero.We can therefore write

$$
(f(\theta))^{\frac{2}{p}} \approx a_o + a_2\cos 2\theta + b_2\sin 2\theta \tag{12}
$$

The error in this approximation is related to the higher, rapidly-converging even-frequency terms of the Fourier expansion. If we set the coefficients of M as follows

$$
\begin{aligned}
M_{1,1} &= \frac{1}{2}a_0 + a_2 \\
M_{1,2} = M_{2,1} &= b_2 \\
M_{2,2} &= \frac{1}{2}a_0 - a_2
\end{aligned} \tag{13}
$$

we have a good approximation of $g(\theta)$ to $(f(\theta))^{\frac{2}{p}}$ and hence a good approximation of our metric to the estimated error. For M to be positive definite (as required by the metric), the Fourier approximation must also be positive definite. Unfortunately, a function which

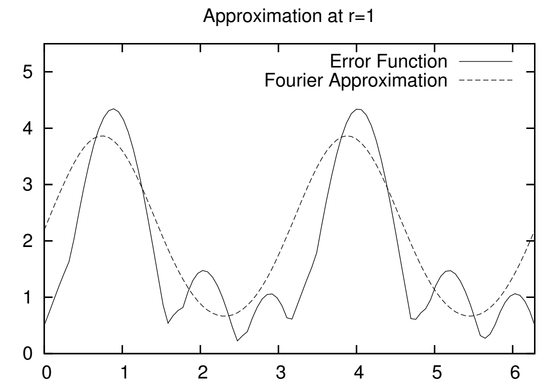


Figure 2: The error and associated Fourier approximation.

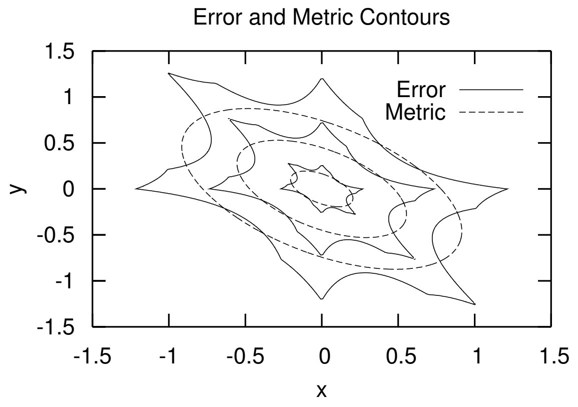


Figure 3: Error contours and $d(x,y)^3$ contours.

is positive semi-definite such as $(f(\theta))^{\frac{2}{p}}$ will not necessarily have a positive definite Fourier transform. To address this, we add the restriction that $\frac{1}{2}a_0 > \sqrt{(a_2)^2 + (b_2)^2}$ by instead having $a_0 = \max\left(a_0, 2\sqrt{(a_2)^2 + (b_2)^2}\right)$. This ensures and the metric M will always be positive definite.

As an example, assume that the variables u and v have the following third derivatives and that the solution is 3rd order ($p = 3$):

	$\frac{\partial^3}{\partial x^3}$	$\frac{\partial^3}{\partial x^2 y}$	$\frac{\partial^3}{\partial x y^3}$	$\frac{\partial^3}{\partial y^3}$
u	-4	-20	-16	2
v	6	10	2	-10

Then, by Equation 6, the error at a distance (x, y) away is

$$\text{Error}(x,y) = \left(\frac{\left|-\frac{2}{3}x^3 - 10x^2y - 8xy^2 + \frac{y^3}{3}\right| + \left|x^3 + 5x^2y + xy^2 - \frac{5}{3}y^3\right|}{2} \right) \tag{14}$$

The Fourier coefficients from Equation 12 can then be calculated by numerically integrating Equation 14 around a unit circle. This gives

$$(f(\theta))^{\frac{2}{3}} \approx (3.224) + (0.07636)\cos 2\theta + (0.8473)\sin 2\theta$$

A comparison of the Fourier approximation and the actual error measure around a unit circle in Figure 2 demonstrates the validity of this approximation. By Equation 13, the matrix that will be used in the metric is

$$M = \begin{bmatrix} 1.688 & 0.8473 \\ 0.8473 & 1.536 \end{bmatrix}$$

Substituting into our metric from Equation 8

$$d(x,y) = \left(1.688x^2 + 1.6946xy + 1.536y^2\right)^{\frac{1}{2}}$$

Our hypothesis was that $d(x,y)^3$ would be a good approximation to $\text{Error}(x,y)$. Comparing the same contour values in Figure 3 illustrates that our hypothesis was indeed correct. We have therefore chosen a metric that approximates our error measure.

One aspect of generating the error measure that hasn't been addressed is computing the order p derivatives. This is not a trivial task, especially for schemes of order of accuracy higher than two. However, a solver such as ANSLib[27] which can generate fourth-order solutions must already contain the algorithms necessary for reconstructing the third derivatives.[4] While reconstructing the derivatives and numerically integrating the error function for every cell requires computation time, we have found that in comparison with overall solution time, the cost is minor.

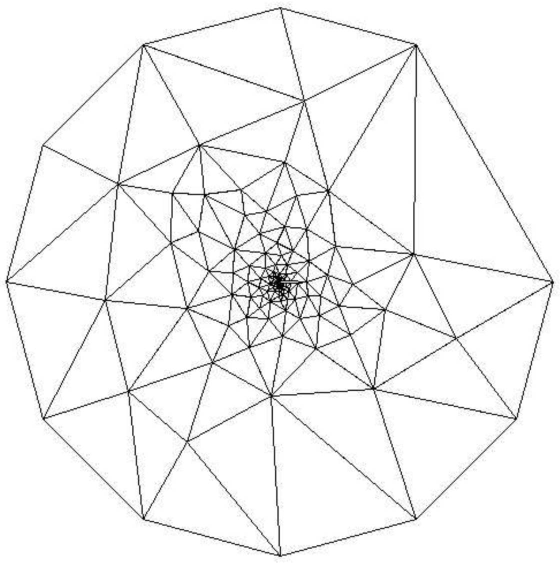
The process of determining Fourier coefficients also has the bonus of providing an intuitive control over the approximated error. Limits on a_0 control the average error for each cell, and limits on $\sqrt{(a_2)^2 + (b_2)^2}$control how much that error is allowed to vary within the cell (and thus anisotropy). It would also be logical to smooth the metric (if desired) based on these coefficients. However, we do not examine this in the following test cases.

Results

We present several test cases to demonstrate the effect of using our metric in an anisotropic adaptation cycle. We compare these results for both second and third order solutions with those using uniform refinement. All solutions are computed using the vertex-centered Finite-Volume solver described in [27]. This solver uses L_2 reconstruction [4, 29], Roe's scheme [36], and Newton-GMRES for rapid convergence [20, 22]. For viscous flows, the viscous terms are discritized as described in [29]. To create anisotropic meshes from the calculated metric, we use the anisotropic meshing modifications to GRUMMP [26] proposed in [30].

For all three cases, we start with the initial mesh shown in Figure 4. This mesh is the Delaunay triangulation produced by GRUMMP without any refinement parameters. The far-field boundary is a circle of radius 100 chords centered at the leading edge of the airfoil. We have chosen to use this mesh because it is requires the least user input and is obviously ill-suited to all three test cases. Furthermore, by choosing a large boundary, we hope to negate some of the errors that can be generated from boundary placement.

[4]For details on how this is done for finite volume solvers, see papers by Barth and Frederickson[4] as well as Ollivier-Gooch and Van Altena[29].


(a) Farfield view of initial mesh

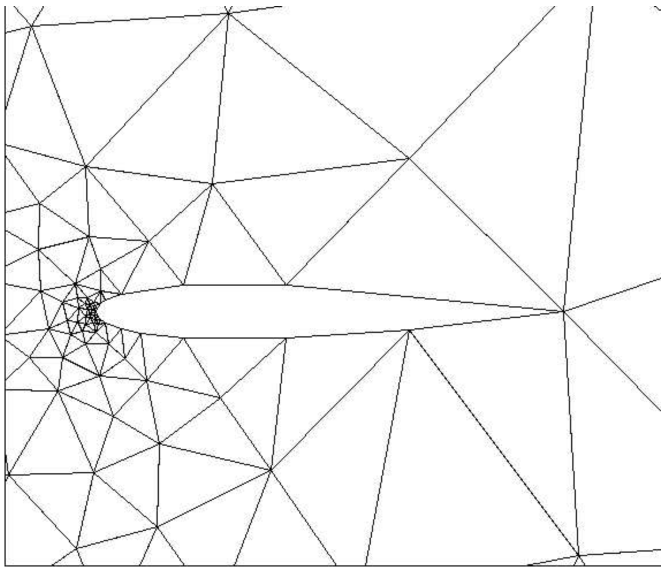

(b) Closeup view of intial mesh

Figure 4: NACA 0012: Initial mesh with 197 Vertices.

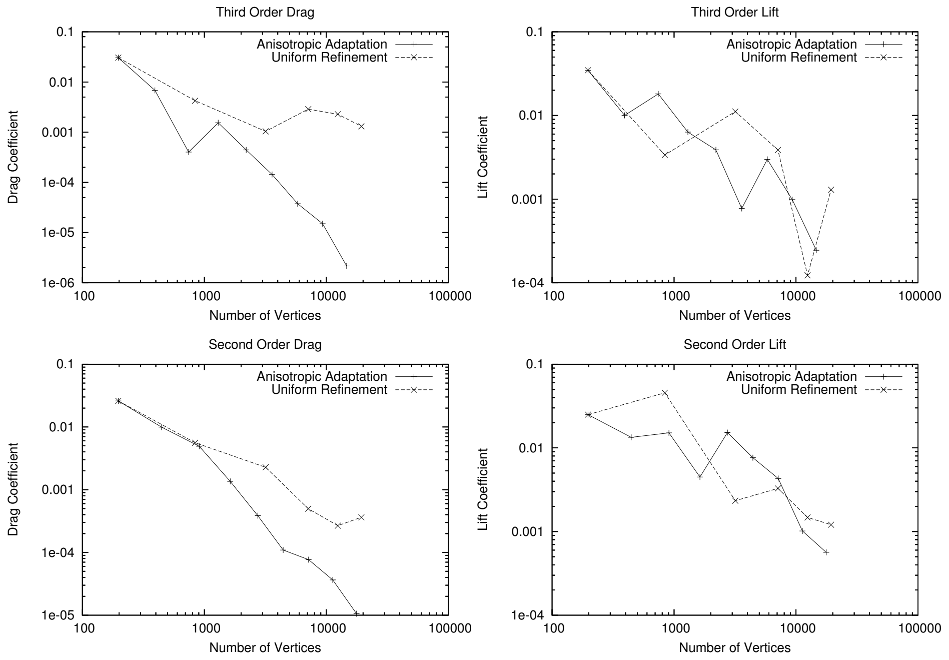
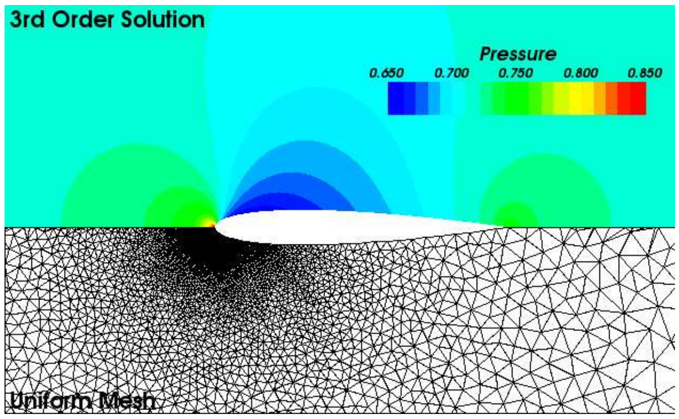


Figure 5: NACA-0012 subsonic inviscid flow: comparisons of drag and lift coefficients

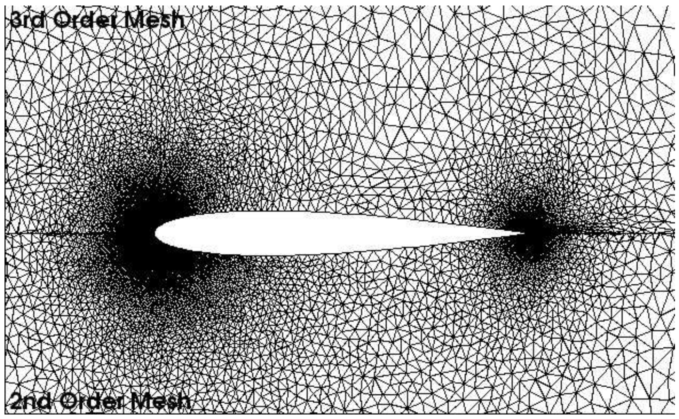
Subsonic inviscid flow around an airfoil.

