Visualization for High-Order Discontinuous Galerkin CFD results

by

David Walfisch

S.B., Massachusetts Institute of Technology (2006)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2007
[September 2007]

Author .... [illegible]
Department of Aeronautics and Astronautics
August 23, 2007

Certified by .... [illegible]
Robert Haimes
Principal Research Engineer
Thesis Supervisor

Accepted by .... [illegible]
David L. Darmofal
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Visualization for High-Order Discontinuous Galerkin CFD results

by

David Walfisch

Submitted to the Department of Aeronautics and Astronautics
on August 23, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis demonstrates a technique that reduces the perceived error introduced in the visualization of higher-order DG solutions with traditional tools. Most visualization techniques work by using the solution at the mesh vertices and linearly interpolating to color the interior of the element. For higher-order solutions (where the visualization mesh is the same as the solution mesh) this method renders the high-order solution linear. By inserting new nodes at element quadrature points where the difference between the actual solution and the linear rendering is above a user-defined tolerance, additional elements are created and used for the visualization process. In order to keep the counts low for this new mesh, after each insertion a local rearrangement is performed to readapt the parent element so that the total visualization error is reduced.

The method introduced here has many advantages over isotropic adaptation used by some higher-order visualization techniques. Isotropic adaptation adapts all the elements regardless of error, thus creating a higher total element count and therefore requiring more memory and rendering time. In some cases isotropic elements are not ideal in representing the solution (*ie*: boundary layers, shocks, wakes, etc.). Lastly, by providing an option to define the maximum visualization error allows the user to specify how close the visualized solution is to the actual calculated one (at the expense of a denser visualization mesh).

Second, this work introduces a new method to apply an accuracy maintaining post-processor on DG vector fields to improve on the standard streamlining algorithms. Time integration methods do not work accurately and may even fail on discontinuos fields. The post-processor smoothens the field and eliminates the discontinuity between elements thus resulting in more accurate streamlines. To keep down the computational cost of the method, the post-processing is done in a one dimensional manner along the streamline.

Thesis Supervisor: Robert Haimes
Title: Principal Research Engineer

Acknowledgments

I would like to thank my advisor, Bob Haimes, for his continuous guidance throughout the course of this work. It was a real privilege to work with him. His incredible insight to scientific visualization was a major help in this research.

I would also like to express my gratitude to Jennifer Ryan and Mike Kirby for their help in the streamlining section of this work. Even though they are in other parts of the country, they were always available to answer questions and guide me through any challenges I ran into during my research.

I also want to thank the entire ProjectX team for their help. Being just a few feet away, Chris, Garrett, Tod, Laslo, JM and Josh were always ready to answer any question. They always took the time to help me tackle any new problems.

Lastly, I would like to thank my family for their support. My parents, Peter and Eva, my sister, Rita and my grandparents always encouraged me. They always let me follow my dreams even when these dreams led me to the other side of the world from them.

Contents

1 Introduction **15**

1.1 Current Visualization techniques . . . 15

1.2 DG Discretization of the compressible Navier-Stokes equations . . . 17

1.3 Simplex Cut Cells . . . 17

1.4 Visualization Error . . . 18

1.5 Streamline . . . 20

1.6 Contributions of this Thesis . . . 21

2 Visualization of High Order DG solutions **23**

2.1 Current Techniques . . . 23

2.2 New Adaptation Technique for Visualization . . . 25

2.2.1 Calculate Linear Solution . . . 26

2.2.2 InsertNode . . . 27

2.2.3 Swapping . . . 27

2.3 Simplex Cut Cell terminology . . . 29

2.4 Isotropic Refinement for Simplex Cut Cell Visualization . . . 30

2.5 Adaptation Technique for Simplex Cut Cells . . . 32

2.5.1 Algorithm for cut cell visualization . . . 32

2.5.2 Pre-process . . . 34

2.5.3 Modifications to Calculate the Linear Solution in multiply cut cells . . . 34

2.6 Visualization Results . . . 35

2.6.1 Body Conforming Results . . . 35

2.6.2 Cut Cell Results 43

3 Streamlining in DG fields **47**

3.1 Current Techniques 47

3.2 The uniform symmetric post-processor 49

3.3 Post-processing for accurate streamlining 51

3.4 Fully one-sided post-processor 52

3.5 Numerical integration for the post-processor 53

3.6 Ideal characteristic length 55

3.7 Choice of time-integration technique 59

3.8 Streamline results 60

4 Conclusions and Future Work **65**

4.1 Conclusions 65

4.2 Future work 66

List of Figures

1-1 Cartesian cut cell refinement around body boundary (figure used with the permission from [13]) 19

1-2 Reference, Original and Reference Shadow elements (figure used with the permission from [14]) 20

2-1 Node insertions in a parent that already has one step of refinement done on it (top left image). 28

2-2 Face Swapping in two dimensions. On the left a configuration when the face can be swapped. On the right, swapping would result in negative element area (shaded region). 29

2-3 Background element cut by the boundary creating the shaded cut cell. 30

2-4 Multiply Cut Cell at the trailing edge of an airfoil 30

2-5 Isotropic refinement gets rid of elements completely outside of flow domain 31

2-6 Quadrature points inside Cut Cells for a 2D NACA 0012 airfoil . . . 33

2-7 Visualization on the solution mesh (E_{vis} = .127) and the uniformly refined mesh (E_{vis} = .00113). Left side shows density contour lines while on the right the visualization mesh can be seen. 36

2-8 Results on the visualization-error based adapted meshes. Top was adapted to $E_{tol} = 10^{-2}$ and the bottom one with $E_{tol} = 10^{-3}$. 37

2-9 Single element adapted anisotropically along expansion fan 38

2-10 Results on the solution mesh (E_{vis} = .217). 40

2-11 Results on the isotropically refined ($E_{vis} = .00278$) and visualization-error based adapted meshes. The bottom two images were adapted with a $E_{tol} = .001$. 41

2-12 Results on the solution and the $E_{tol} = .001$ meshes. Left side shows density contour lines on a 2D cut plane while on the right the contour lines on the body are visible. 42

2-13 Results on the solution mesh $E_{vis} = .027$. Solution exists inside the airfoil but may be disregarded. 44

2-14 Results on the isotropically refined $E_{tol} = .0012$ and visualization-error based adapted meshes. The bottom two images were adapted with a $E_{tol} = .0012$. 45

2-15 Adapted cut cells 46

3-1 Convolution kernels for the $k = 1$ case. 53

3-2 Discontinuous integrand as seen by a streamline. 54

3-3 Continuous line is the real higher-order streamline, dashed line is built from linear segments 55

3-4 A sample streamline over the continuous field with a starting location of $(0, .3)$ 55

3-5 X velocity along the streamline (x-axis is in 10^{-4} arch-length coordinates). 56

3-6 Maximum error (in x velocity) vs. characteristic length (in arch-length). 56

3-7 Results with char length of .3 on the top and .05 on the bottom. . . . 57

3-8 Results with char length of .14 58

3-9 Streamline results on 2D transonic case. Dashed line is streamline without post-processing. Both streamlines were created with a $\Delta t = .01$ 60

3-10 Streamline results on 3D transonic case. Two different views of the wing and the streamline. 61

3-11 Streamline close to the airfoil. Separation can be seen towards the trailing edge. The x location marks the starting point of the streamlines on Figure 3-12. 62

3-12 Streamlines at the separated region. Streamline starting location is the same for both cases. $\Delta t = .1$ 63

List of Tables

2.1 Number of new elements introduced per insertion 27

3.1 Post-processed error when the characteristic length is set to the maximum element size. 58

Chapter 1

Introduction

1.1 Current Visualization techniques

Basic forms of Computational Fluid Dynamics (CFD) were established in 1966 with John Hess' and A. M. O. Smith's first paper on methods to solve the linearized potential equations. CFD began developing quickly in the 1980's as the improvement of computing and processor technology gave scientists the means to solve the complex fluid dynamics equations they already had. With these new solutions came the need for visualization: without being able to see the solutions it is significantly harder to understand them. As the technology further improved and CFD solvers started producing three dimensional results, the question arose on how to visualize something three dimensional on a computer monitor.

Currently OpenGL is the most commonly used graphics library for both two and three dimensional visualization. It was developed in 1992 by Silicon Graphics (SGI). OpenGL deals with linear display components, building the visualized image from polygons. While this method can be fast and efficient, it does not always produce accurate images.

Most visualization techniques for CFD use OpenGL to produce imagery. This is accomplished by using the solution at the mesh vertices and linearly interpolating to color the interior of the element. For higher-order solutions, if one uses the same visualization mesh as the solution mesh, the solution will be rendered linearly with loss of accuracy. Elements with high order geometric faces also get misrepresented during the process as they lose their curved nature. These problems in visualization may mislead the user to believe that either the solution is incorrect, or the solver itself has an error. Thus, it is crucial to be able to visualize a solution accurately as its interpolation order increases. For time-dependent solutions, it is also important to be able to do so quickly since the solution must be rendered at each time step. There are techniques which allow pixel exact visualization, such as ray tracing, but these techniques are relatively slow and have to be restarted every time the solution is rotated, zoomed in/out or moved around in any way.

Another significant problem is the calculation of streamlines in Discontinuous Galerkin (DG) fluid fields. A streamline can be defined by an ordinary differential equation (ODE) as it is done in Section 1.5. There are standard ODE solvers (time-integrators [25, 3]) available that help trace a streamline, but they were designed for continuous smooth fields. If, for example, a fourth-order Runge-Kutta (RK4) method is used on a DG fluids solution, its convergence would only be first order [8].

The discontinuity between elements may cause other significant problems for simple time-integration methods. For example, one can imagine a situation in which a streamline exiting one element sees a velocity solution pointing in the direction from which it just came. Such a situation would cause the streamline to get stuck on the element boundary with the time integrator in an infinite loop. One solution to avoid potential problems during the streamlining process is to use some type of post-processing prior to the time integration step. Most of these techniques use convolution with B-splines to smooth out oscillations and discontinuities while also increasing the accuracy of the solution [7, 30, 29].

1.2 DG Discretization of the compressible Navier-Stokes equations

The time dependent, strong form of the compressible Navier-Stokes equations is:

$$\partial_t u_k + \partial_i F_{ki}(\mathbf{u}) - \partial_i F^v_{ki}(\mathbf{u}) = 0, \qquad k \in |1, n_s| \tag{1.1}$$

In the above equation $\mathbf{u}$ represents the conservative state vector $(\rho, \rho u, \rho v, (\rho w,)\rho E)$, ρ is the density and E represents the total energy. F_{ki} is the inviscid while F^v_{ki} is the viscous flux components. Equation 1.1 contains the three conservation laws (mass, momentum and energy).

The calculation is done on a domain Ω which can be divided up by a triangular/tetrahedral mesh. The solution is a piecewise polynomial function in each element that satisfies the weak form of the above equation. A detailed overview of DG discretization of the Navier-Stokes equations, and the way to obtain the weak form, is provided by Cockburn and Shu [5] and Fidkowski et al [14].

1.3 Simplex Cut Cells

Most CFD solvers work by using one of two types of body-conforming meshes: structured or unstructured. Structured meshes do not store element connectivity data and thus take up less memory, but they require significant user involvement to generate. Conversely, unstructured meshes can be generated robustly even for complex geometries, but they must store element connectivity data, which also makes them slower. However, both types of meshes run into difficulties when higher-order elements are required by the solver (i.e.: body conforming higher order discretization methods [1]). Many issues arise in the generation of these types of grids which are further discussed in [13].

One of the most promising methods used to solve the meshing problems introduced due to the curved boundary requirement is the cut cell method. Purvis and

Burkhalter first introduced the cartesian cut cell method in 1979 [26]. This method used rectangular/box shaped cells, faces aligned with the coordinate system, to cut through the geometry of the body of interest. With these cut cells, the mesh no longer has to be body conforming, allowing the grid generation process to become automated. However, the intersection of the body with the background mesh must be calculated and the solver must be able to solve on the arbitrarily shaped cells which result from the cutting.