The first test case is inviscid flow over the NACA-0012 airfoil. Theoretically, both the drag and lift coefficients should approach zero. However, due to the asymmetry of the mesh, this is rarely the case. Figure 5 compares drag and lift coefficients for both second and third order solutions. The uniform meshes used for comparison are created by incrementally increasing the refinement parameter in GRUMMP. In all four cases, the solution on the finest mesh was more accurate with anisotropic refinement than without. For drag predictions, and in particular for third order solutions, our refinement algorithm produced a noticeable increase in convergence order throughout the refinement process. Convergence of lift to zero with refinement is poor in all cases, because this quantity depends heavily on mesh symmetry, which doesn't improve nearly as rapidly as cell size decreases.

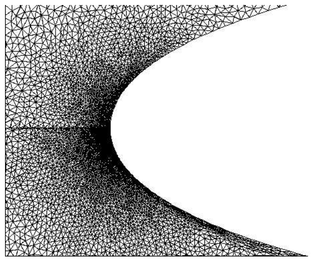
It is obvious in Figure 6 that the uniformly refined mesh fails to sufficiently resolve the trailing edge of the airfoil. While the second and third order meshes are very similar, close inspection reveals slight differences at the leading and trailing edges of the airfoil. For the



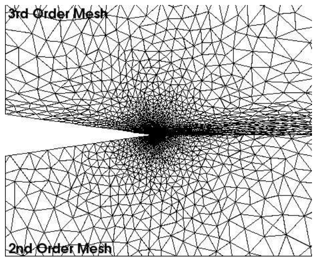
(a) Pressure distribution from third order solution (N Vertices) and uniformly refined mesh (N Vertices).



(b) Final Meshes: Third Order (top; N vertices) and Second Order (bottom; N Vertices).



(c) Leading Edge (top half: third order; bottom half: second order)



(d) Trailing Edge (top half: third order; bottom half: second order)

Figure 6: Inviscid Subsonic flow around NACA 0012: $Mach = 0.5$, $\alpha = 0$.

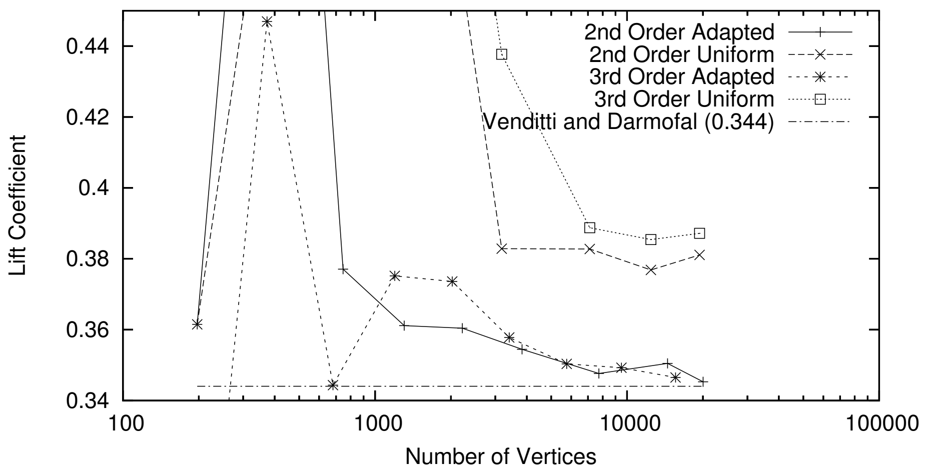


Figure 7: Lift Convergence for NACA 0012: Mach 0.8, $\alpha = 1.25^o$.

second order solution, high second derivatives of pressure parallel to the boundary produce anisotropic cells at the leading edge in the second order mesh but not in the third order mesh. At the trailing edge of the third order mesh, convection of entropy produced at the trailing edge (an artificial result of the point singularity) may produce the slight anisotropy. Since second order schemes dissipate entropy faster than third order schemes, the second order mesh is more uniform at the trailing edge.

Transonic Inviscid flow around an airfoil. (I just wrote this subsection so there are some holes and lack-of-flow problems)

Earlier in the derivation of our metric, we assumed that the underlying solution was smooth. This assumption is necessary for showing the validity of using Taylor expansions. Unfortunately, not all solutions of interest are actually smooth. In particular, for transonic and supersonic flows, discontinuities (or shocks) can appear in the solution. At these shocks, our error estimate is no longer valid. This does not mean, however, that our metric cannot be used for solutions in the presence of discontinuities. To illustrate this, we have examined Mach 0.8 flow over the NACA 0012 airfoil with an angle of attack of 1.25°. To avoid over-refining the discontinuities, we have limited our error measure from Equation 7 such

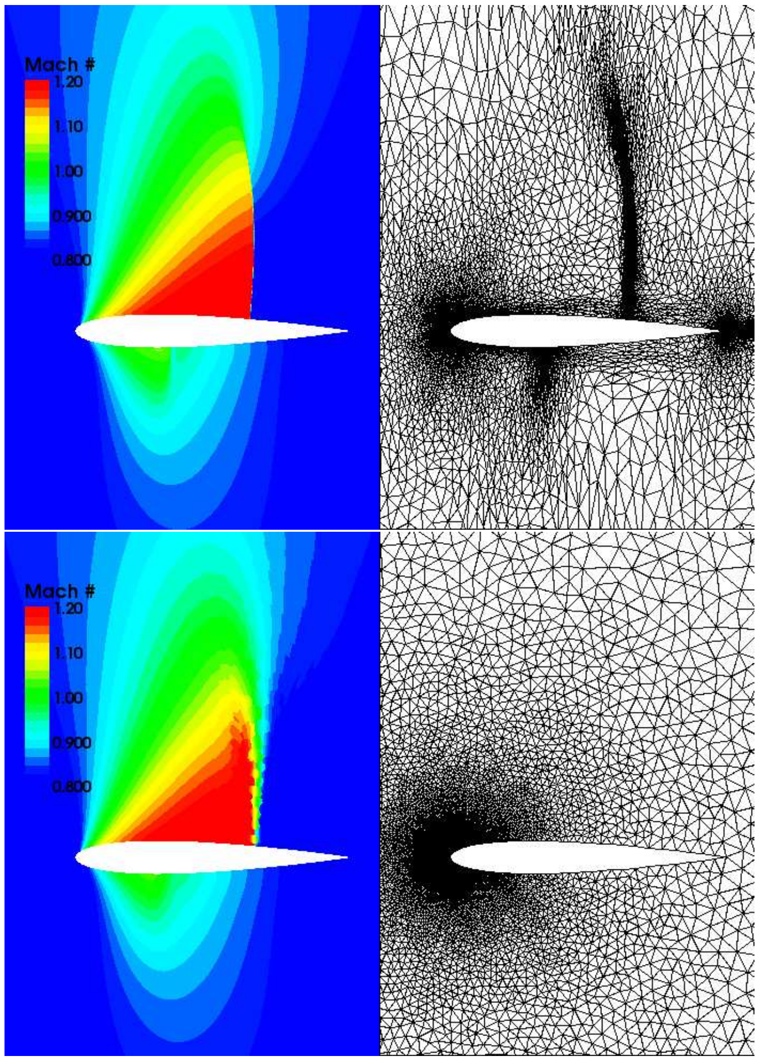


Figure 8: 3rd order Mach number profiles on adapted mesh and uniform mesh for NACA 0012: Mach 0.8, $\alpha = 1.25^o$

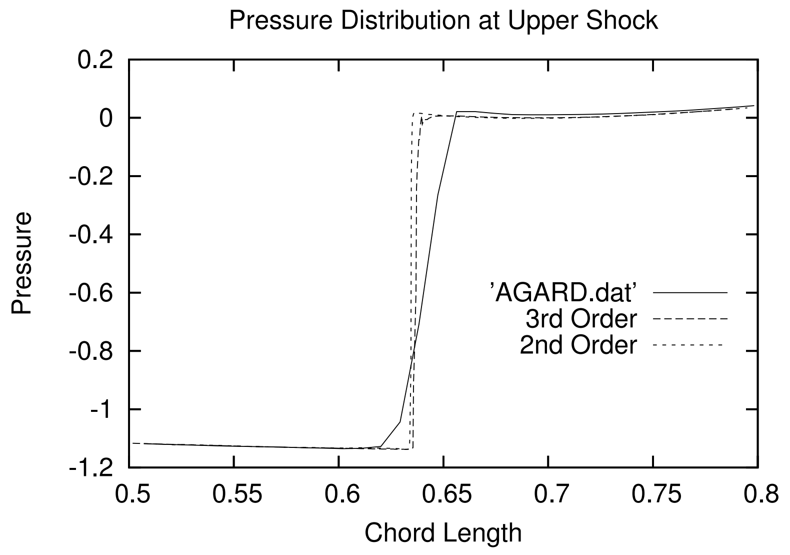


Figure 9: Pressure distribution on upper surface of NACA 0012:Mach 0.8, $\alpha = 1.25^o$.

that $\text{Err}(1, \theta) < 1 \times 10^3$. This number is chosen to bound the error measure at the shock without dramatically affecting other areas of the solution. In the future, this bound should be based on a statistical distribution of error instead of a somewhat arbitrary constant.

Using the above modifications, the results for both second and third order anisotropic adaptation are dramatically improved over uniform refinement. A comparison of third order velocity profiles is shown in Figure 8 In fact, without adapted refinement, the lift coefficients in Figure 9 do not appear to converge towards the solution of Venditti and Darmofal[41]. To show the accuracy of our second order and third order results, we have compared the pressure distribution on the upper surface of the airfoil with data from [1]. Not only does the shock apear in relatively the same location, but the shocks that we computed are extrememly sharp and have very little overshoot.

Subsonic laminar flow around an airfoil

Inviscid flows do not generally have high levels of anisotropy, other than possibly at shocks, so to better demonstrate the advantages of anisotropic meshing we will examine viscous subsonic flow around the NACA 0012 airfoil (Re = 5000, $M = 0.5$, $\alpha = 0°$). Our results for

Method	Final Mesh Size	$C_{D,P}$	$C_{D,\nu}$	C_L	$\frac{x_{sep}}{x_{chord}}$
3rd Order Anisotropic	15464	0.0225	0.0323	0.00004	0.811
3rd Order Isotropic	15682	0.0227	0.0342	0.00414	0.910
3rd Order Solution / 2nd Order Metric	20635	0.0224	0.0323	0.000487	0.805
2nd Order Anisotropic	20187	0.0226	0.0323	0.000114	0.795
2nd Order Isotropic	20642	0.0224	0.0332	0.0095	0.834
ARC2D	320 x 128 cells	0.0221	0.0321	–	0.824
Mavriplis [19]	320 x 64 cells	0.0229	0.0332	–	0.814
Radespiel [33]	512 x128 cells	0.0224	0.0330	–	0.814

Table 1: Comparison of drag, lift, and separation point for NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

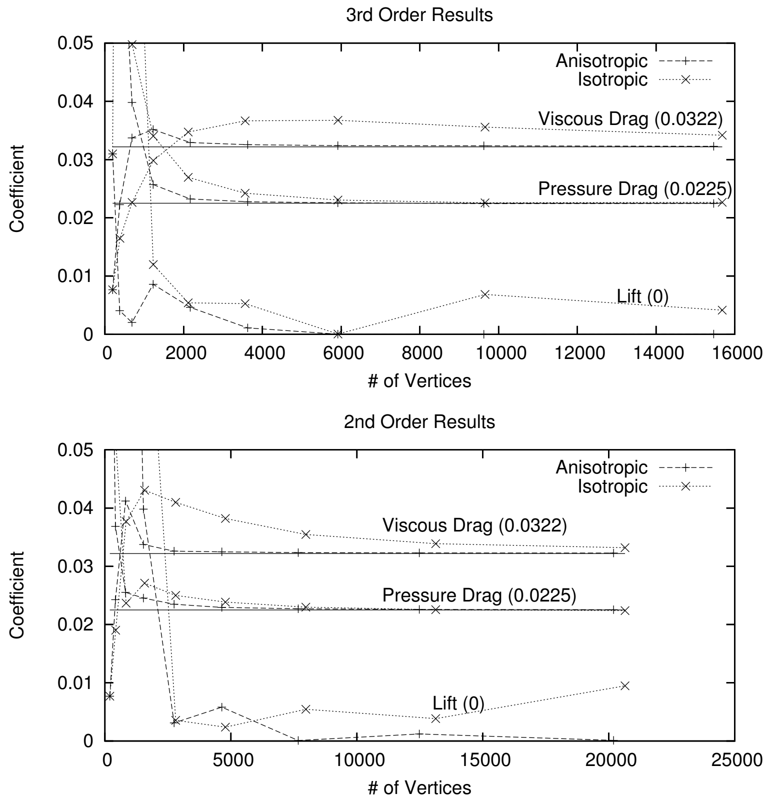


Figure 10: Anisotropic vs. Isotropic drag and lift convergence for NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

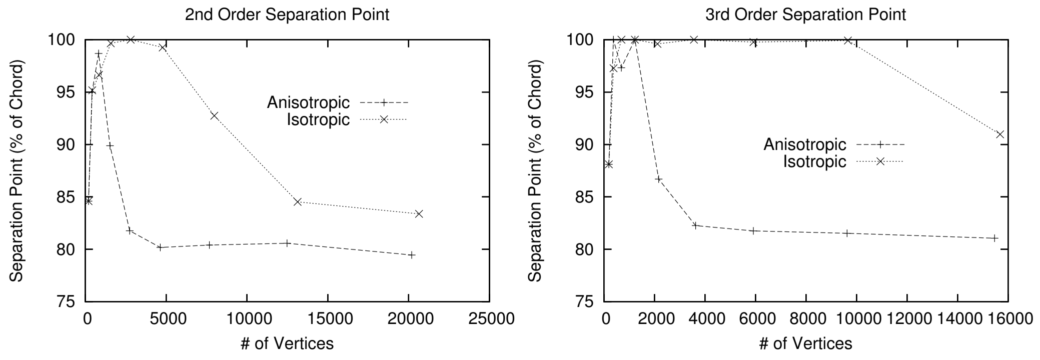


Figure 11: Anisotropic vs. Isotropic separation point convergence for NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

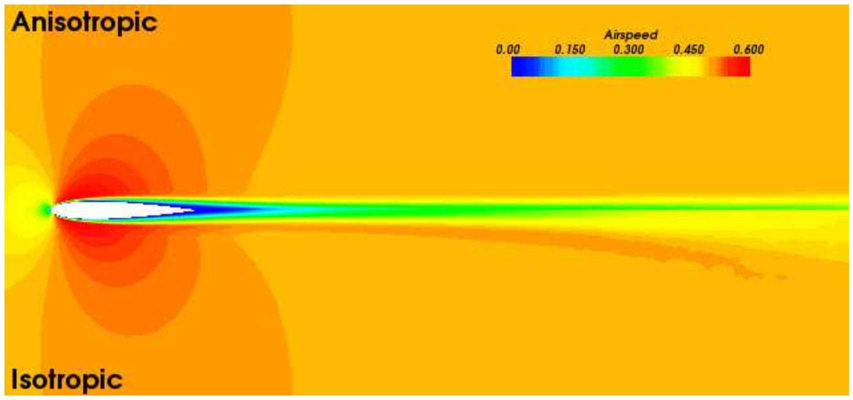


Figure 12: Comparison of anisotropic and isotropic third order Mach profile for NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

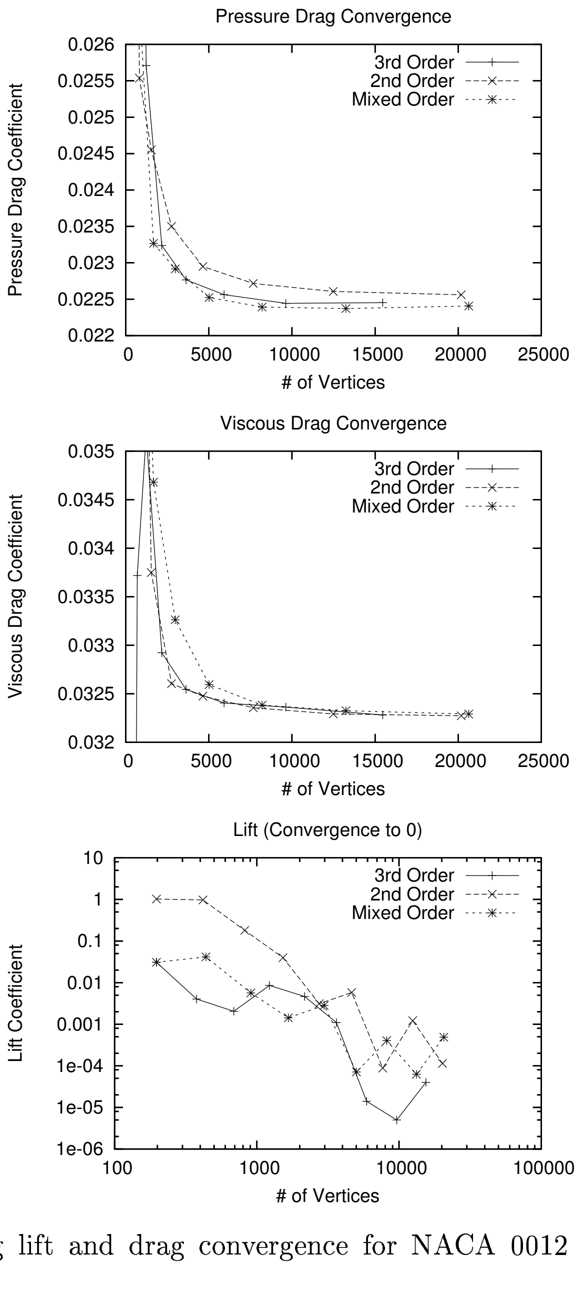


Figure 13: Comparing lift and drag convergence for NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

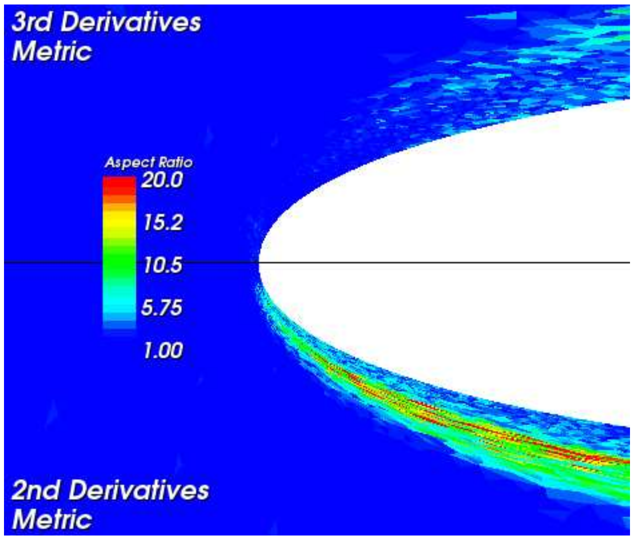


Figure 14: Triangle aspect ratios from 3rd order and 2nd order metrics at leading edge of NACA 0012 airfoil: $Mach = 0.5$, $Re = 5000$, $\alpha = 0$.

second and third order solutions with various metric parameters are presented in Table 1 along with other published results for the same set of parameters. Both the second and third order solutions are clearly converging to similar values, and those values lie within the range of published results.

To show the benefits of anisotropic meshing over isotropic meshing, we set the a_2 and b_2 Fourier coefficients from Equation 12 to zero. This allows the metric to vary in size but not shape. Figure 10 shows that by doing this, the convergence is dramatically reduced in both second and third order solutions for all three functionals. Nowhere is this more apparent than in the convergence of separation point in Figure 11 where the isotropic solutions separate much later than expected. The most noticeable difference in solutions between the anisotropic and isotropic meshes is the much more diffused wake as illustrated in Figure 12.

Dompierre et al.[2] suggest that Hessian based refinement should be suitable for high-order solutions even though the second derivatives are already resolved. To test this theory, we used our 2nd order metric to refine for a third order solution. The resulting convergence plots are shown in Figure 13. It does appear that there might be some benefit to using the third order metric for third order solutions, particularly in viscous drag convergence. However, there is not a significant enough difference between the two results to be conclusive. The only qualitative difference between the mesh generated by the 2nd order metric and the third order metric is that the third order metric is much more isotropic at the leading edge of the airfoil. This is shown in Figure 14. After the leading edge, both meshes become very anisotropic in the boundary layer and have roughly the same size distribution. More precise meshing would be necessary to examine what effect the different metrics have on the solution accuracy.

Conclusions

We have presented a general framework for anisotropically refining unstructured meshes. Our approach is designed to be sufficiently general to allow refinement for solvers of any order and any number of dependent variables. The basis behind our approach is to approximate the local reconstruction error with Taylor expansions and then use Fourier transforms to choose an appropriate metric function. The metric can then, in turn, be used to refine the mesh and improve the solution.

We have shown through several test cases that this refinement procedure improves solution accuracy better than uniform mesh refinement. We have also shown that for both second and third order viscous solutions, anisotropic refinement produces much better convergence and is therefore more efficient than isotropic refinement.

The differences between our second order and third order solutions are more difficult to interpret. In general, the third order solutions were more accurate than the second order solutions, but it remains unclear whether that warrants the additional computational cost. Our results are also inconclusive as to whether or not it is acceptable to use a metric based on second derivatives for third order solutions.

While our metric is designed to work for problems of any dimension, we have only tested it for two dimensional flow. This is due to current limitations of three-dimensional anisotropic meshing. Examining the effect of this refinement approach for higher-order three-dimensional solutions is an item for future work.

Another potential improvement would be to combine our metric with adjoint-based error estimation. Venditti and Darmofal [41, 42] clearly demonstrated that adjoint based-methods provide greater accuracy for the chosen functional than Hessian based methods. It is likely that this remains the case for higher-order methods as well. Combining adjoint based adaptation with our local shape definition seems an intuitive extension, especially for highly anisotropic solutions. This would still leave the problem of choosing the proper functional, which is often but not always obvious for engineering applications.

One problem with our method is that id does not provide a quantitative estimate of solution error. as with [3, 41, 42, 13]. Instead, we repeatedly refine our mesh so as to estimate error by convergence. Ultimately, we would like to have a better error estimate at each stage of the refinement process. Essentially, this would combine the two approaches and allow for better user control.