Cartesian cut cells solved the problem of automated mesh generation but added the constraint of isotropic background elements which may not be ideal for many fluid dynamics problems (i.e.: boundary layers, shocks, wake, etc.). Even though adaptive mesh refinement (AMR) [23] can be used to resolve the grid, it is not the most efficient method. The main problem with AMR is that refinement can not be done purely locally, and that all the elements have to be isotropic. Figure 1-1 shows a cartesian cut cell mesh adapted around a curved boundary in 2D.

In order to be able to introduce anisotropic adaptation, the background elements had to be changed from cartesian to simplex. Simplex cut cell method was introduced by Krzysztof J. Fidkowski in his doctoral thesis [13]. This method is similar to the cartesian method, but in the simplex method the background mesh is built from triangles in two dimensions and tetrahedra in three.

The development of simplex cut cells necessitated the ability to visualize the solutions derived from their calculations. This thesis presents two novel techniques for that purpose.

1.4 Visualization Error

As mentioned earlier, most visualization techniques are not pixel exact, especially when looking at higher order solutions. There is some error introduced due to the linear interpolation inside elements, but this error is easily quantifiable. The exact solution can be calculated anywhere in the domain with the use of the basis functions (Equation 1.2). The linearly interpolated solution can be calculated in the same man-

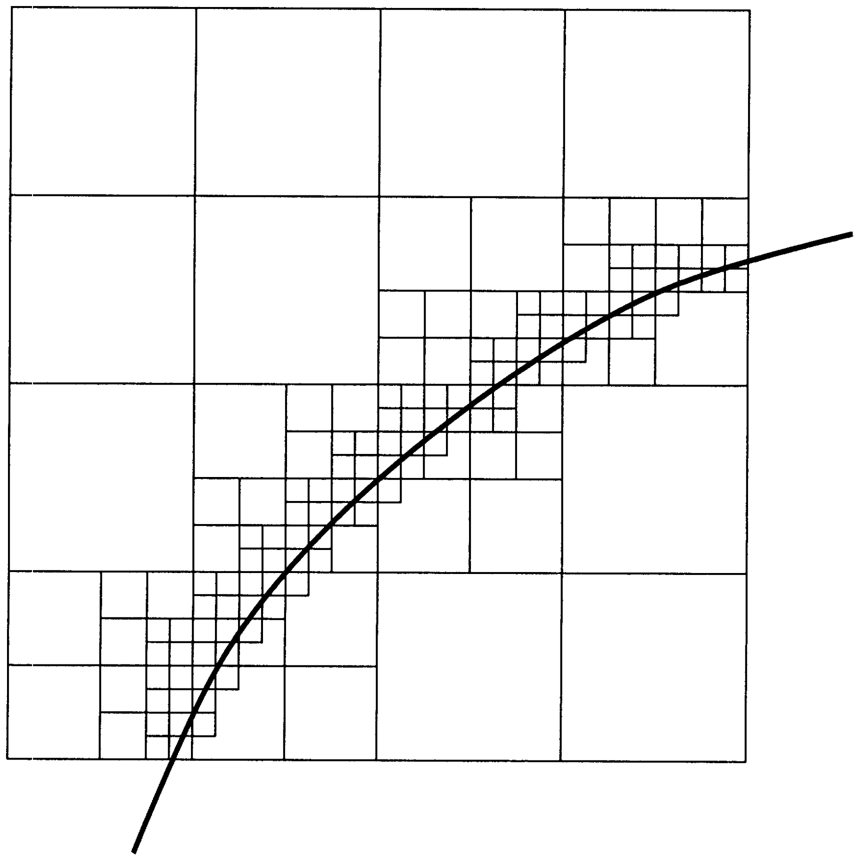


Figure 1-1: Cartesian cut cell refinement around body boundary (figure used with the permission from [13])

ner if the higher order basis functions are replaced by the linear function (Equation 1.3).

$$u(\xi_1', \xi_2', \xi_3') = \sum_{i=1}^{\binom{p+d}{d}} a_i \phi_i(\xi_1', \xi_2', \xi_3') \tag{1.2}$$

$$u_L(\xi_1', \xi_2', \xi_3') = \sum_{i=1}^{d+1} a_i \phi_i^L(\xi_1', \xi_2', \xi_3') \tag{1.3}$$

In the equation d is the problem dimensionality and p is the solution polynomial order. With the use of mapping (Equation 1.4), the coordinates of any point can be converted to shadow reference coordinates [14] for which the basis functions are defined. Figure 1-2 shows the reference, original, and shadow reference elements, and the mapping between each of them.

$$\vec{\xi'} = \mathbf{B}^{-1}\vec{x} \tag{1.4}$$

The error introduced during the visualization process is simply the difference between the real and the linearly interpolated solutions at every point $E_{vis} = |u - u^L|$.

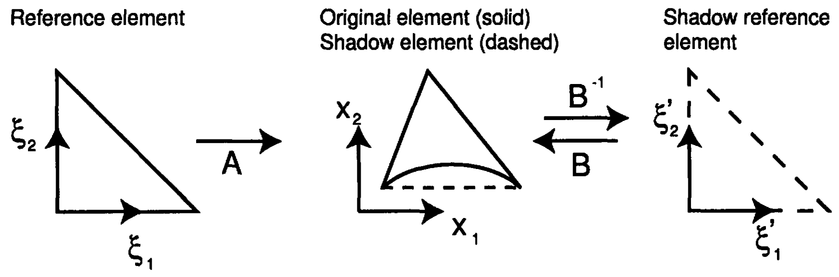


Figure 1-2: Reference, Original and Reference Shadow elements (figure used with the permission from [14])

1.5 Streamline

A streamline is defined as a curve with tangents pointing in the direction of the velocity vector at every point [18] at the same instance of time. The mathematical definition of a streamline is the following;

$$\frac{dr}{d\xi} = \vec{u} \tag{1.5}$$

In Equation 1.5, $\vec{u}$ is the velocity vector which can be written as $\vec{u} = (u, v, w)$ and r is the streamline. The directional derivative is defined as:

$$\frac{d}{d\xi} = \vec{u} \cdot \nabla \tag{1.6}$$

It can be seen that Equation 1.5 is a first order ODE. As such there are numerical methods available to get an approximate solution of a streamline on a given vector field with a given starting point. In CFD it is advantageous to be able to visualize streamlines in order to understand how the flow behaves around (or for some cases inside) a body.

1.6 Contributions of this Thesis

This thesis demonstrates an anisotropic visualization-error based mesh adaptation for higher order DG results in order to reduce the error and improve the efficiency of the visualization process. Additionally, a one sided post-processing technique is demonstrated in a one-dimensional manner along a streamline in a continuous manner without the loss of accuracy.

Chapter 2

Visualization of High Order DG solutions

2.1 Current Techniques

Solving the problem of accurate visualization of high order methods has become an increasingly urgent issue in the past ten years. As mentioned in Section 1.1, ray tracing is a pixel exact visualization technique. Turner Whitted presented the first algorithm in 1979 and a broad overview of the technique can be found in Andrew Glassner's book [16]. Ray tracing's main problem until this year was its high computational cost. However, Professor Philipp Slusallek from University of Saarland, Germany developed a new algorithm that promises more efficient and faster ray tracing in the near future. While ray tracing produces pixel exact images, there are some methods that aim for the same result in a more efficient fashion.

The most common method to solve the problem of high order visualization is to isotropically subdivide all the elements in the solution mesh according to the order or the element type, whichever is higher. This technique does not require any preprocessing. It inserts the same number of new nodes in every cell in such a way that the reference elements are divided up to p^d equal sized elements. The solution is then sampled at the new node locations in order to enforce accurate visualization at those locations. Since the behaviour of the flow solution is not utilized in any way to control the subdivision of the element, the method is not adaptive. Due to the non-analytic manner of the algorithm, new nodes are inserted and the elements are refined even at locations where the linear interpolation is already adequate. In addition, the user does not have control over the magnitude of the introduced error thus giving a false impressions of the solution.

There are other techniques that try to resolve the issue of high order visualization. Jean-Francois Remacle developed an isotropic visualization-error based adaptation algorithm [28]. In this method, elements are refined isotropically to the highest level. After the subdivision, E_{vis} is checked at every new node location and compared to the tolerance requirement with and without an element. The mesh is then coarsened to the minimum element number at which all the possible nodes still meet the requirement. Even though the total element count is less than for the pure subdivision technique mentioned earlier, isotropic elements are not ideal when the features of the flow are not isotropic (i.e.: vortices, shocks, boundary layers).

Douglas Quattrochi introduced another adaptation technique in his master thesis [27]. The algorithm uses quadrature points as sample points to calculate the visualization-error, and the new nodes are inserted at these locations. Even though the method does not enforce isotropic elements, the triangulation inside a solution cell is done with a Delaunay algorithm [12] which does not allow elements that are too strongly stretched in any direction. In addition, this meshing technique is computationally expensive, thus the cost of the visualization pre-processing step increases exponentially as the element count gets higher.

The goal in this work is to find a technique that approaches the higher order solution accurately based on the user needs. From the user's point of view, the algorithm is ideally both efficient computationally and also enables anisotropic elements to minimize the total number of nodes required.

2.2 New Adaptation Technique for Visualization

The technique introduced here anisotropically adapts one computational element (parent element) at a time by calling the same algorithm on all cells. The refinement is built up from the children elements. This algorithm could be applied on Continuous Galerkin solutions as well, but this might introduce discontinuities between elements in the visualized solution due to the possibility of hanging nodes in the final mesh. The general algorithm is used for visualization in both two and three dimensions, with the differences mentioned explicitly.

Main algorithm

1. Get the quadrature points (sample points C).
2. Calculate the real (high order) solution (u) at quadrature points.
3. While there are sample points with visualization-error (E_{vis}) larger than Error tolerance (E_{tol}).
   (a) Calculate the Linear solution (u_L) for sample points that are inside the area that was affected by the latest re-meshing (Calculate Linear Solution, Section 2.2.1). However, u does not have to be recalculated since it does not change.
   (b) Calculate u_L and the High order solution (u) for new sample points.
   (c) Find the sample point with largest E_{vis} (check points that had $E_{vis} < E_{tol}$ earlier since re-meshing may have changed this relationship).
   (d) If $\max_C E_{vis} > E_{tol}$ then insert a sample point as a new node (InsertNode, Section 2.2.2).
   (e) Insert new sample points on the quadrature points of the new element(s).
   (f) Rearrange area to minimize the total E_{vis} on the sample points (Swapping, Section 2.2.3).

There are multiple termination criteria for the loop. If there are no more sample points with $E_{vis} > E_{tol}$, the loop will end. However, if the user defines an E_{tol} too low, the algorithm may take longer than desired. To avoid this situation, the user can input the maximum number of new nodes introduced per computational element.

2.2.1 Calculate Linear Solution

First, the element which contains the sample point has to be determined.

1. The location of the sample point was known before the last re-meshing.

2. Loop until the element containing the point is found

   (a) Calculate the weights of the sample point in the element. With a matrix multiplication (discussed in Section 1.4, Equation 1.4) the shadow reference coordinates can be calculated (ξ_1', ξ_2', ξ_3')
   2 Dimensions

   $$w_1 = \xi_1', \ w_2 = \xi_2', \ w_3 = 1 - \xi_1' - \xi_2'$$

   3 Dimensions

   $$w_1 = \xi_1', \ w_2 = \xi_2', \ w_3 = \xi_3', \ w_4 = 1 - \xi_1' - \xi_2' - \xi_3'$$

   (b) If all the weights are positive then the element is found.

   (c) If any of the weights are negative then the next element that needs to be checked is the one which is opposite to the face with the most negative weight.

3. Point coordinates are converted to shadow reference coordinates (ξ_1', ξ_2', ξ_3').

4. Node solutions can be linearly interpolated with the use of ξ_1', ξ_2', ξ_3' to get the solution at the sample point (as shown in Equation 1.3).