Ultimately, the goal of our work is to allow for fast, accurate solutions to CFD problems with a minimum amount of input from the user. While we have shown that more accurate solutions can be generated, it is not yet clear whether or not this approach is indeed faster. Even without examining user input, it is very difficult to compare solution time on a uniformly refined mesh with solution time for an adapted mesh, primarily because of the added complexity of choosing when to refine the mesh and by how much. An added complexity with anisotropic meshing is that its effect on matrix conditioning, and by extension matrix acceleration methods, is unclear.

Overall, there still remains insufficient data to determine whether or not our method for anisotropic adaptation will truly improve the efficiency of CFD solvers. What we have done is to provide a simple and effective way of combining two previously uncombined developments: high order methods with anisotropic adaptation. Hopefully, as further research is done into improving high-order CFD solvers and anisotropic mesh generators, the advantages of combining the two will also significantly increase. If so, there is no doubt that high order anisotropic adaptation will soon be commonplace among commercial CFD code.

Delaunay-Based Anisotropic Mesh Refinement

Introduction

Numerical solution of PDEs is an important analysis tool for scientists and engineers. Ideally, such numerical software should require only specifying the physics, geometry, and boundary conditions of the problem and then produce a solution of a certain accuracy. Unfortunately, assessing and minimizing discretization error is a non-trivial task, requiring time consuming mesh dependence analysis or computation of solutions on meshes that are much finer than the required solution accuracy dictates. Mesh adaptation seeks to automate this process by first computing the solution on a coarse mesh and then successively refining the mesh so that each mesh is more nearly optimal for the solution being computed. Unstructured meshes are well-suited for adaptation because they can be modified locally much more easily than structured meshes, which have a fixed topology. In this paper, we present an anisotropic extension of an existing Delaunay adaptation scheme [5, 44] and we show that it is possible to produce high quality anisotropic meshes without costly local maximization routines.

We have chosen to use Delaunay methods because Delaunay mesh refinement can easily be used to insert new vertices into a valid mesh. Delaunay methods therefore extend more naturally to anisotropic mesh refinement then Quad-Tree or Advancing Front methods. Shewchuck and Labelle [16] have developed a theoretically sound extension of guaranteed-quality Delaunay meshing by constructing anisotropic Voronoi diagrams and then dualizing to a Delaunay triangulation. Unfortunately, these anisotropic Voronoi diagrams can be difficult to generate, and the dualization to a valid triangulation requires restrictions on metric smoothness.

Alternatively, there are other more heuristic methods for generating anisotropic meshes that rely on local optimization algorithms. Dompierre et al.[13] suggest splitting edges with length[5] above a threshold, removing nodes with all edge lengths below a threshold, and swapping edges and moving nodes to equidistribute length. New edges are assigned lengths based on an interpolated error estimate from the background mesh. Buscaglia and Dari[8] take this a step further by testing a large set of potential operations on a sub-mesh and then selecting the one that most increases quality according to an error estimate on a background mesh; this approach can produce excellent meshes, but at a high cost. Our algorithm combines the heuristic nature of these two approaches with some of the attributes of isotropic Delaunay meshing. This gives us two principle advantages over the other heuristic methods. First, we do not rely on moving vertices and so metric information can be easily retained without storing and referencing the original mesh. Second, because of an improved choice of where to insert vertices, our algorithm produces quality anisotropic meshes without costly local maximization routines.

To explain our algorithm we will first examine metric spaces and how they can be used for anisotropic meshing. Then, we will create a quality measure for our anisotropic mesh and modify several existing mesh operations to increase that quality measure. Finally, we will show that this process leads to meshes that are well suited to the desired metric.

Metric spaces

Before defining what a quality anisotropic mesh is, we must first define the desired cell sizes and shapes. To do this, we use a function called a metric. Essentially, a metric is a way of defining distance between two points. The exact requirement of a metric for a vector space

[5]The term error is used in the cited paper, however the metric length can be defined as equivalent to the error and so we will use the term length to be consistent with usage later in the paper.

X is that $d : X \times X \rightarrow \mathbb{R}$ is a metric iff the following conditions are satisfied for every $x, y, z \in X$.

$$\begin{aligned} d(x,y) &> 0 \iff x \neq y \\ d(x,x) &= 0 \\ d(x,y) &= d(y,x) \\ d(x,z) &\leq d(x,y) + d(y,z) \end{aligned}$$

For uniform isotropic meshing, the metric is the standard distance measure $\|x - y\|$. More complicated metrics allow for distance to be measured differently depending on orientation, thus introducing anisotropy. In particular, the metric most commonly used in anisotropic meshing can be found by linearly transforming the standard distance measure $d_{\text{iso}}(x, y)$ into one which is anisotropic $d_{\text{ani}}(x, y)$ as follows

$$\begin{aligned} d_{\text{iso}}(x,y) &= \sqrt{V^T V} \\ d_{\text{ani}}(x,y) &= \sqrt{(TV)^T (TV)} \\ d_{\text{ani}}(x,y) &= \sqrt{V^T (T^T T) V} \end{aligned}$$

Where $V = x - y$ and T is an invertible linear transformation. Since a triangle is perfectly defined by the lengths of its sides, any triangle quality measure used in isotropic meshing has an equivalent triangle when measured by the metric. For example, if a triangle's edges have metric lengths (a, b, c) then the area A and circumradius R of a triangle can be calculated by the following formulas

$$\begin{aligned} A &= \frac{1}{4}\sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)} \\ R &= \frac{abc}{4A} \end{aligned} \tag{15}$$

Although we use the linear transformation metric throughout the article, our algorithm is based purely on vertex spacing and could be adapted to use any metric.

For most problems of interest, different metrics must be used in different areas of the domain. This is the principle difficulty faced by anisotropic meshing algorithms because even if the metric function is valid at a specific point, the interpolated metric space between any two points might not be valid. We deal with this problem by defining the metric discretely at vertices. When refining the mesh, we measure the length of each triangle's edge according to the average metric of the three vertices. This provides a good estimate of how big that triangle is and whether or not it requires insertion. If a vertex is inserted, the metric at that vertex is assigned based on a linear interpolation from the three vertices of the triangle in which it was inserted. Alternatively, when coarsening the mesh we estimate the length of each edge by first calculating what that length is according to the two adjacent triangles and then averaging those lengths. These two definitions give unique triangle quality and edge length measures which can then be used for mesh improvement operations.

Quality Measures

There are a staggering number of triangle quality measures used to determine how good the elements of a mesh are. Since no one has adequately proven that one quality measure is superior for every numerical solver, we have chosen to use the quality measure that best suits the approach we have taken. Miller, Talmor and Teng [21] pointed out that the most natural quality measure for Delaunay refinement is the ratio of circumradius to shortest edge. We will therefore define a *quality space*

$$(\xi, \eta) \quad = \quad \left(\frac{3^{\frac{1}{4}}}{2\sqrt{A_{\text{des}}}} l_{\text{min}}, \frac{3^{\frac{3}{4}}}{2\sqrt{A_{\text{des}}}} R \right) \doteq \left(\frac{0.658}{\sqrt{A_{\text{des}}}} l_{\text{min}}, \frac{1.140}{\sqrt{A_{\text{des}}}} R \right) \qquad (16)$$

where l_{min} is the length of the shortest edge, R is the circumradius, and A_{des} is the desired area. For a uniform isotropic mesh, the ideal triangle is an equilateral triangle with area

$A_{\text{des}} = \frac{A_\Omega}{N_v}$ where A_Ω is the total area of the domain Ω and N_v is the number of triangles; this triangle has $(\xi, \eta) = (1,1)$ for our chosen constants. We use this same concept to choose A_{des} in Equation 16. To estimate the area A_Ω of the domain Ω in the metric space, we sum the area of individual triangles. Then, the ideal area A_{des} is found by dividing the total area A_Ω by the number of triangles N_v and a desired refinement ratio r, so that $A_{\text{des}} = \frac{A_\Omega}{rN_v}$. The equilateral triangle with this area would be located at $(\xi_{\text{des}}, \eta_{\text{des}}) = (1,1)$ in the quality space.

Having defined the perfect triangle, we must now define what range of triangle qualities are considered acceptable. To accomplish this, we define a value τ such that triangles with $\xi < 1-\tau$ or $\eta > 1+\tau$ are considered bad. To ensure shape quality, we restrict τ to so that we have a lower bound on the minimum angle of the triangle, which can be written as $\frac{l_{\min}}{R} = 2\sin\theta_{\min}$. Eliminating $l_{\min}$ and R in favour of ξ and η, we have

$$\frac{\xi\sqrt{3}}{\eta} = 2\sin\theta_{\min}$$

and so for the corner point $(1-\tau, 1+\tau)$ in Figure 15,

$$\frac{1-\tau}{1+\tau} = \frac{2}{\sqrt{3}}\sin\theta_{\min} \tag{17}$$

$$\tau = \frac{3-2\sqrt{3}\sin\theta_{\min}}{3+2\sqrt{3}\sin\theta_{\min}} \tag{18}$$

So choosing θ_{min} is equivalent to choosing τ. Chew's uniform isotropic Delaunay refinement algorithm[11] produces triangles whose angles are all greater than thirty degrees so we have chosen to use this as the target angle giving $\tau = 2 - \sqrt{3} \doteq 0.2679$. An illustration of the quality space is found in Figure 15.

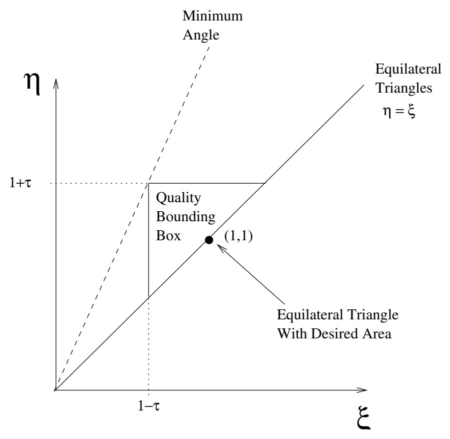


Figure 15: The quality space used to evaluate triangles in the mesh.

Mesh operations

There are three principal operations that our algorithm uses to improve the quality distribution of triangles in the mesh. The first two operations, edge swapping and point insertion, are similar to their isotropic Delaunay equivalents with modifications to account for the metric space. The last operation, vertex removal, is done simply to remove the smallest edges in the mesh.

Swapping

The first and simplest operation is edge swapping. For isotropic Delaunay meshing algorithms, edge swapping is used to ensure that the interior of every triangle's circumcircle is empty of vertices. For our discrete representation of the metric, this is not necessarily possible. Furthermore, whenever an edge is swapped, the metric used to examine the new triangles can differ from the metric of the old triangles, possibly reversing the swapping decision. Both of these problems are solved using a heuristic approach. Instead of swapping for cells with non-empty circumcircles, we instead swap to maximize the minimum quality ratio ξ/η for each pair of adjacent triangles. For an isotropic metric, this is equivalent to maximizing the minimum angle, which produces a Delaunay triangulation. To determine which configuration does this, we calculate (ξ, η) explicitly for both the current triangles and the triangles that would be formed if the edge were swapped. Note that our metric definition is dependant on which vertices are connected and so swapping an edge effectively changes the metric. Because of this, a Delaunay triangulation cannot be found by simply mapping the local vertex locations to the metric space.

After swapping, we check whether the resulting triangles are Delaunay in the metric space. The reason we do this is that Delaunay insertion is only valid when the interior of every circumcircle is empty. There are no guarantees that swapping will produce such a triangulation in our metric space, so we must identify non-Delaunay triangles so that we can handle them heuristically during insertion. Before we can check whether or not the edge is Delaunay in the metric space, we must scale one of the triangles so that the shared edge is the same length for both triangles. At this point, the relative locations of the four vertices are known in a pseudo-metric space. Then, we test whether or not the circumcircles of each triangle are empty. Because the Delaunay property is independent of size, it does not matter which triangle gets scaled. In Figure 16, the triangles T1 and T2 are first transformed to have the lengths assigned by their respective metrics. Then T2 is scaled by a/d so that the edges can be aligned and the circumcircles tested.

Vertex Insertion

The second mesh modification operation that we use is vertex insertion. In isotropic Delaunay refinement, vertices are inserted for triangles with large circumradii or poor shape. After insertion, the resulting triangles have smaller circumradii, and no edge smaller than the original circumradius is created. In our anisotropic adaptation, we hope for similar behaviour by inserting at an approximate circumcenter. We find this approximate circumcenter by creating a virtual sub-mesh around the initial triangle. To construct this virtual sub-mesh, we first transform the initial triangle to one with lengths as described by the metric. Then, we calculate that triangle's circumcenter and build the virtual sub-mesh one triangle at a time until the circumcenter lies within it. Since the metric length of a shared

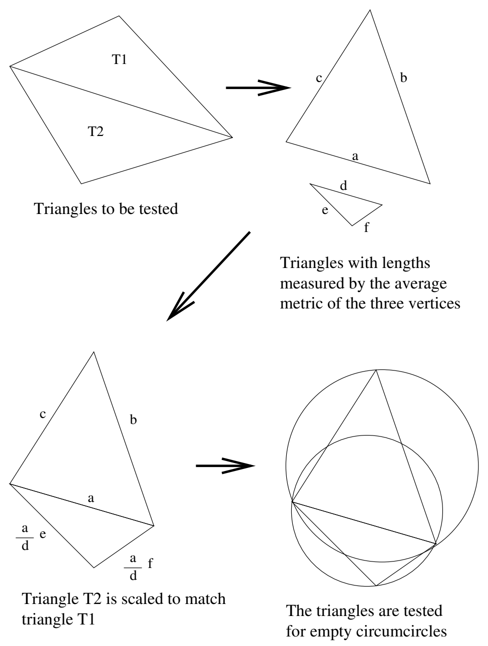

T1
T2
Triangles to be tested
c
b
a
d
e
f
Triangles with lengths measured by the average metric of the three vertices
c
b
a
$\frac{a}{d}$ e
$\frac{a}{d}$ f
Triangle T2 is scaled to match triangle T1
The triangles are tested for empty circumcircles

Figure 16: Testing for Empty Circumcircles in the Metric Space

edge is not the same for both triangles, a scaling factor is again required. This is similar to the Delaunay test procedure done for edge swapping. In Figure 17 Triangle T1 is chosen for insertion and the circumcenter is calculated in the virtual space. Then, Triangles T2 and T4 are appropriately scaled so that the virtual sub-mesh contains T1's circumcenter. Once that insertion location is chosen in the virtual sub-mesh, it is linearly transformed back to the real mesh so that it has the same barycentric coordinates in both the real and the virtual triangle. The new vertex is inserted with the most simple connectivity and swapping is performed recursively. The desired effect of each insertion is to decrease circumradii without dramatically reducing the quality of the local triangles. Unfortunately, due to the discrete nature of the metric, there are some cases where this approach fails.

The first degenerate case occurs when the circumcircle of the triangle we wish to eliminate by insertion is not empty in the virtual mesh. When this occurs, we instead split the longest edge of that triangle. There is theoretically no limit on the minimum edge that this operation will create. To address this, we smooth the new vertex to maximize the minimum angle in the virtual mesh, resulting in a much better insertion location. A similar approach is used

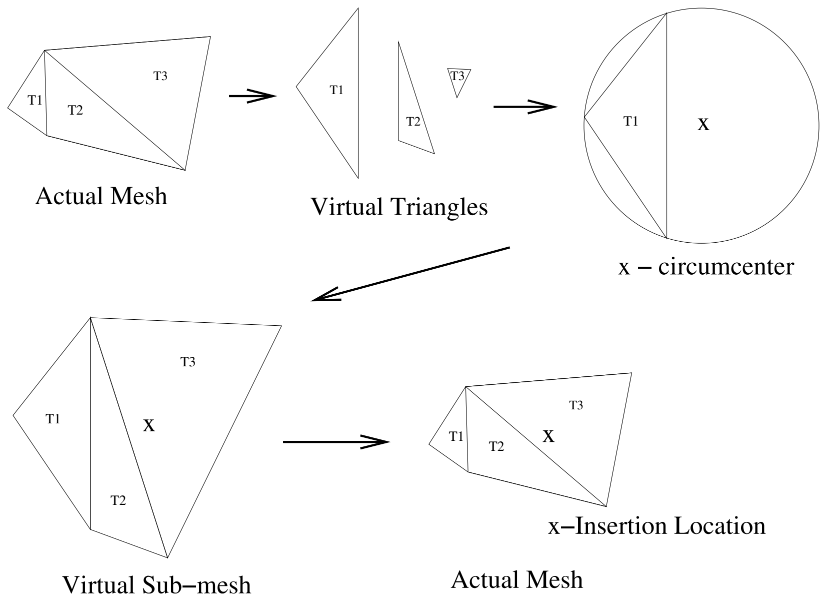


Figure 17: Finding an approximate circumcenter of triangle T1 by constructing the virtual mesh

in other anisotropic refinement algorithms [2, 8]. In our algorithm, however, optimization is only required for degenerate cases, which are a small fraction of the total number of insertions.[6]

The second degenerate case occurs when inserting the new vertex does not affect the triangle for which it was inserted. In other words, insertion did not reduce the largest circumradius in the mesh. This happens most often when there is large difference in metric size between triangles. Currently, insertion is simply repeated for the original triangle. Since that triangle will no longer have an empty circumcircle, refinement will split the longest edge and then locally optimize the location.

Vertex Removal

The final operation that we use to improve the mesh is coarsening. Coarsening is not a natural component of Delaunay meshing so there are no standard isotropic techniques to

[6]The exact percentage of degenerate cases is entirely dependent on how the metric varies within the mesh. Experience has shown that degeneracy in 5-10% of insertions is typical for the metric we used.

apply on the virtual mesh. Instead, we know that removing vertices increases edge lengths in the mesh. Thus, removing a vertex that is part of the smallest edge will always increase the minimum edge. To remove a vertex, we first swap edges until there are only three edges incident on that vertex and combine the three incident triangles to form one triangle when the vertex is removed. The edges are then swapped recursively in the same manner described in Section . Coarsening is the only operation that can destroy metric information. Because of this, repeated refinement and coarsening tend to smooth the metric.

Boundary Protection

A key element of Delaunay meshing is boundary treatment. While the previous discussion of point insertion assumed that the circumcenter lies within the domain, that is not necessarily true. For cases when the circumcenter lies outside the domain or across a boundary, the virtual mesh cannot be completely constructed. When this happens, the boundary edge which terminated the virtual mesh is split. For cases where the circumcenter of a triangle lies inside the domain, but near the boundary, there are no constraints on the circumradius of the new triangle formed. To address this problem, we adapt the boundary protection proposed by Shewchuk[39]. That is, we disallow any insertion within the diametral lens of a boundary segment and instead split that boundary segment. For an anisotropic mesh, the diametral lens is approximated by transforming any potential boundary triangle using the average metric of the two boundary vertices.

A further complication with boundaries is that any curvature of the boundary must be preserved throughout refinement. If the curvature of the boundary is not preserved, then the geometry will not be converging to the modeled geometry, thus introducing unnecessary modeling error. This is also very important for high-order solutions because correct curved boundary representation is essential to achieving high order accuracy. Ollivier-Gooch and Boivin[5] solved this problem for isotropic meshes by limiting the curvature of the boundary discretization and thus enforcing that the diametral lens lie outside the boundary curve.

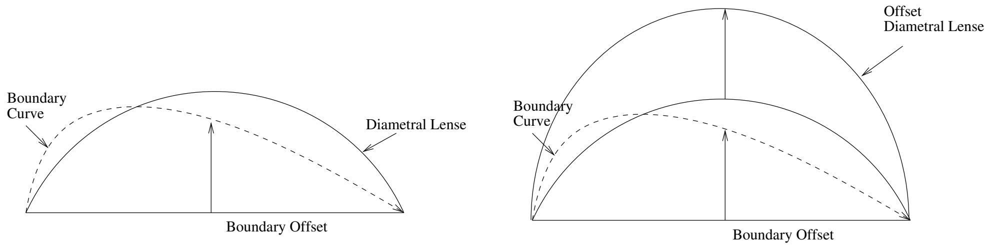


Figure 18: Isotropic, modified diametral lens for curved boundary

This is a much more difficult task when the metric is allowed to vary along the boundary edge because the measured curvature changes depending on what metric is used. Instead we rely again on heuristics: expansion of the diametral lens and encroachment checking before boundary insertion.

First, we expand the diametral lens to better account for the curvature. To do this, we first calculate the vertical offset between the linear boundary segment and the actual curved boundary. Then, we elongate the diametral lens of the segment by the same offset. An isotropic example of this is shown in Figure 18. Again, for the anisotropic metric, we perform this procedure based on the average metric of the two boundary vertices .

As with similar approaches in isotropic meshing, making this adjustment still does not guarantee that new vertices will be inserted within the domain or that splitting a boundary curve will produce good quality triangles. To remedy this, whenever a boundary edge is split, any vertex that encroaches on the new edges in the virtual mesh must be removed. Together, these methods ensure that the refined meshes will respect the boundary discretization and that no overly-flat triangles will be created along the boundary.

Algorithm

We apply the following algorithm to produce quality anisotropic meshes that match an arbitrary metric and have a prescribed maximum number of vertices.

1. Assign a metric to each vertex.

2. Sum the anisotropic area of each triangle and calculate A_{des} and thus (ξ, η) for every triangle.

3. Remove any interior vertices that encroach on a boundary segment according to the metric.

4. Recursively swap edges to minimize the ratio η/ξ. If this does not result in an anisotropic Delaunay triangulation, mark offending edges as being non-Delaunay.

5. Insert vertices for triangles with $\eta > (1 + \tau)$ until the maximum number of vertices is reached or there are no more triangles with large circumradii. Assign metrics to new vertices based on a linear interpolation from the three vertices of the triangle in which the vertex was inserted. If a face used in constructing the virtual mesh is marked as non-Delaunay, then the treatment for non-empty circumcircles from Section is applied.

6. Remove vertices for edges with $\xi < (1 - \tau)$ until there are no more small edges or sufficiently many triangles with large η are created. Limiting how many large triangles can be created is an effective way of reducing the overlap between refinement and coarsening. If this limit is a fraction of the number of vertices inserted in the previous refinement step then it adds a guarantee of termination irrespective of how many iterations are allowed.

7. Repeat the process of inserting and removing vertices until all triangles are within the acceptable tolerance, no more vertices can be removed, or a fixed number of iterations are performed. For quality meshes, the final process must always be mesh refinement.

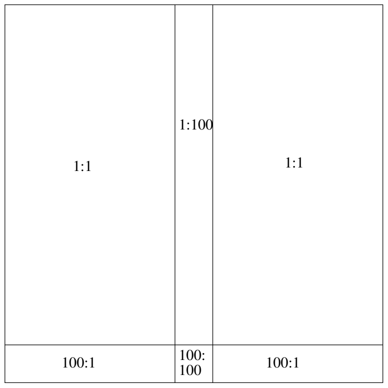


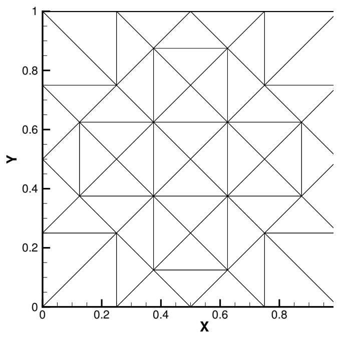
Figure 19: Expected Triangle aspect ratios given the metric from Equation 19.