This function has to be called multiple times because while the higher order solution does not change, u_L changes as the element gets adapted. However, the algorithm uses targeted searching for the element, instead of looping through all of them to find the one that contains the sample point. In addition, the last step, linear interpolation, is a simple vector matrix multiplication where the matrix size only depends on the dimensionality of the problem, not the order.

2.2.2 InsertNode

The sample point is inserted creating new elements in the structure. The number of new elements introduced depends on the where the node is inserted and the dimensionality of the problem. There are three possible locations the algorithm allows for a node to be inserted: in the interior, on an original face of the parent element or on a face which is inside the parent element (interior face). Table 2.1 summarizes dependencies:

Table 2.1: Number of new elements introduced per insertion

	2D	3D
interior	2	3
boundary face	1	2
interior face	2	4

Figure 2-1 shows the three different locations a node can be inserted at in two dimensions. The top right image has child element 2 split up by an interior node into three elements, yielding two new element. On the bottom left, child element 3 is split up by a boundary face node into two elements, one new. Finally, on the bottom right, elements 1 and 2 are split up by an interior face node into 4 elements, 2 new.

Insertion is done automatically. The new node is connected to the nodes of the element in which it is contained. As new elements are introduced, both the total cell count and the node number have to be modified. The connection data between elements also needs to be updated, as does the boundary information if the element is on the boundary.

2.2.3 Swapping

This is the part of the method that can get the most computationally expensive as the total element count becomes high. The number of possible mesh configurations

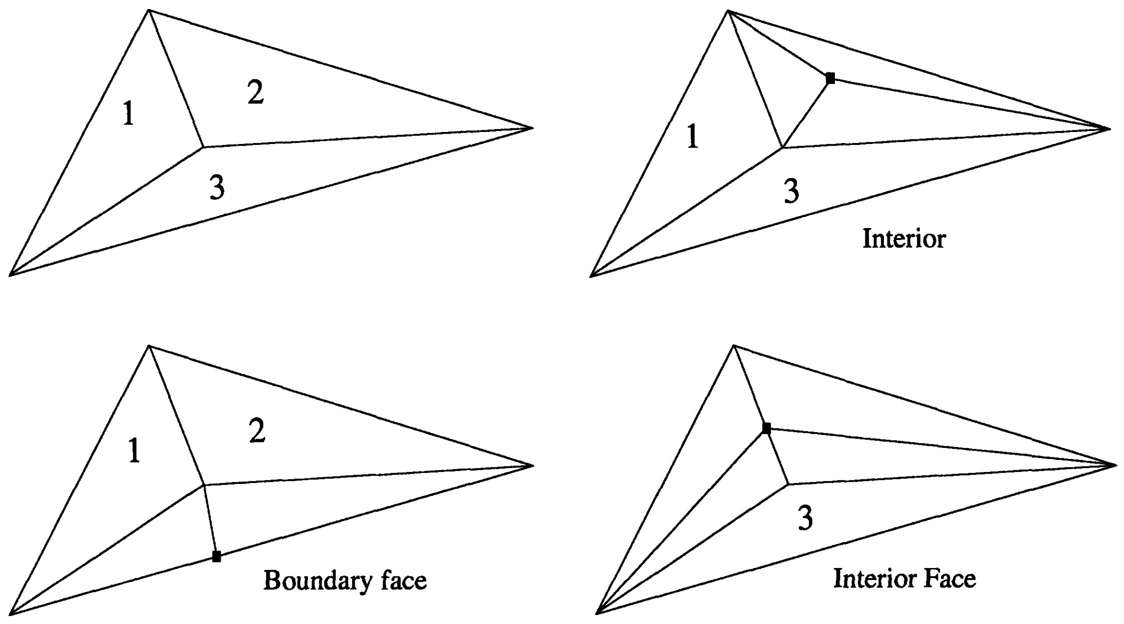


Figure 2-1: Node insertions in a parent that already has one step of refinement done on it (top left image).

increases exponentially with the total number of nodes. In order to guarantee the most efficient algorithm in two dimensions, it is necessary to record the faces that were not swapped, and only check them the next time around if their neighborhood has changed.

1. Initially, when the function begins, all interior edges are marked as swappable.
2. Follow the algorithm until no swapping is performed during the loop.
   (a) Loop through all the interior edges which are marked as swappable.
   (b) Check to see that if the face is swapped it would not create an element with negative area. This is a simple cross product check for 2 dimensional problems. Figure 2-2 on the left shows a swappable face, and on the right one which would create an element with negative area.
   (c) If it is still a candidate for swapping, then check the E_{vis} for all the sample points inside the two neighboring elements of the face for both configurations.

(d) If the second configuration has a lower total E_{vis}, then swap the face and update the locations of the sample points affected.

(e) Mark all other faces on the affected elements as not swappable for this loop but swappable for the next iteration.

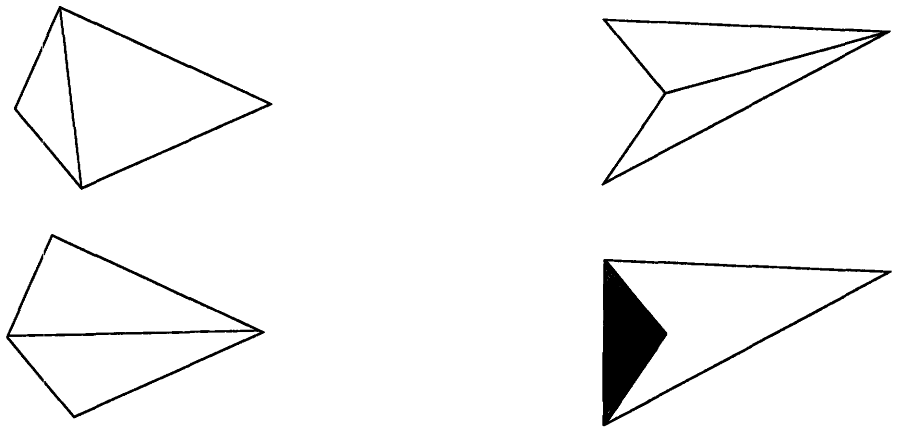


Figure 2-2: Face Swapping in two dimensions. On the left a configuration when the face can be swapped. On the right, swapping would result in negative element area (shaded region).

Three dimensional rearrangement (swapping) is done with TURIN (Tetrahedral Unstructured Remeshing Interface) software developed by Robert Haimes. Instead of swapping edges like in with triangles, in three dimensions a common face of two neighboring tetrahedra can be swapped into an edge and create three tetrahedral.

2.3 Simplex Cut Cell terminology

As described in Section 1.3, cut cells are produced when the body intersects an element. Figure 2-3 shows a two dimensional background element (triangle with 2 edges) that is cut by the boundary. The shaded region of the element represents the section that is inside the flow domain; this valid area is called the cut cell.

There are cases when one background element gets cut into two or more pieces that lie inside the computational domain, these are multiply cut cells, with the cells being each other's sibling element. Figure 2-4 shows the trailing edge of a NACA0012

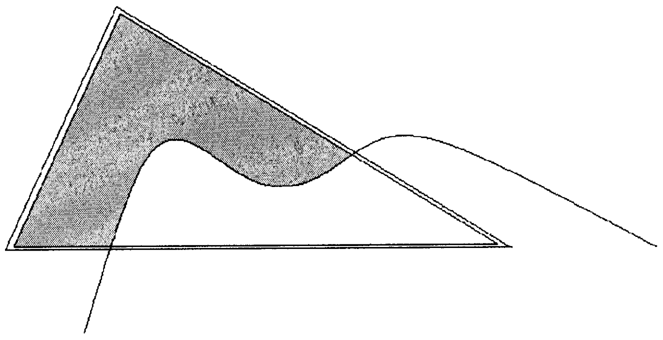


Figure 2-3: Background element cut by the boundary creating the shaded cut cell.

airfoil with a multiply cut cell. It can be seen that cut cells 1 and 2 share the same background element, they are sibling elements.

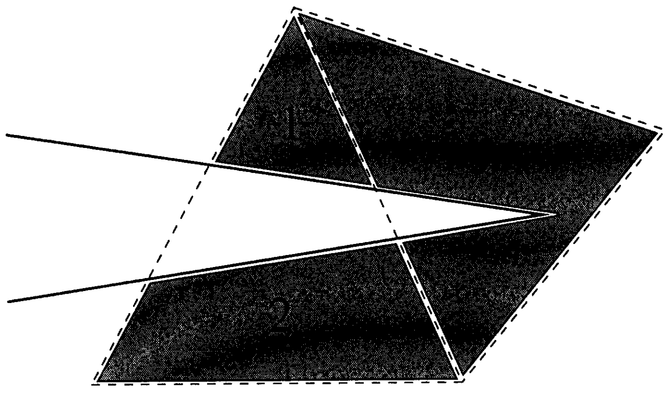
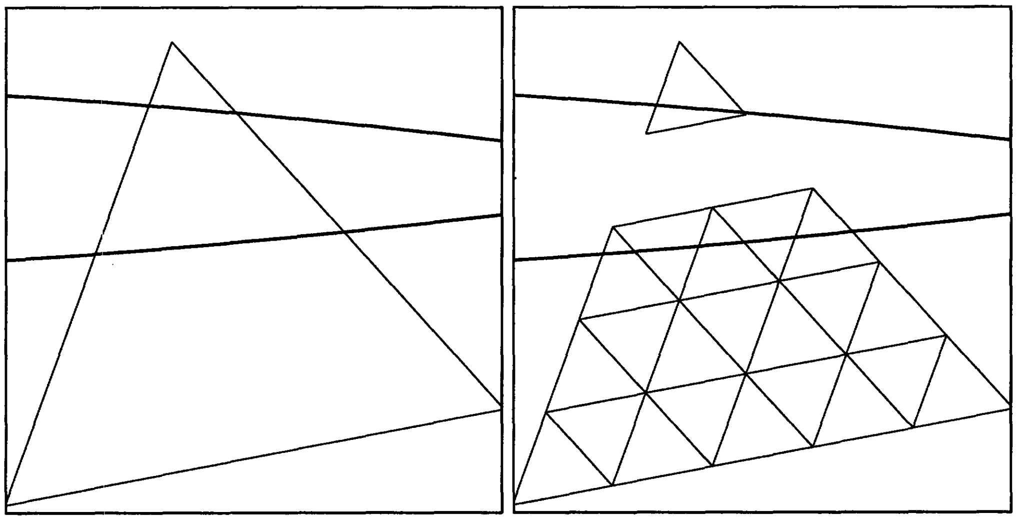


Figure 2-4: Multiply Cut Cell at the trailing edge of an airfoil

2.4 Isotropic Refinement for Simplex Cut Cell Visualization

The isotropic subdivision that can be used on body conforming meshes (described in Section 2.1) can also be applied to Cut Cells with some small additions to the algorithm. All the elements can be adapted in the same manner as before but assigning the solution data to the newly added nodes is different for multiply cut cells (Cut Cells with the same background element).



(a) Non-refined cut cell (b) Refined cut cell

Figure 2-5: Isotropic refinement gets rid of elements completely outside of flow domain

Since the background element is the one which gets isotropically adapted, every new node has to be assigned a solution from one of the cut elements it supports. In order to decide which solution coefficients should be used for the calculation, it must be determined which cut element contains (or is nearest to) the new node. The correct element can be determined by calculating the distance between the quadrature points of all the elements (of the same background cell) to the node and finding the closest.

Second, it is important to add every new element only once. Multiply cut cells share the same background element and thus get cut multiple times. If the same element is present in the mesh multiple times it results in unnecessary memory usage and rendering time. As the case inscreases in size, it gets more important not to waste computing resources.