Results

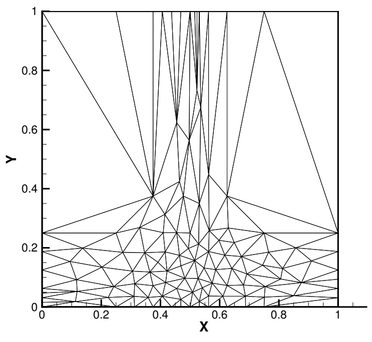
To demonstrate our algorithm's ability to create anisotropic meshes from a metric, the metric $d(\vec{x_1}, \vec{x_2})(x, y)$ defined by Equation 19 is assigned at every vertex.

$$\begin{aligned} d(\vec{p}_1, \vec{p}_2) &= \sqrt{(\vec{p}_2 - \vec{p}_1)^T M (\vec{p}_2 - \vec{p}_1)} \\ M(x,y) &= \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \\ a &= \begin{cases} 1 \times 10^4 & 0.475 < x < 0.525 \\ 1 & \text{Elsewhere} \end{cases} \\ b &= \begin{cases} 1 \times 10^4 & 0 \leq x < 0.05 \\ 1 & \text{Elsewhere} \end{cases} \end{aligned} \tag{19}$$

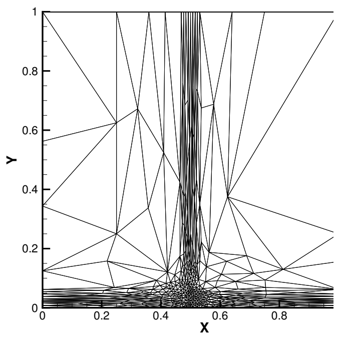
This metric should produce a mesh with distinct anisotropic regions. To simulate the adaptive procedure, we first assign the analytic metric to vertices of a very coarse mesh and then refine the mesh so that the metric-based area per triangle is approximately halved. Then, the vertices of the new mesh are again assigned the metric from Equation 19. We repeat this process four times to produce a mesh that is approximately sixteen times more refined then the original mesh and well representative of the analytic metric. The meshes produced from one iteration to the other are shown in Figure 20. To demonstrate that the



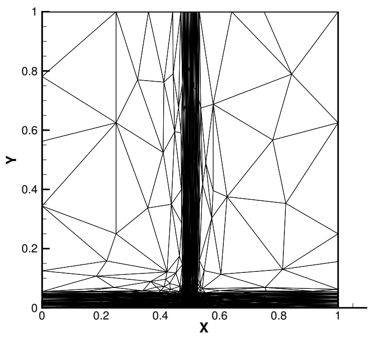
(a) Initial Isotropic Mesh (37 Vertices)



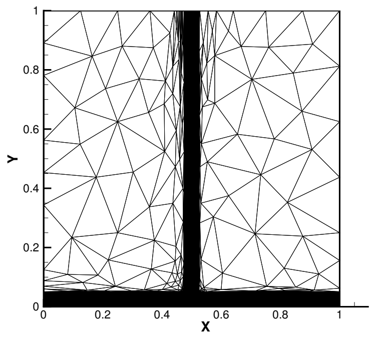
(b) First Refinement (117 Vertices)



(c) Second Refinement (364 Triangles)

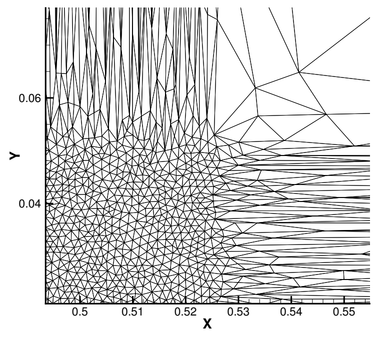


(d) Third Refinement (985 Vertices)

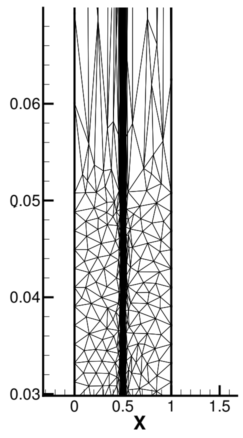


(e) Fourth Refinement (2423 Vertices)

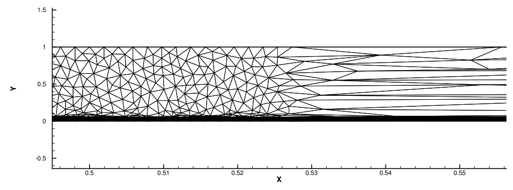
Figure 20: A square mesh being recursively refined according to the metric defined by Equation 19.



(a) 1:1 Axis Scaling

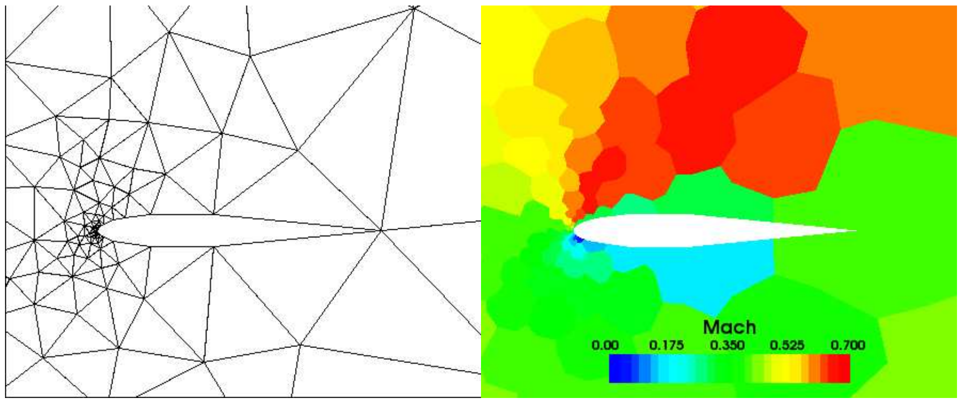


(b) 1:100 axis scaling



(c) 100:1 Axis Scaling

Figure 21: Mesh from Figure 20e viewed with different axis scalings.

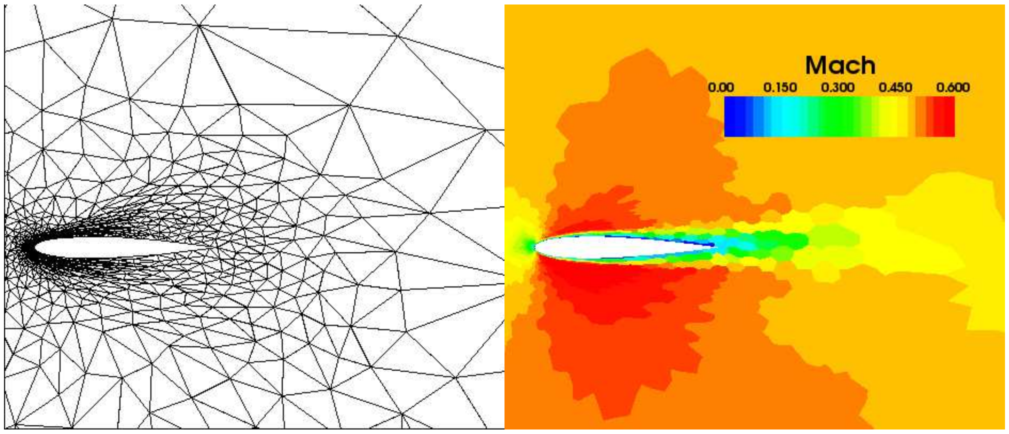


(a) Mesh with 407 Vertices (b) Flow Speed

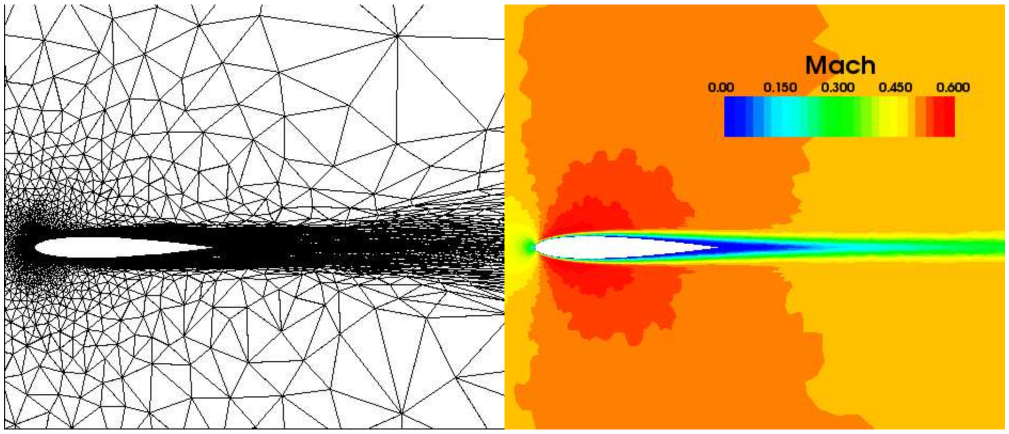
Figure 22: NACA 0012, Mach=0.5, Re=5000, 197 Vertices

mesh achieved the desired anisotropy we can view the mesh around the point $(0.525, 0.1)$ with different axis scalings. It is obvious in Figure 21 that the mesh has achieved the desired anisotropy because the mesh appears isotropic when certain regions are scaled accordingly.

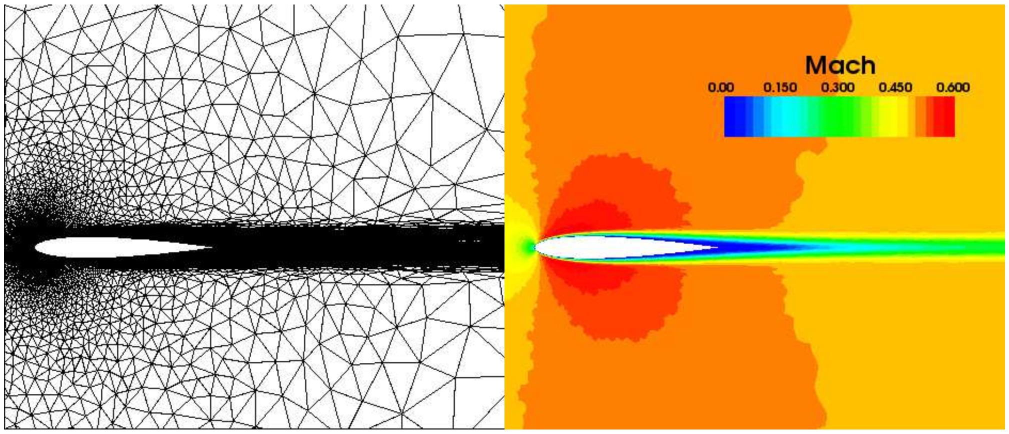
For a more practical example, and one involving curved boundaries, we examine laminar flow around the NACA-0012 airfoil. To compute the solution on each mesh, we use a second-order, vertex-centered finite-volume solver. This solver uses least-squares reconstruction [29], Roe's scheme [35], and Newton-GMRES for rapid convergence [20, 24]. Viscous terms are discritized as described in [29]. To calculate the metric between adaptation loops, we use the reconstruction error metric proposed in [31]. The initial mesh, shown in Figure 22a, is constructed using the isotropic mesh generator in GRUMMP [5, 26, 25]. This mesh is obviously ill-suited to computing viscous flow around the airfoil since refinement is done based only on boundary curvature. The asymmetry of the resulting solution in Figure 22b is evidence of how poor the mesh is. After choosing a metric based on this solution, we create a new mesh that should be better suited to the flow solution. We repeat this process for the three intermediate meshes in Figure 23 and the solution is improved with each successive refinement. After only four iterations of anisotropic refinement, the solution is dramatically improved. The final mesh and solution are illustrated in Figure 23. Close inspection of the stagnation point in Figure 24 shows that cell elements are small and