Last, after all elements are subdivided, there may be some children elements that are completely outside the flow domain. A simple test can be done to check for this situation: if the shadow reference coordinates of all the nodes are outside the bounding box of the parent cut cell then they are outside the domain. In order to save memory and rendering time, these cells are not passed to the visualizer. Figure

2-5 shows an example of isotropic refinement on a cut cell with unnecessary elements removed.

2.5 Adaptation Technique for Simplex Cut Cells

Higher order solutions on Simplex Cut Cell meshes need to undergo the same adaptation as any other solution in order to limit the error introduced from the linear rendering of the visualization process. Some difficulties arise if one tries to use the anisotropic algorithm on Cut Cells.

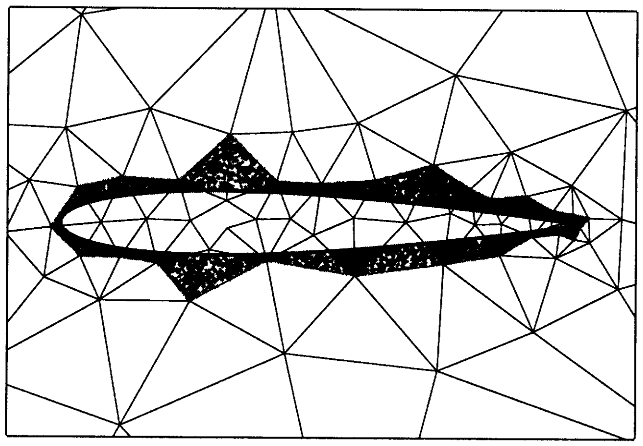
As mentioned in Section 2.2, quadrature points are used as the sample points to calculate the visualization-error and also are used for node insertion. Every Cut Cell has its own quadrature rule because each cell is unique as a result of the cutting. Figure 2-6 shows an example of quadrature points in a two-dimensional NACA 0012 grid. The refinement algorithm examines these quadrature points and uses them as the sample points. By not using the normal quadrature rules we guarantee that the adaptation is done only on the part of the element that is inside the flow region, and no memory and computing power is wasted on points outside the domain.

Another main problem that arises with Cut Cell visualization is the presence of multiply-cut elements. Figure 2-4 shows an example of this situation. The individual elements that belong to the same background element have their own shadow reference coordinates, where the basis functions are defined, and their own set of quadrature points. As a result multiple sets of quadrature rules, solution coefficients and shadow reference mapping matrices have to be caried around with the same background element.

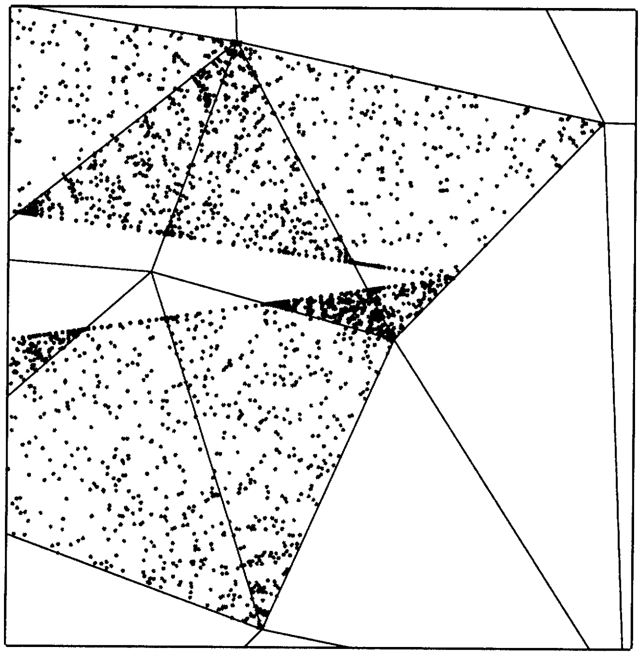
Because of the aforementioned differences between Cut Cell and boundary conforming meshes, there must be changes made to the adaptation algorithm:

2.5.1 Algorithm for cut cell visualization

1. A pre-processing step is used to find all multiply cut cells (Pre-process, Section 2.5.2).



(a) All quadrature points



(b) Trailing Edge quadrature points

Figure 2-6: Quadrature points inside Cut Cells for a 2D NACA 0012 airfoil

2. A sample point structure is created using the quadrature points of all the cells that have the same background element. [13].

3. The same adaptation algorithm is called as for body conforming meshes.

2.5.2 Pre-process

As mentioned earlier, it is essential to determine multiply cut cells from the others and for every element be able to find all of its sibling elements (if it has any).

1. Loop over all Cut Cell elements (A).
   (a) Loop over all the other Cut Cell elements (B).
   (b) Check if A and B has the same background element, if they do than they are multiply cut.
   (c) For every Cut Cell store all of its sibling elements.
2. Mark all the multiply cut elements in order to save time in the adaptation algorithm.

2.5.3 Modifications to Calculate the Linear Solution in multiply cut cells

The sample points come from sections of the same background element each with different mapping to the shadow reference coordinates. Also the sample points have different state solutions at the support nodes. One can consider the situation shown in Figure 2-4 with an element cut into two pieces: in order to be able to get the linear (and also the higher order) solution on any point inside the element, two mapping matrices ($\mathbf{B_1}, \mathbf{B_2}$) and two sets of solution coefficients (a_i) need to be carried around. It is essential to keep track of the cut element to which the sample points belong.

During the adaptation process it is possible to have a visualization element which has nodes in different cells of the solution mesh, multiply cut cell. In order to calculate

u_L for a sample point inside such an element, different sets of solution coefficients have to be used to calculate u for the element nodes.

2.6 Visualization Results

In this section, results will be presented on body conforming and cut cell meshes in both two and three dimensions. All the contour plots show density distributions on both isotropically refined meshes and on ones which were anisotropically adapted. The contours are generated on the visualized solution. Since this solution is linear inside every element, the contour lines themselves are built up from linear sections.

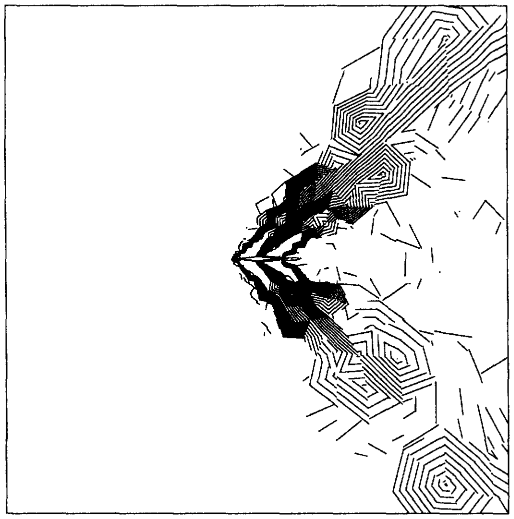
2.6.1 Body Conforming Results

Supersonic Symmetric Diamond Airfoil

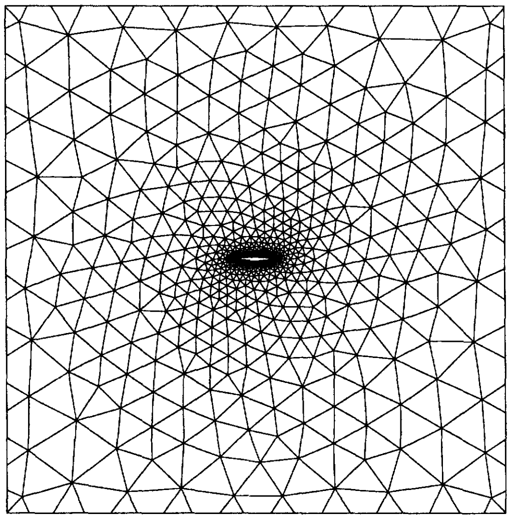
First, the results of a $M = 1.5$ flow about a symmetric diamond airfoil are presented. The solution uses a fifth order polynomial in each element, with a solution mesh built up from 1156 elements. For the error based adaptation, the E_{tol} was set to 10^{-2} and 10^{-3}. Contour lines are shown between $\rho = .7675$ and $\rho = 1.2333$ with 25 increments.

Figure 2-7 shows the contour lines and visualization meshes on the solution grid and on the isotropically subdivided mesh. It is clear that the shocks are not well refined if the visualization is done without any mesh adaptation. The isotropic refinement results in a total of 28,900 elements. It can be seen that upstream from the leading edge the flow is relatively uniform and thus the refinement not necessary. The maximum visualization-error on the isotropically subdivided mesh is $E_{vis} = .00113$ while on the solution mesh it is $E_{vis} = .127$. The error was calculated at the children elements quadrature points.

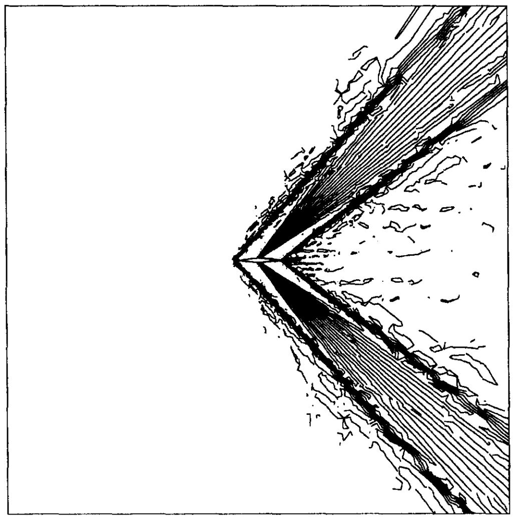
Figure 2-8 shows the same solution but this time the mesh was adapted based on user defined error tolerances. It can be seen that the leading edge shock and expansion fans are well defined even with an $E_{tol} = 10^{-2}$. The anisotropically adapted mesh includes 4,200 elements, while at $E_{tol} = 10^{-3}$ the mesh has 21,157 (still less than the



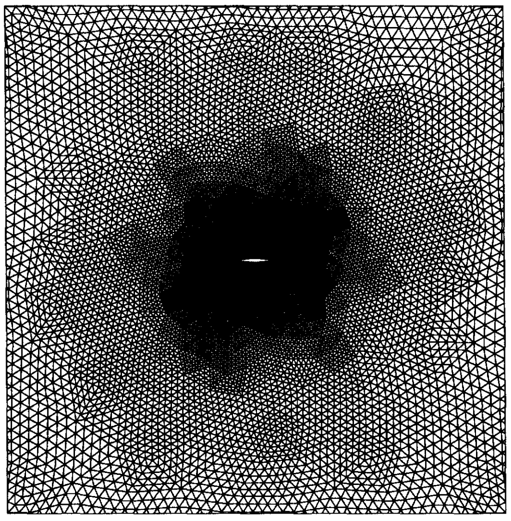
(a) Density contours



(b) Solution mesh



(c) Density contours



(d) Refined mesh

Figure 2-7: Visualization on the solution mesh ($E_{vis} = .127$) and the uniformly refined mesh ($E_{vis} = .00113$). Left side shows density contour lines while on the right the visualization mesh can be seen.

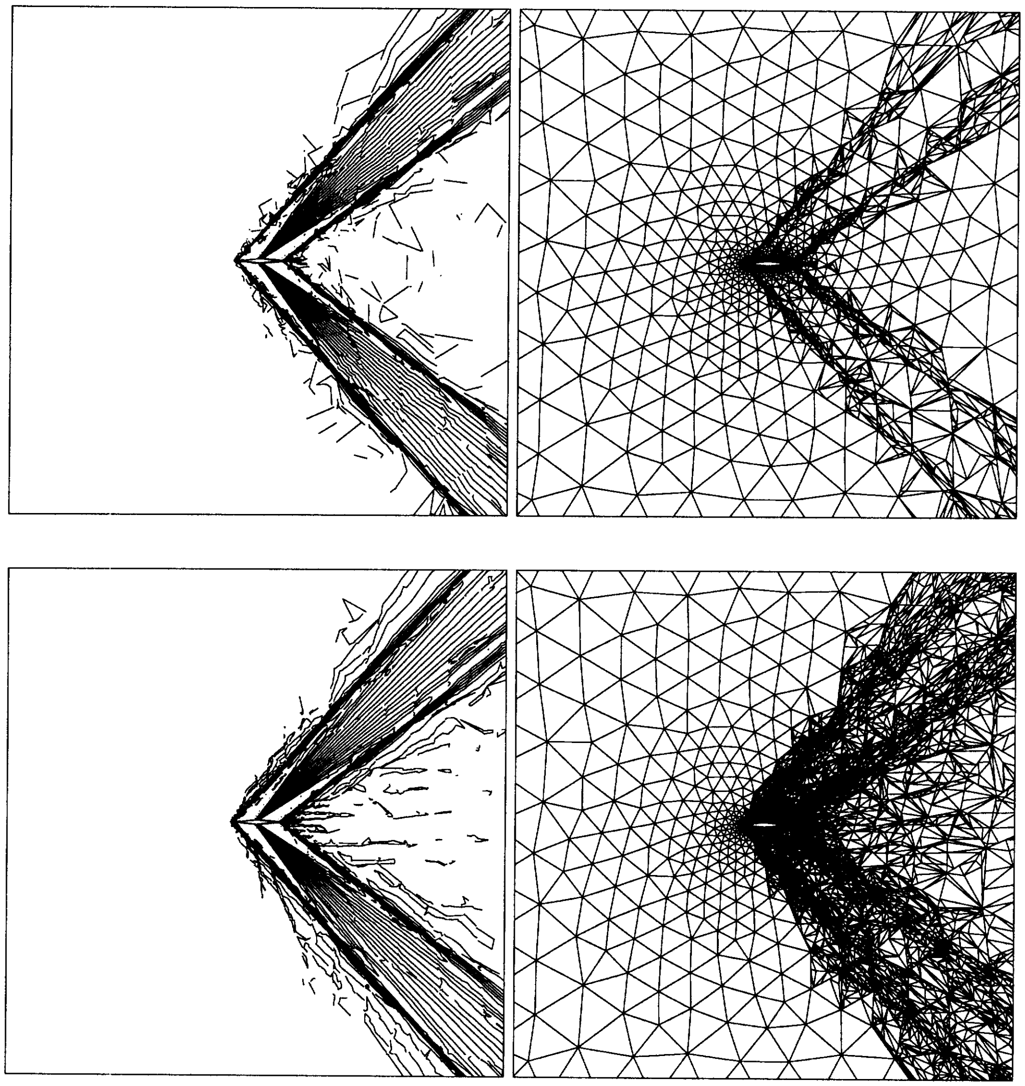
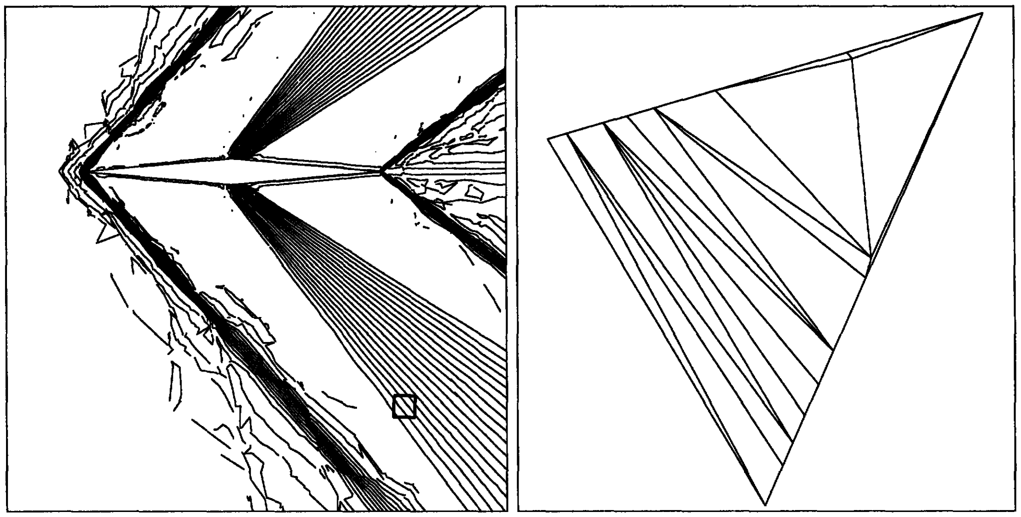


Figure 2-8: Results on the visualization-error based adapted meshes. Top was adapted to $E_{tol} = 10^{-2}$ and the bottom one with $E_{tol} = 10^{-3}$.

isotropically refined needed number). Fewer elements corresponds to less memory used by the visualizer which can be extremely important for larger solutions.

In order to demonstrate the anisotropic manner in which this technique works, a single solution element from the lower expansion fan area was magnified and is shown in Figure 2-9. It can be seen that the new elements created are highly anisotropic and much smaller in the normal direction of the shock.



(a) Contour lines with the magnified section highlighted

(b) Single parent element adapted anisotropically

Figure 2-9: Single element adapted anisotropically along expansion fan

Transonic NACA 0012 airfoil

The following case is a NACA0012 airfoil in transonic conditions M =.8 with an angle of attack $\alpha = 1.128$. The solution again uses a fifth order polynomial in each element and in whis case is built up from 1836 elements. In order for the solver to be able to resolve the solution the domain was set to a relatively large size resulting in the high element count even without adaptation.

In Figure 2-10 it can be seen how small the body is compared to the entire domain. On the solution mesh $E_{vis} = .217$.

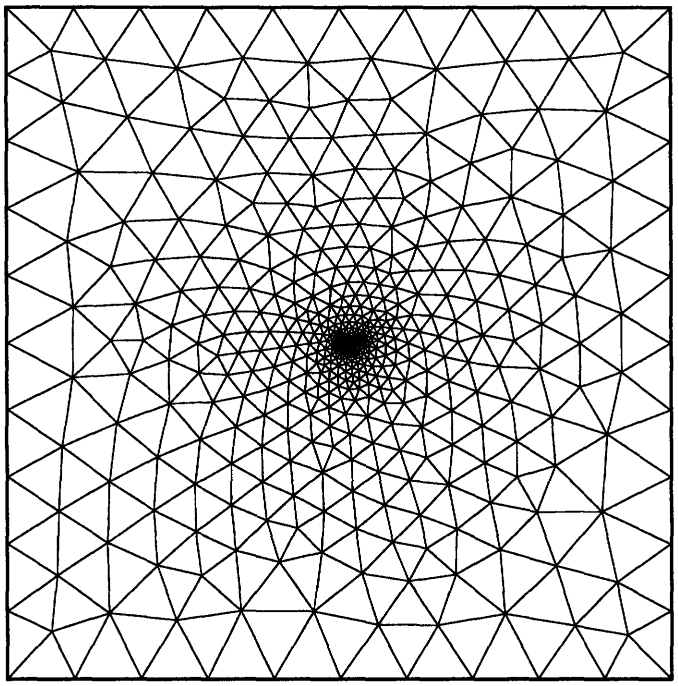
Figure 2-11 shows the density (ρ) solution on the isotropically refined mesh, $E_{vis} = .00278$, with 45,900 element count, and on a mesh which was adapted with an $E_{tol} = 10^{-3}$ and contains 6,672 elements. It can be seen that the new adaptation technique only refines the mesh where it is necessary (mainly the weak shocks on the top and the bottom of the airfoil, and the leading and trailing edges). The shocks are well resolved in both cases. In some locations in the anisotropically refined mesh the shocks are better resolved, even though the element count is almost an order of a magnitude less.

The two cases shown here demonstrate how the new technique reduces the total element number, and thus the total memory requested to achieve a required error tolerance, compared to a simple isotropic refinement. Low element number does not just cut down the memory required for the visualization but also reduces the rendering time of the visualizer. The use of the method becomes more important as the order of the solution polynomial increases. Higher polynomial produces larger visualization errors which cannot always be resolved efficiently by isotropic elements.

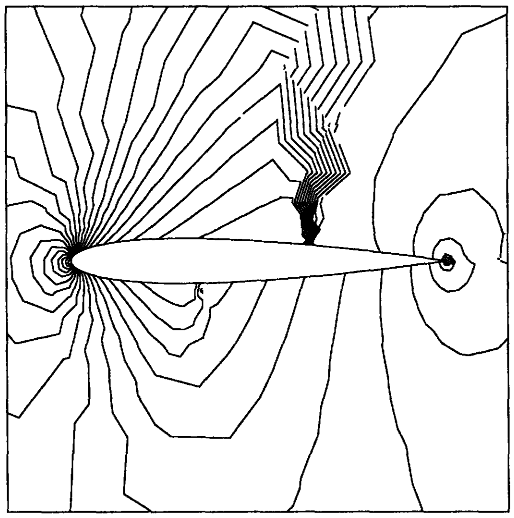
On the other hand, the adaptation has a pre-processing phase where the new required node locations are calculated. This phase takes time that depends on the E_{tol} the user wishes to reach.

Three dimensional wing

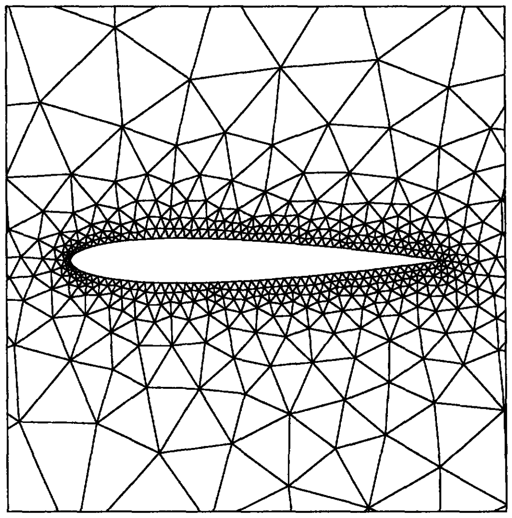
A three dimensional onera wing is presented in M = 1.5 conditions. Since the solution is only a second order polynomial, the difference between visualization on a non-adapted versus an adapted mesh is not as significant as for the earlier examples. The element size of the solution mesh is 45,937. This number clearly shows how the dimensionality of the problem affects the element count and thus the computing memory required for the visualization process. The isotropically refined mesh includes 363,336 elements and has an $E_{vis} = .001$. The same visualization error can be reached by anisotropic adaptation with a 318,937 element count.



(a) Solution grid on the entire domain

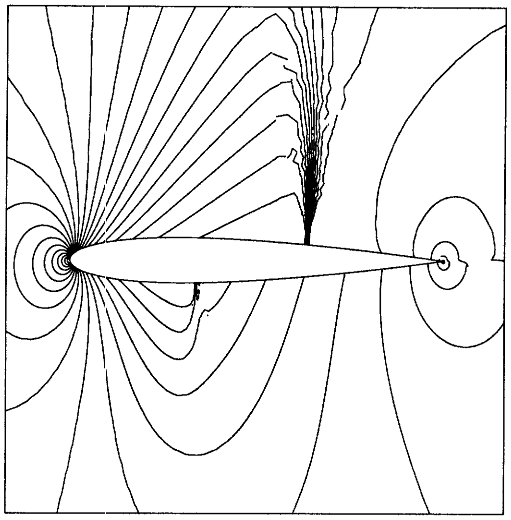


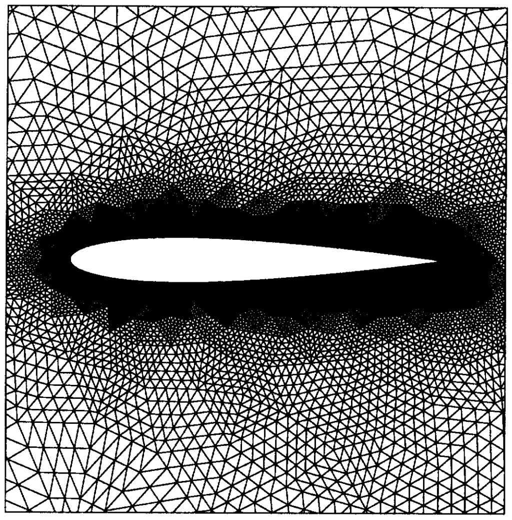
(b) Density contours on the solution mesh

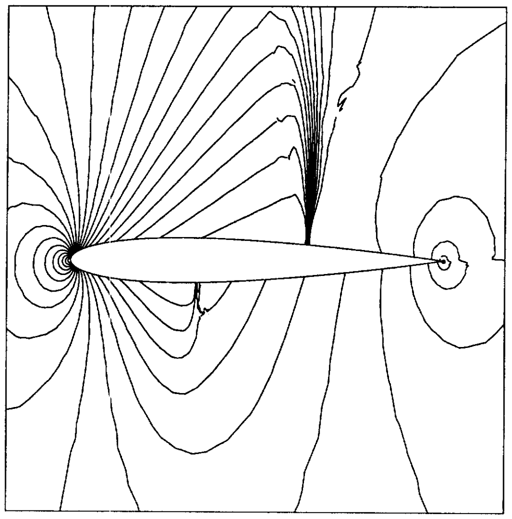


(c) Solution mesh zoomed in

Figure 2-10: Results on the solution mesh ($E_{vis} = .217$).