(a) 1018 Vertices



(b) 6430 Vertices



(c) 20137 Vertices

Figure 23: Intermediate Meshes and Solutions: NACA 0012, Mach=0.5, Re=5000

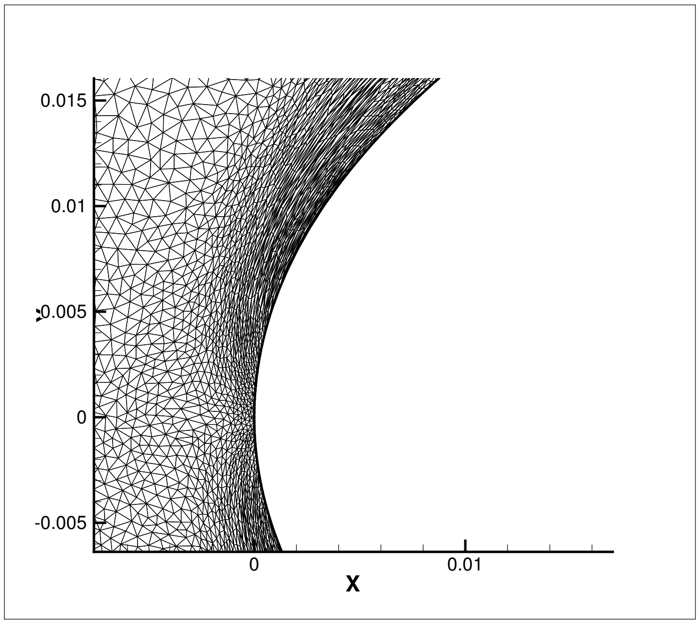
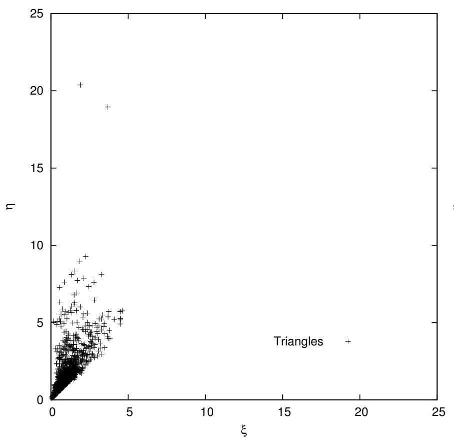


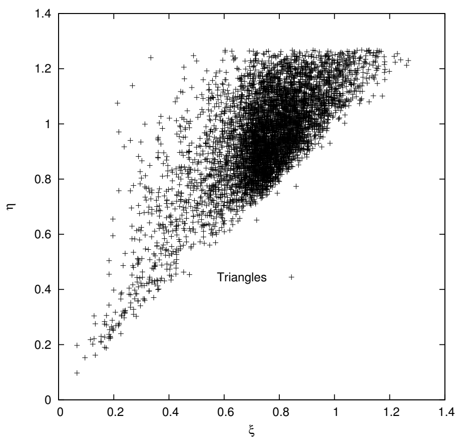
Figure 24: Close up view of the stagnation point of the finest mesh in Figure 23 a.

isotropic around the stagnation point and then they quickly become highly anisotropic as the boundary layer develops. Since the metric in this case is based on the Hessian of the flow variables, this is exactly the type of mesh that one would expect.

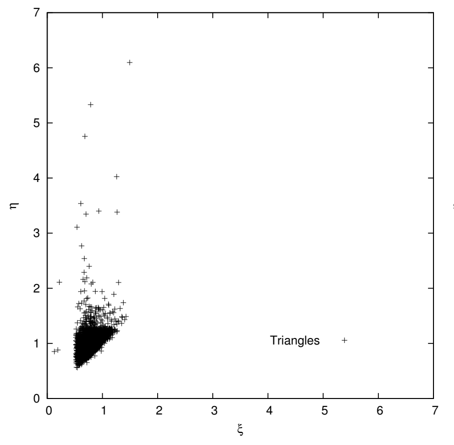
For each mesh adaptation, there is an inner series of refinement and coarsening loops that aim to improve the final mesh. To illustrate the effect of these inner iterations on the quality of the mesh, we have shown the distribution of the triangles within the quality space in Figure 25 for the intermediate mesh created in Figure 23a. It is obvious from the change between distributions a and b in Figure 25 that the initial refinement step is very effective at concentrating triangles within the the desired area. Unfortunately, it also produces many small edges and some poor angles. After coarsening is performed in Figure 25c, most of the triangles with small edges are removed but some triangles with very large circumradii are created. Those small edges that do remain are all located in regions of high boundary curvature. The next refinement again eliminates the large circumradii triangles without creating many small edges. The final two passes of coarsening and refinement have very little net impact on the quality of the mesh. While the final mesh contains some triangles



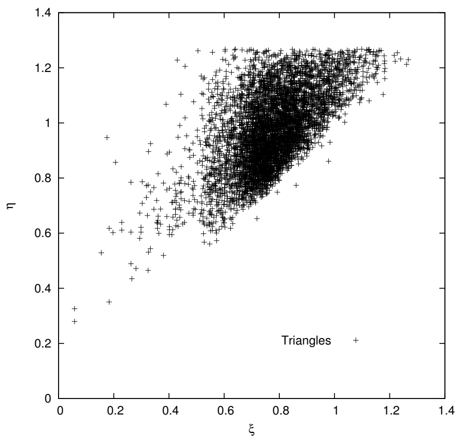
(a) Initial Mesh



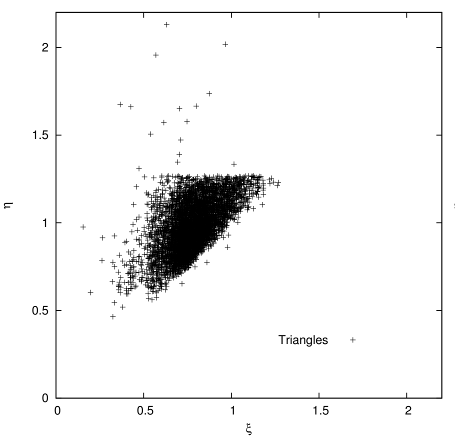
(b) First Refinement Pass



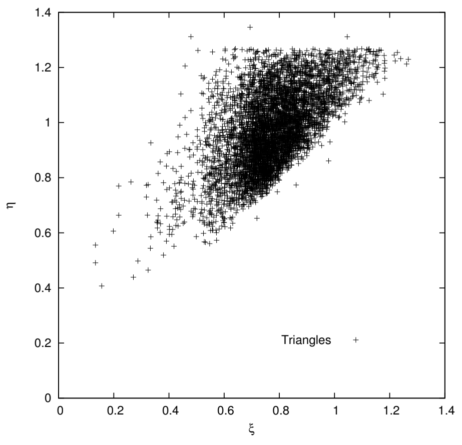
(c) First Coarsening Pass



(d) Second Refinement Pass



(e) Second Coarsening Pass



(f) Final Refinement Pass

Figure 25: Sample Quality Distribution During Refinement and Coarsening

outside the desired bounds, this is inevitable for a general geometry and metric.

Conclusions and Future Improvements

We have presented a method for anisotropic mesh refinement that produces high quality meshes without expensive local optimization routines. We have done this by assigning metrics to vertices and then defining triangle quality based on approximate lengths. The three mesh modification tools we used are swapping to maximize quality, inserting at approximate circumcenters to decrease circumradii, and removing vertices to eliminate small edges. Because there are no guarantees on the results of these modification tools, we use them iteratively to produce an optimal mesh. We have presented examples demonstrating that our algorithm produces meshes that are well adapted to the prescribed metric. When we combine this anisotropic adaptation with an effective metric estimator, we can recursively improve CFD solutions without the enormous increase in computation cost associated with uniform refinement.

There are also several improvements to our algorithm that could be implemented in the future. For example, we have stated that local mesh optimization is generally an expensive procedure. However, experience with isotropic meshing in GRUMMP has shown that if the mesh is already of high quality, local optimization-based smoothing can be a fast and effective way of further improving the mesh [14]. It is therefore our belief that the most efficient way of creating high-quality anisotropic meshes would be to insert and remove vertices according to our algorithm and then perform a limited number of passes over the mesh using local optimization. Since the mesh is already of high quality, many fewer passes over the mesh would be required to maximize the quality then if local optimization was the principle tool for mesh improvement.

Another potential improvement to our algorithm is a modification of the desired quality bounds in areas near boundaries with small features. The isotropic approach is to assign length scales based on the boundary geometry[39]. A similar approach might work for anisotropic meshing by decreasing the area of the desired equilateral triangle used in the calculation for $(\xi_{\text{des}}, \eta_{\text{des}})$ in Figure 15. This would likely reduce the number of instances where vertex removal creates poor quality triangles and thus it should reduce the need for repeated vertex insertion and removal iterations.

Finally, our meshing algorithm produces some small angle triangles that could be improved by further insertion. This would result in smaller triangles than desired, but the improvement in triangle quality might be worth the extra computation cost associate with more triangles. A final pass of quality-based refinement seems a likely way of addressing this problem.

Although many two dimensional meshing algorithms proclaim a straightforward extension to three dimensions, experience in three-dimensional isotropic meshing shows that this is rarely the case. However, since our algorithm is based on an heuristic adaptation to Delaunay meshing, it might be possible to apply the same general procedure in three dimensions and still produce quality meshes. This would be an ideal focus of future work.

Conclusions

Summary

In order to improve the accuracy of CFD solutions relative to the time required to generate those solutions, we have developed a method of refining meshes to equidistribute error. This method involves two separate but essential steps: creating a metric (Chapter) and then meshing the geometry based on that metric (Chapter).

To create the metric on which the refinement is based, we first estimate the error based on local reconstruction. That is, we assume that the error at each vertex is equal to the higher-order terms of the Taylor expansion. This requires reconstructing the un-resolved derivatives and then combining them according to some norm of the variables. We then evaluate this error estimate along a unit circle and calculate the first three even Fourier coefficients. Since we can choose matrix values such that the metric will have the same Fourier coefficients our metric will be a good approximation to the error measure.

Once the metric has been chosen for each vertex, we then refine our mesh so as to be uniform according to the metric. All of our meshing operations are based on a definition of quality that involves both the circumradius and minimum edge length according to the metric. The most simple operation, swapping, is done if it increases the quality of the worst triangle. For triangles that have large circumradii, we insert vertices at approximate circumcenters. This is very similar to the Delaunay approach for isotropic meshing. For triangles with small edges, we remove a vertex from that edge, thereby eliminating small edges. By repeatedly inserting and removing vertices, we are able to create high-quality meshes without costly minimization routines.

Results

By examining the flow around the NACA 0012 airfoil with several different parameters, we were able to demonstrate the benefits of anisotropic adaptation for both second and third order solutions.

In all the cases examined, the benefit of adaptive meshing over uniform refinement was obvious. That's because even for the finest uniform meshes that we used, some of the flow features are insufficiently resolved. In particular, the boundary discretization that we chose requires very little refinement at the trailing edge relative to the leading edge of the airfoil. Since resolution at the trailing edge is important to most flow solutions, the meshes that we created by uniform refinement are ill suited to the resulting flows. Similarly, by choosing a large farfield boundary, our uniformly adapted mesh had many cells in areas with virtually constant solution. Both of these problems are resolved by refining based on reconstruction error.

Our adaptive refinement algorithm is designed to equidistribute error between cells but also within each cell. This is accomplished by choosing a metric that is anisotropic and from that creating anisotropic cells. For inviscid flows, most error is isotropic and thus allowing for anisotropy had no noticeable effect on solution accuracy. For viscous flows, however, the presence of boundary layers leads to extremely anisotropic errors. By comparing anisotropic refinement with isotropic refinement for these flows, we were able to show that using anisotropic cells can dramatically increase solution efficiency. This is especially important when considering the separation point.