(a) Density contours


(b) Refined mesh


(c) Density contours

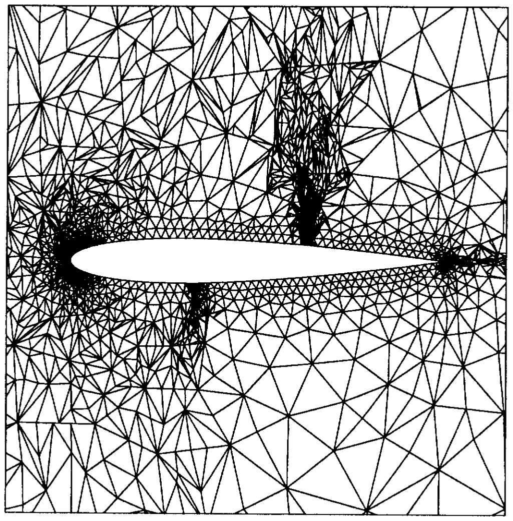

(d) Adapted mesh

Figure 2-11: Results on the isotropically refined ($E_{vis} = .00278$) and visualization-error based adapted meshes. The bottom two images were adapted with a $E_{tol} = .001$.

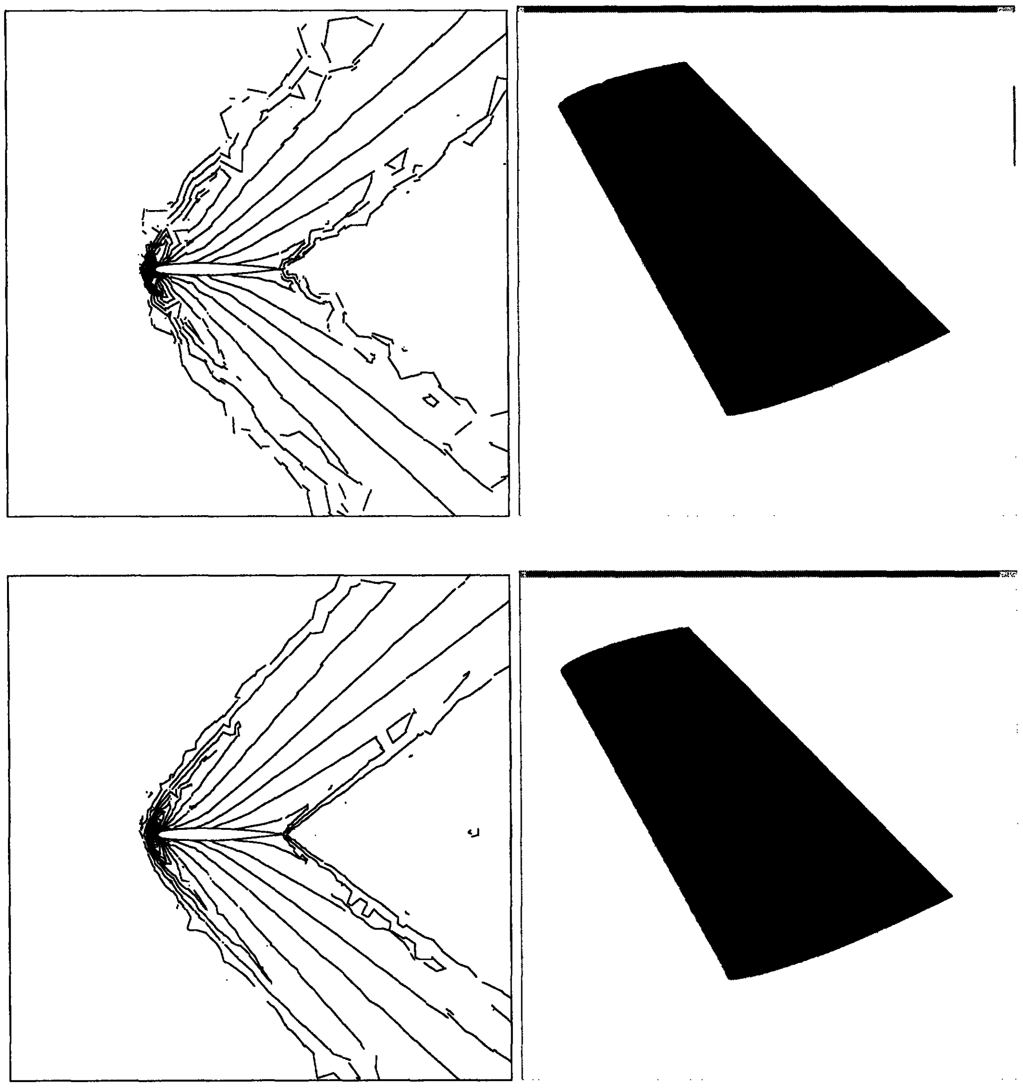


Figure 2-12: Results on the solution and the E_{tol} = .001 meshes. Left side shows density contour lines on a 2D cut plane while on the right the contour lines on the body are visible.

2.6.2 Cut Cell Results

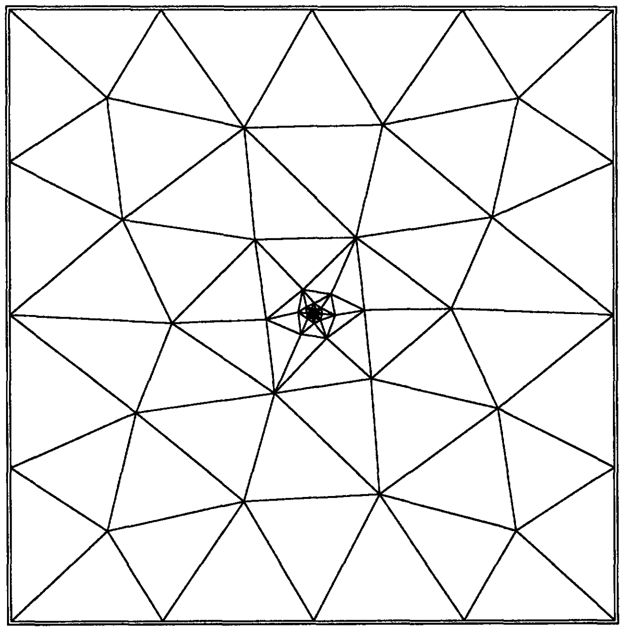
NACA inviscid airfoil with multiply cut elements

This example shows a NACA 0012 airfoil with an angle of attack $\alpha = 1.128$ and $M = .5$ on a non-adapted cut cell mesh. The mesh includes multiply cut elements, and even though the solution is converged, it still needs refinement to resolve the singularity at the trailing edge. The example is primarily shown in order to demonstrate the new techniques ability to handle multiply cut elements. The solution is a second order polynomial on 290 elements. The low element count is the result of using a coarse cut cell mesh.

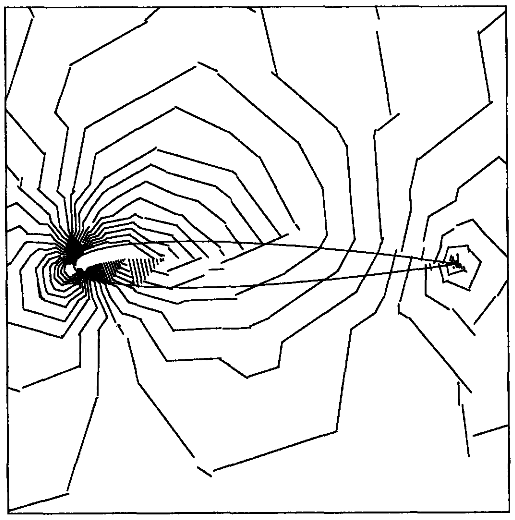
Figure 2-13 shows the solution mesh and the density distribution contour between $\rho \in [.9, 1.08]$. Since the solution is a relatively low order polynomial the maximum visualization-error is only $E_{vis} = .027$. There are density contours inside the airfoil since the visualization is done on the background mesh. The body is overlayed on the solution so the user can tell what part of the solution is inside the flow domain.

Figure 2-14 shows density contours and visualization grids for both the isotropically subdivided and adapted meshs. The refined grid is built up from 1819 elements and has a $E_{vis} = .0012$. Since this method creates some elements that are completely inside the body, those were taken out from the mesh to reduce the rendering time, however this increases the pre-processing step. The adapted results were created with a $E_{vis} = .0012$ and include 1763 elements. Since there are no new nodes inside the body, unlike in the isotrpoically refined case, there are no elements that can be taken out the mesh, causing the only minor difference in element counts. However, the nodes are inserted in an adaptive manner resulting in much smoother contour lines inside the cut cells, close to the body.

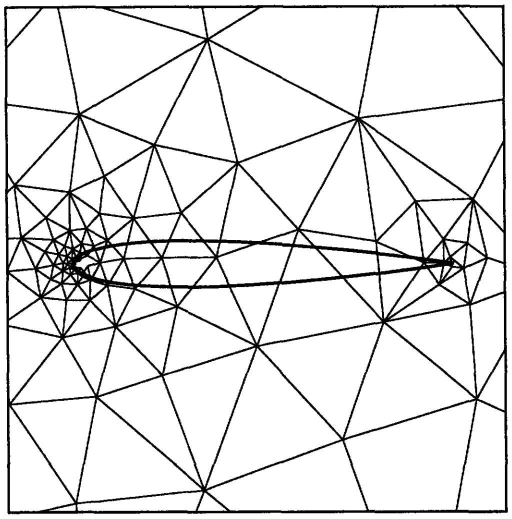
Lastly, Figure 2-15 shows a multiply cut and a simply cut element that were adapted by the new technique. There are no new nodes inserted inside the airfoil, only on the surface and inside the flow domain. The children elements are not multiply cut, so an element has all of its node solutions from the same cut cell.



(a) Solution grid on the entire domain

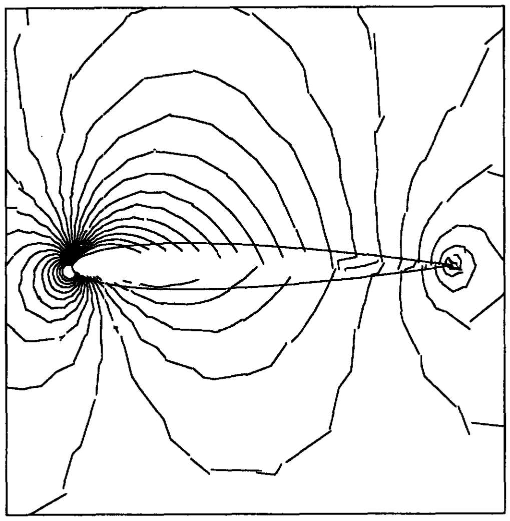


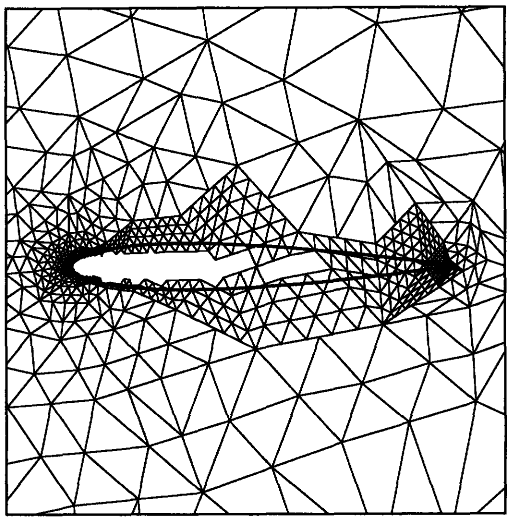
(b) Density contours on the zoomed in solution mesh

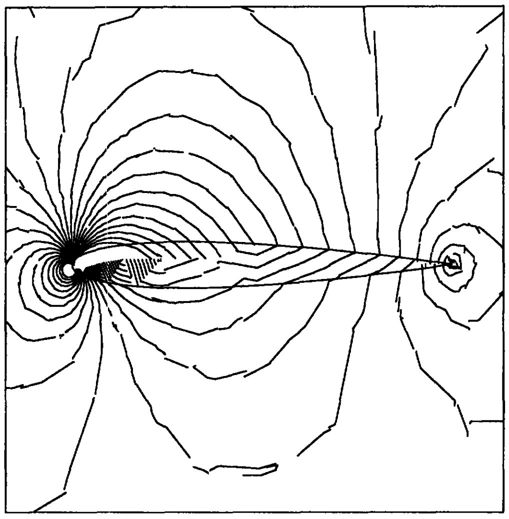


(c) Zoomed in solution mesh

Figure 2-13: Results on the solution mesh E_{vis} = .027. Solution exists inside the airfoil but may be disregarded.


(a) Density contours


(b) Refined mesh


(c) density contours

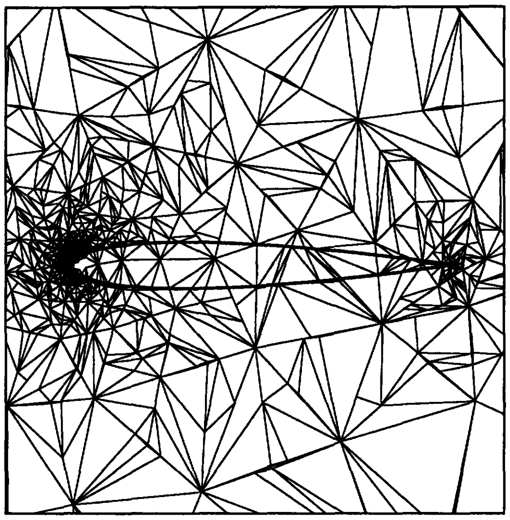
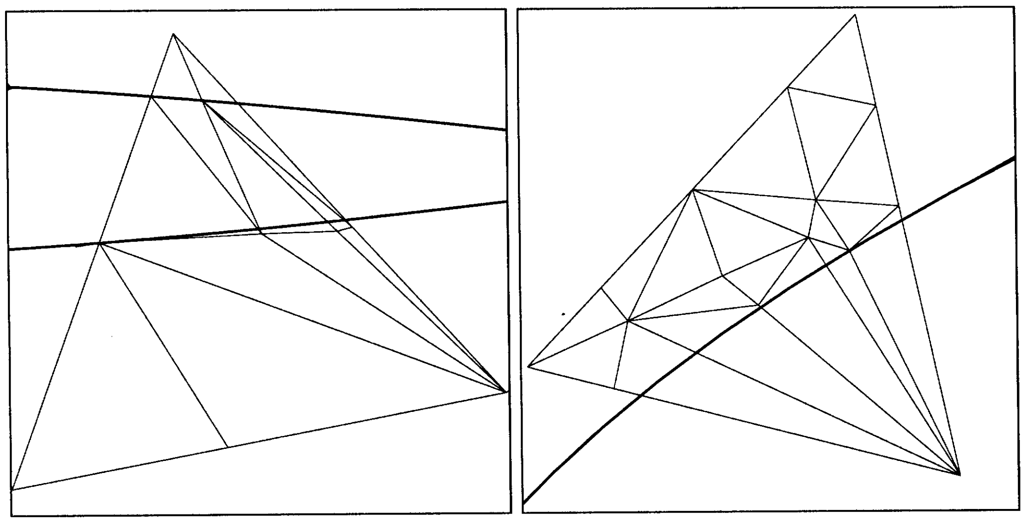

(d) Adapted mesh

Figure 2-14: Results on the isotropically refined $E_{tol} = .0012$ and visualization-error based adapted meshes. The bottom two images were adapted with a $E_{tol} = .0012$.



(a) Multiply cut element

(b) Cut element

Figure 2-15: Adapted cut cells

Chapter 3

Streamlining in DG fields

3.1 Current Techniques

Visualizing vector fields has always been a challenge. There are several calculation techniques that try to adress this challenge: particle tracing [19, 31], icon [24, 11] or texture [6, 10] based methods, etc. One of the most commonly used techniques is streamline visualization.

Streamlines can be useful in understanding flow solutions and thus they can be helpful in the engineering process. However, little effort has been spent on finding efficient and accurate techniques to calculate and then visualize the results. The current method to trace streamlines is to use one of the many available numerical ordinary differential equation (ODE) solvers on the vector field. All of these time-stepping algorithms assume continuity of the underlying field and their order of accuracy decreases as the field becomes discontinuous. Curtis et. al [8] demonstrated how the Runge-Kutta 4 method, that has fourth-order convergence on continuous smooth fields, demonstrates only first-order convergence when applied on a discontinuous field. As the convergence rates worsen, these methods require more steps (smaller timesteps), and thus more computing power and time. In addition to this increase in processing time, discontinuities in the solution between elements may cause problems (described in Section 1.1) that can result in physically incorrect streamlines or possibly the failure of the time integrator itself.

Darmofal and Haimes [9] analyzed different types of time integrators (multistep, multistage and hybrid schemes) for their efficiency, convergence rate and computational cost in stream-, streak- and path-line calculations. They also showed that an improperly chosen timestep can result in non physical streamlines and incorrect critical points even in continuous fields.

The generic form of a multistep scheme is the following:

$$\sum_{t=0}^{s} \alpha_i x^{n+1-t} = k \sum_{t=0}^{s} \beta_t u(x^{n+1-t}, t^{n+1-t})$$

Where s represents the number of steps, for explicit schemes $\beta_0 = 0$. The main disadvantage of higher order accurate multistep methods ($s > 1$) is that they require $s - 1$ startup steps that are lower order accurate thus additional error gets introduced to the time-integration. Computationally they are not as expensive as multistage methods since only one velocity field needs to be evaluated per iteration; the previous positions can be stored and reused.

Even though some families of multistage methods can be written down in a generic form (for example Strong Stability Preserving Runge-Kutta methods), there is no general descriptive formula. The most commonly used multistage techniques are from the Runge-Kutta family. One of the main concerns with these schemes is that they require the calculation of the velocity at intermediate steps which can be computationally expensive and result in longer calculation time. In order to be able to calculate the velocity, the underlying element in the grid that contains the intermediate stage has to be found. This process is the most computationally expensive part of the streamline calculator. In addition, for cases when the solution is only known at sample points, intermediate velocity calculations can cause difficulty that is usually solved by the use of a Newton-Rhapson scheme to get an interpolated value. Introducing this extra numerical algorithm for interpolation produces additional error in the streamlining process. If the time integrator has access to the full polynomial solution, not just the visualized one, the interpolation phase can be avoided and the order of accuracy is not compromised.

There are methods that use adaptive timesteps (e.g. predictor-corrector, RK-4/5) in order to bound the local (per step) error. In a DG setting for the time-integrator the jump in the solution at element boundaries appear to be an error. The local error depends on the size of the jump and the Δt. In order to have minimal introduced error at big jumps, the timestep needs to be very small. As Δt becomes smaller, more iteration is needed and getting passed a single element boundary becomes computationally expensive. As a result, if one wishes to achieve a relatively accurate streamline with these methods, the process can become rather time consuming due to all the discontinuities the streamline encounters in its path [15].

There have been many filtering techniques developed that aim to smoothen out discontinuous fields and create a continuous function from a discrete solution. A group of these methods uses B-splines for the filtering. An overview of B-spline filtering techniques can be found in [22]. Hou and Andrews [17] use linear combination of cubic B-splines for image processing. Their technique is very similar to the one used in this work. Mitchell and Netraveli [20] also use piecewise cubic filters for image reconstruction, not limiting to B-splines.

The goal of this work is to demonstrate a post-processing algorithm that can be applied during the time-integration in order to improve the accuracy of streamlining on unstructured meshes in discontinuous fields. The technique uses a one dimensional convolution kernel to introduce continuity between elements, increase smoothness while not introducing additional error.

3.2 The uniform symmetric post-processor

The post-processor takes advantage of the information the negative norm gives about the oscillatory behaviour of the error introduced in Galerkin methods [4]. It improves the accuracy in the L^2 norm by filtering the oscillations out of the solution. The framework of the post-processor was introduced by Bramble and Schatz [2] and Mock and Lax [21]. Cockburn et al. showed that the method, if applied on DG approximations of linear hyperbolic systems on uniform meshes, can improve the accuracy from order $k+1$ to $2k+1$.

The post-processor is a convolution kernel that satisfies the following three requirements. 1) it is a linear combination of B-splines, 2) it has compact support, 3) it can reproduce polynomials of degree $2k$ by convolution.The post-processed solution can be calculated by the following way:

$$u^\star(x) = K^{2(k+1),k+1} \star u_h(x) \tag{3.1}$$

$$u^\star(x) = \frac{1}{h}\int_{-\infty}^{\infty} K^{2(k+1),k+1}\left(\frac{y-x}{h}\right) u_h(y)dy \tag{3.2}$$

Where h represents the element size, u_h is the DG solution, $u^\star$ is the post-processed solution and $K^{2(k+1),k+1}$ is the convolution kernel. The B-splines for the kernel are convolutions of the characteristic function χ with itself k times.

$$\chi(x) = \begin{bmatrix} 1, & x \in \left(-\frac{1}{2},\frac{1}{2}\right) \\ 0, & otherwise \end{bmatrix}$$

The convolution kernel looks like the following:

$$K^{2(k+1),k+1}(x) = \sum_{\gamma=-k}^{k} c_\gamma^{2(k+1),k+1}\psi^{(k+1)}(x-\gamma) \tag{3.3}$$

In the above equation $\psi^{(k+1)}$ represents the $(k+1)$th convolution of the characteristic function with itself and $c^{2(k+1),k+1}$ are constants. The property of the kernel that it reproduces polynomials of degree $2k$ by convolution is used to calculate these constants.

$$\begin{bmatrix} \int \psi^{k+1}(x-y-k)y^0dy & \dots & \int \psi^{k+1}(x-y+k)y^0dy \\ \vdots & \ddots & \vdots \\ \int \psi^{k+1}(x-y-k)y^{2k}dy & \dots & \int \psi^{k+1}(x-y+k)y^{2k}dy \end{bmatrix} \begin{bmatrix} c_{-k} \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} x^0 \\ \vdots \\ x^{2k} \end{bmatrix} \tag{3.4}$$

The constants only need to be calculated once, since they only depend on the polynomial order of the solution. The post-processor can be expanded to two and three dimensions. The higher dimensional kernel is simply a product of the one dimensional formulation, in 2D:

$$K^{2(k+1),k+1}(x,y) = \sum_{\gamma_x=-k}^{k} c_{\gamma_x}^{2(k+1),k+1}\psi^{(k+1)}(x-\gamma_x) \sum_{\gamma_y=-k}^{k} c_{\gamma_y}^{2(k+1),k+1}\psi^{(k+1)}(y-\gamma_y)$$

3.3 Post-processing for accurate streamlining

Curtis et. al [8] demonstrated a technique to apply the post-processor for streamline calculation. In their work they used the convolution kernel on the final numerical solution before the use of a time integrator. This method is very effective and computationally manageable if the underlying mesh is uniform structured and there are no objects inside the domain (i.e. boundaries are periodic).