Because the reconstruction error is necessarily different depending on the order of the reconstruction, we hypothesized that high-order solutions require adaptation based on higher order derivatives. However, our results for third order solutions showed that refinement based on third derivatives is not convincingly better than refinement based on second derivatives. We believe this is due to the fact that regions with high third derivatives in general also have high second derivatives. Between errors in reconstructing the derivatives and approximations made in creating the meshes, the difference between the different order derivatives appears to be insignificant. While there exist trivial examples, such as the sine wave, where third order solutions should not be refined based on second order derivatives. Our comparison of metrics based on second and third order derivatives for third order solutions are inconclusive.

For every mesh that we adapted, our algorithm was able to create high quality anisotropically adapted meshes. This was done very quickly because the only two mesh operations we used were adding vertices in some regions and deleting some in others. This is much cheaper computationally than the local quality maximization algorithms that are more commonly applied.

Recommendations

We claim to have developed a method of anisotropic adaptation that can be applied to any Taylor-based CFD algorithm, however, we have only tested this approach for two dimensional finite volume, compressible flow around the NACA-0012 airfoil. Before the generality of our approach can be properly demonstrated, there remain a gamut of simulations to be performed. This would include using other solvers, other PDEs and more complicated geometries.

Eventually, the goal of this work is to show that accurate solutions can be generated faster using mesh adaptation rather than computing on a single fine mesh, no quantitative discussion of computation time is given in this thesis. This is because factors such as optimizing the adaptive cycle and the code itself require further analysis. In particular, the metrics used throughout the work were computed using a fully converged solution. Ait-Ali-Yahia et al. [2] explicitly say that full convergence is not required to generate improved meshes. Therefore, much computation time could be saved by determining when a sufficient convergence is reached for creating the metric. Another issue with computation time is that current incompatibility between the solver and the mesh generator prevents direct bit-wise object transfer from one program to the other. Instead, we are required to repeatedly read and write both the mesh and metric between programs. Having mesh modification done within the solver would substantially decrease computation time. Finally, there is a third issue of how much to refine the mesh between solutions. We have arbitrarily chosen to double the mesh density between iterations. If the metric is not very accurate, this could be too big a step between iterations and could negatively effect mesh optimality. If the metric (and the interpolation of that metric) is very precise, then perhaps a larger step would still produce optimal meshes and thus fewer steps would be required. All of these factors are essential to the efficiency of anisotropic adaptation and require analysis before our algorithms could be commercially applicable.

We have defined our metric so that it applies to any order solution, with any number of variables, in any dimension. While this general extension is theoretically sound, we have only tested it for second and third order solutions with four variables in two dimensions. The reasons for this are a lack of support in our solver for fifth order reconstructions, as well as the absence of high quality three dimensional anisotropic mesh generators. Future work should aim to create these structures so that the general applicability of this method can be better demonstrated. Furthermore, solver modifications such as turbulence modeling and slope limiting are bound to have an effect on the calculated metric and likewise the anisotropy of the mesh will influence the effectiveness of those modifications. A thorough analysis of these relations is also a useful avenue for further research.

With respect to mesh refinement, there are many minor improvements to the algorithm that could potentially increase the quality of the resulting mesh. This includes better support for small boundary features as well as a local optimization algorithm to be used after all the points have been inserted. As with most heuristic methods, there are several parameters in our meshing algorithm that can be tweaked for improved performance for a particular type of problem. In particular, the rate at which the metric is smoothed during swapping could dramatically change the resulting mesh even though the quality is still considered high. Proper analysis of this parameter would also require examining how beneficial smooth variation in cell size and shape is to the accuracy and efficiency of the solutions produced. Another parameter which affects the resulting mesh, and how long it takes to generate it, is the ratio of how many new large circumradius triangles can be created during a coarsening loop. Increasing this number improves the final measured quality of the mesh but decreasing it reduces the time required to create the mesh and also reduces the overall number of vertices removed. This also relates to the smoothing of the metric because removing vertices potentially removes metric information.

While the list of future work is large, both the metric computation and the mesh refinement algorithms are new approaches and show promise for future development. In particular, the idea that anisotropic adaptation can be applied to higher order solutions may contribute to the debate over whether or not high-order solvers are truly beneficial for most CFD applications. Regardless, there is no doubt that the advantages of anisotropic adaptation and high-order methods can be successfully combined and this paper contains a simple and effective way of doing so.

Bibliography

[1] AGARD Fluid Dynamics Panel. *Test Cases for Inviscid Flow Field Methods.* AGARD Advisory Report AR-211, AGARD, May 1985.

[2] D. Ait-Ali-Yahia, G. Baruzzi, W. G. Habashi, M. Fortin, J. Dompierre, and M. G. Vallet. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part II: Structured grids. *International Journal for Numerical Methods in Fluids*, 39(8):657–673, July 2002.

[3] R. Balasubramanian and J.C. Newman III. Comparison of adjoint-based and feature-based grid adaptation for functional outputs. *International Journal for Numerical Methods in Fluids*, 53:1541–1569, 2007.

[4] Timothy J. Barth and Paul O. Frederickson. Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction. AIAA paper 90-0013, January 1990.

[5] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1213, December 2002.

[6] Houman Borouchaki, Paul Louis George, Frederic Hecht, Patrick Laug, and Eric Saltel. Delaunay mesh generation governed by metric specifications. part 1: Algorithms. *Finite Element Analysis and Design*, 25:61–83, 1997.

[7] Houman Borouchaki, Paul Louis George, and Bijan Mohammadi. Delaunay mesh generation governed by metric specifications. part 2: Applications. *Finite Element Analysis and Design*, 25:85–109, 1997.

[8] Gustavo C. Buscaglia and Enzo A. Dari. Anisotropic mesh optimization and its application in adaptivity. *International Journal for Numerical Methods in Engineering*, 40:4119–4136, 1997.

[9] A. Bykat. Automatic generation of triangular grid: I - subdivision of a general polygon into convex subregions. ii - triangulation of convex polygons. *International Journal for Numerical Methods in Engineering*, 10(6):1329–1342, 1976.

[10] M.J. Castro-Diaz, F.Hecht, B. Mohammadi, and O.Pironneau. Anisotropic unstructured mesh adaptation for flow simulations. *International Journal for Numerical Methods in Fluids*, 25:475–291, 1997.

[11] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Dept. of Computer Science, Cornell University, 1989.

[12] Francois Courty, David Leservoisier, Paul-Louis George, and Alain Dervieux. Continous metrics and mesh adaptation. *Applied Numerical Mathematics*, 56:117–145, 2006.

[13] J. Dompierre, M. G. Vallet, Y. Bourgault, M. Fortin, and W. G. Habashi. Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part III. Unstructured meshes. *International Journal for Numerical Methods in Fluids*, 39(8):675–702, July 2002.

[14] Lori A. Freitag and Carl F. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[15] Weizhang Huang. Mathematical principles of anisotropic mesh adaptation. *Communications in Computational Physics*, 1(2):276–310, 2006.

[16] François Labelle and Jonathan Richard Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 191–200. Association for Computing Machinery, 2003.

[17] S. H. Lo. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering*, 21:1403–1426, 1985.

[18] D. L. Marcum and Kelly P. Gaither. Solution adaptive unstructured grid generation using pseudo-pattern recognition techniques. In *Proceedings of the Thirteenth AIAA Computational Fluid Dynamics Conference*, July 1997. AIAA 97-1860.

[19] Dimitri Mavriplis and Antony Jameson. Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes. AIAA paper 87-0353, January 1987.

[20] Krzysztof Michalak and Carl Ollivier-Gooch. Matrix-explicit GMRES for a higher-order accurate inviscid compressible flow solver. In *Proceedings of the Eighteenth AIAA Computational Fluid Dynamics Conference*, 2007.

[21] Gary L. Miller, Dafna Talmor, and Shang-Hua Teng. Optimal coarsening of unstructured meshes. *Journal of Algorithms*, 31(1):29–65, 1999.

[22] Amir Nejat. *A Higher-Order Accurate Unstructured Finite Volume Newton-Krylov Algorithm for Inviscid Compressible Flows*. PhD thesis, University of British Columbia, Department of Mechanical Engineering, 2007.

[23] Amir Nejat and Carl Ollivier-Gooch. On preconditioning of Newton-GMRES algorithm for a higher-order accurate unstructured solver. In *Proceedings of the Fourteenth Annual Conference of the Computational Fluid Dynamics Society of Canada*. cfdsc, 2006.

[24] Amir Nejat and Carl Ollivier-Gooch. A high-order accurate unstructured finite volume newton-krylov algorithm for inviscid compressible flows. *Journal of Computational Physics*, 227(4):2592–2609, 2008.

[25] Carl F. Ollivier-Gooch. An unstructured mesh improvement toolkit with application to mesh improvement, generation and (de-)refinement. AIAA 98-0218, January 1998.

[26] Carl F. Ollivier-Gooch. GRUMMP — Generation and Refinement of Unstructured, Mixed-element Meshes in Parallel. http://tetra.mech.ubc.ca/GRUMMP, 1998–2005.

[27] Carl F. Ollivier-Gooch. ANSLib: A scientific computing toolkit supporting rapid numerical solution of practically any PDE. In *Proceedings of the Eighth Annual Conference of the Computational Fluid Dynamics Society of Canada*, pages 21–28. Société canadienne de CFD / CFD Society of Canada, June 2000.

[28] Carl F. Ollivier-Gooch and Charles Boivin. Improved cell size and grading in guaranteed quality triangular and tetrahedral meshes. In *Proceedings of the Ninth International Meshing Roundtable*, pages 43–54. Sandia National Laboratories, October 2000.

[29] Carl F. Ollivier-Gooch and Michael Van Altena. A high-order accurate unstructured mesh finite-volume scheme for the advection-diffusion equation. *Journal of Computational Physics*, 181(2):729–752, 2002.

[30] Doug Pagnutti and Carl Ollivier-Gooch. Delaunay-based anisotropic mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, Submitted, 2008.

[31] Doug Pagnutti and Carl Ollivier-Gooch. A generalized framework for high order anisotropic mesh adaptation. *Computer Methods in Applied Mechanics and Engineering*, Submitted, 2008.

[32] Phillipe Pebay and Timothy J. Baker. Analysis of triangle quality measures. *Mathematics of Computation*, 72:1817–1839, 2003.

[33] R. Radespiel. A cell-vertex multigrid method for the Navier-Stokes equations. NASA TM-101557, 1989.

[34] Shmuel Rippa. Long and thin triangles can be good for linear interpolation. *SIAM Journal on Numerical Analysis*, 29(1):257–270, February 1992.

[35] P. L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[36] P. L. Roe. Characteristic-based schemes for the Euler equations. In *Annual Review of Fluid Mechanics*, volume 18, pages 337–365. Annuals Reviews, Inc., 1986.

[37] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, May 1995.

[38] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[39] Jonathan R. Shewchuk. *Delaunay Refinement Mesh Generation.* PhD thesis, School of Computer Science, Carnegie Mellon University, May 1997.

[40] D. A. Venditti and D. L. Darmofal. Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow. *Journal of Computational Physics*, 164(1):204–227, October 2000.

[41] D. A. Venditti and D. L. Darmofal. Grid adaptation for functional outputs: Application to two-dimensional inviscid flows. *Journal of Computational Physics*, 175(1):40–69, February 2002.

[42] D. A. Venditti and D. L. Darmofal. Grid adaptation for functional outputs: Application to two-dimensional viscous flows. *Journal of Computational Physics*, 187:22–46, 2003.

[43] M.A. Yerry and M.S. Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3:39–46, 1983.

[44] Pengtao Yue, Chunfeng Zhou, James J. Feng, Carl F. Ollivier-Gooch, and Howard H. Hu. Phase-field simulations of interfacial dynamics in viscoelastic fluids using finite elements with adaptive meshing. *Journal of Computational Physics*, 219:47–67, 2006.