In order to use the technique on meshes with bodies inside the domain, some changes have to be made. The main problem with using the symmetric kernel for problems like this is as one tries to post-process the solution close to the body, the support of the kernel includes places outside the flow domain. The solution polynomial either does not exist inside the body, or if it does (cut cell meshes) then it is invalid. Avoiding this problem requires the use of a fully one-sided post-processor. However, depending on the shape of the body boundary, it may become a complex problem to find the support that is entirely inside the flow domain. While in two dimensions the problem could be solvable, in three dimensions with complex geometry this may be intractable. The problem in three dimensions becomes a moving 'structured block' grid generation problem. Even if the problem becomes solvable it is computationally prohibitive. Since one of the main goals is to have a fast algorithm, something that becomes more important for pathline calculations and unsteady problems, a new method is necessary for these problems.

In order to reduce the computational cost, the method introduced here does not post-process the entire flow domain. However, as every time integrator needs the solution at sample locations to iterate, these points do need to be post-processed. In addition, a one dimensional convolution (post-processing) is performed along the streamline, thus eliminating the problem that may arise close to complex body boundaries. As only the past of a streamline is known at each step of a time integration, a fully one-sided kernel is used instead of the symmetric kernel. (This one-sided kernel is described in Section 3.4)

Since the post-processing is done along a streamline in arch-length coordinates, the corresponding mesh is not uniform. Even if the underlying grid is uniform a streamline may slice through it in such a way that it would result in greatly varying element sizes. Two methods have been introduced by Curtis, Kirby et al. [7] for non-uniform meshes. First, a local L^2 projection of the mesh to one that is uniform. Second, the idea of characteristic length for the kernel can be introduced. As for non-smoothly varying meshes the second method is more efficient, therefore it has been chosen for this work. The choice of characteristic length is an important part of this technique and is described in Section 3.6.

3.4 Fully one-sided post-processor

The one-sided convolution kernel that was introduced by Ryan and Shu [30], displays only minor differences from the symmetric:

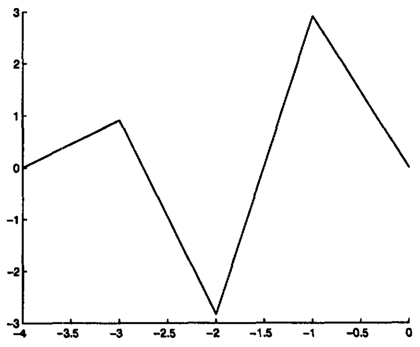
$$K^{2(k+1),k+1}(x) = \sum_{\gamma=-2k-1}^{-k} c_\gamma^{2(k+1),k+1} \psi^{(k+1)}(x-\gamma)$$

The coefficient calculation which are also non symmetric given by:

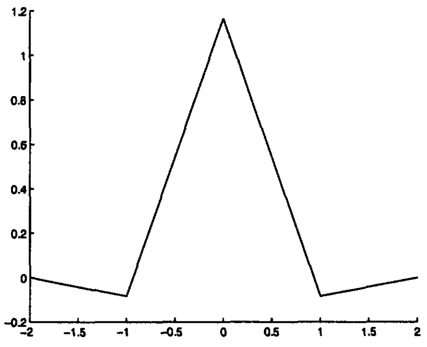
$$\begin{bmatrix} \int \psi^{k+1}(x-y-2k-1)y^0 dy & \dots & \int \psi^{k+1}(x-y-k)y^0 dy \\ \vdots & \ddots & \vdots \\ \int \psi^{k+1}(x-y-2k-1)y^k dy & \dots & \int \psi^{k+1}(x-y-k)y^k dy \end{bmatrix} \begin{bmatrix} c_{-2k-1} \\ \vdots \\ c_{-k} \end{bmatrix} = \begin{bmatrix} x^0 \\ \vdots \\ x^k \end{bmatrix}$$

The entire kernel is on one side of the point of interest therefore the limits of the convolution change. Since the integral is one sided, the solution is not needed from the other side of the point, which is important since the future of the streamline is not know when this post-processing scheme is employed.

It is shown that the one-sided technique also demonstrates an improvement in accuracy from order $k+1$ to $2k+1$, but only for uniform meshes. However, when the underlying mesh becomes non-uniform, the method loses its accuracy enhancing characteristic, but keeps an accuracy conserving behaviour [30].



(a) Fully one-sided kernel



(b) Symmetric kernel

Figure 3-1: Convolution kernels for the $k = 1$ case.

Figure 3-1 shows an example of the symmetric and the fully one-sided kernel for $k = 1$. The x-axis is scaled by the characteristic length, and 0 corresponds to the location of the point that is being post-processed. The y-axis is scaled by the kernel coefficients.

3.5 Numerical integration for the post-processor

The convolution step of the post-processor require us to evaluate an integral where the integrand is the kernel multiplied by the solution (Equation 3.2). For this integral the Jacobi-Gauss quadrature rule is used to get the exact answer. As the integrand is discontinuous, each continuous subregion is numerically integrated according to the appropriate quadrature rule.

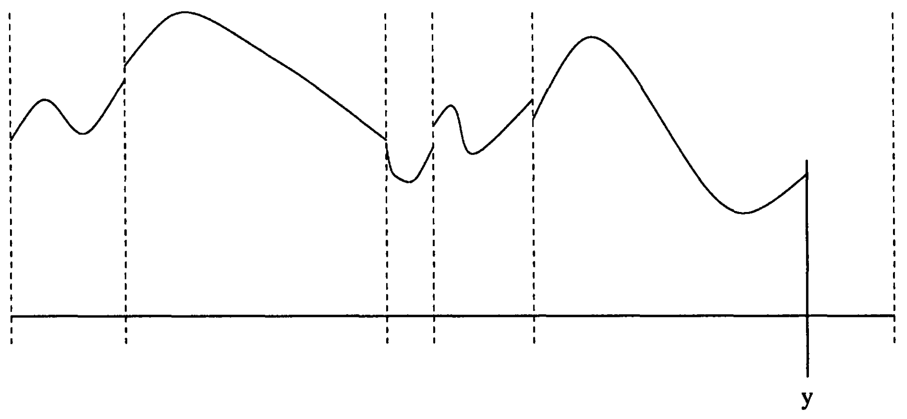


Figure 3-2: Discontinuous integrand as seen by a streamline.

For example, let us investigate the case when the post-processed solution needs to be calculated at point y in Figure 3-2. The integrand is a fifth order polynomial within each continuous region, but there are discontinuities at the boundaries. For the numerical integration to give the exact solution it requires three quadrature points per segment. The integrand is then evaluated at these locations, multiplied by the appropriate weights, and summed up.

Also, the arch-length coordinates of the quadrature points have to be converted to physical coordinates in order to get the solution data. Since the streamline is calculated with a time integrator, it is built up and visualized from linear line segments, while in reality it is a higher order curve. However, the quadrature points of the integral does not necessarily coincide with the locations that the time integrator calculates. To get the real coordinates of a quadrature point that is not one of the actual sample points, an interpolation is done between the two end points of the linear segment. Once the coordinates are found, the real high-order interpolant is used as the solution at the point. Figure 3-3 shows how there can possibly be a difference between the real coordinates of a quadrature point if it is calculated on linear segments or higher-order curves. The squares represent the locations the time-integrator calculated, x is a quadrature point along the streamline.

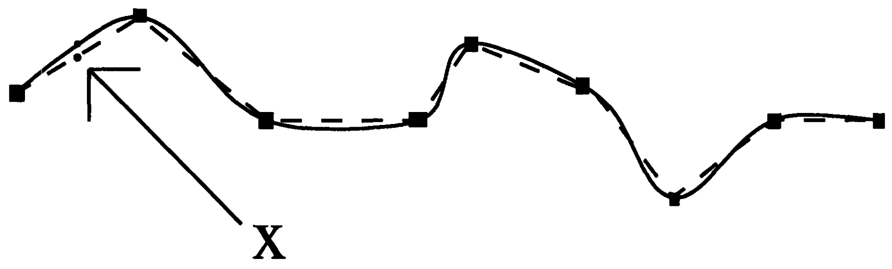


Figure 3-3: Continuous line is the real higher-order streamline, dashed line is built from linear segments

3.6 Ideal characteristic length

The characteristic length is the scaling factor of the convolution kernel, for uniform meshes it is the element size. In order to find the ideal characteristic length one would need using this post-processing method, a test case was set up. A two dimensional vector field was created, one that has streamlines which are oscillating circles. Figure 3-4 shows a sample streamline that was calculated on the original continuous field with the Runge-Kutta 4 (RK4) method and a timestep $\Delta t = .001$. It can be seen that there are a total of 20 cycles of oscillation around one loop.

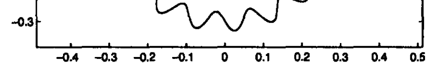
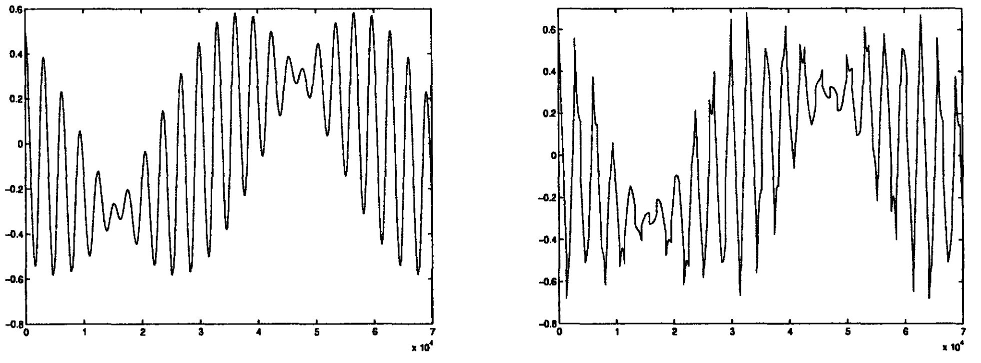


Figure 3-4: A sample streamline over the continuous field with a starting location of (0,.3)

The field was then projected on a uniform mesh with L^2 projection as a first-order polynomial to make it discontinuous, as if it was generated by a DG solver. Figure 3-5 shows the u values (x velocity) along the streamline, in arch-length coordinates (x axis). It can be seen that the projected solution is neither continuous nor smooth.



(a) Actual values (b) L^2 projected values

Figure 3-5: X velocity along the streamline (x-axis is in 10^{-4} arch-length coordinates).

As a next step, the one sided post-processor was applied on this solution along the streamline. The real streamline was sampled with a $\Delta t = 10^{-7}$. The integration was done with Jacobi-Gauss quadrature rules, along the real oversampled streamline, this way the real coordinates of every quadrature point were known exactly. The post processed solution was calculated with characteristic length ranging from .01 to .3 in arch-length coordinate. The maximum error between the real and the post-processed solution is shown of Figure 3-6. The vertical line is the maximum error between the projected and the real solution.

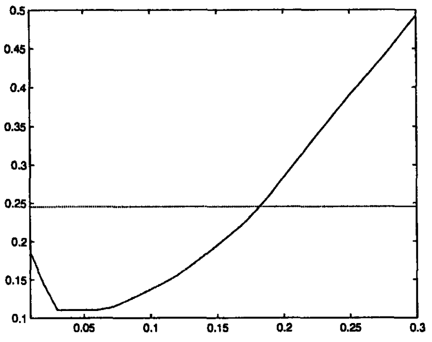
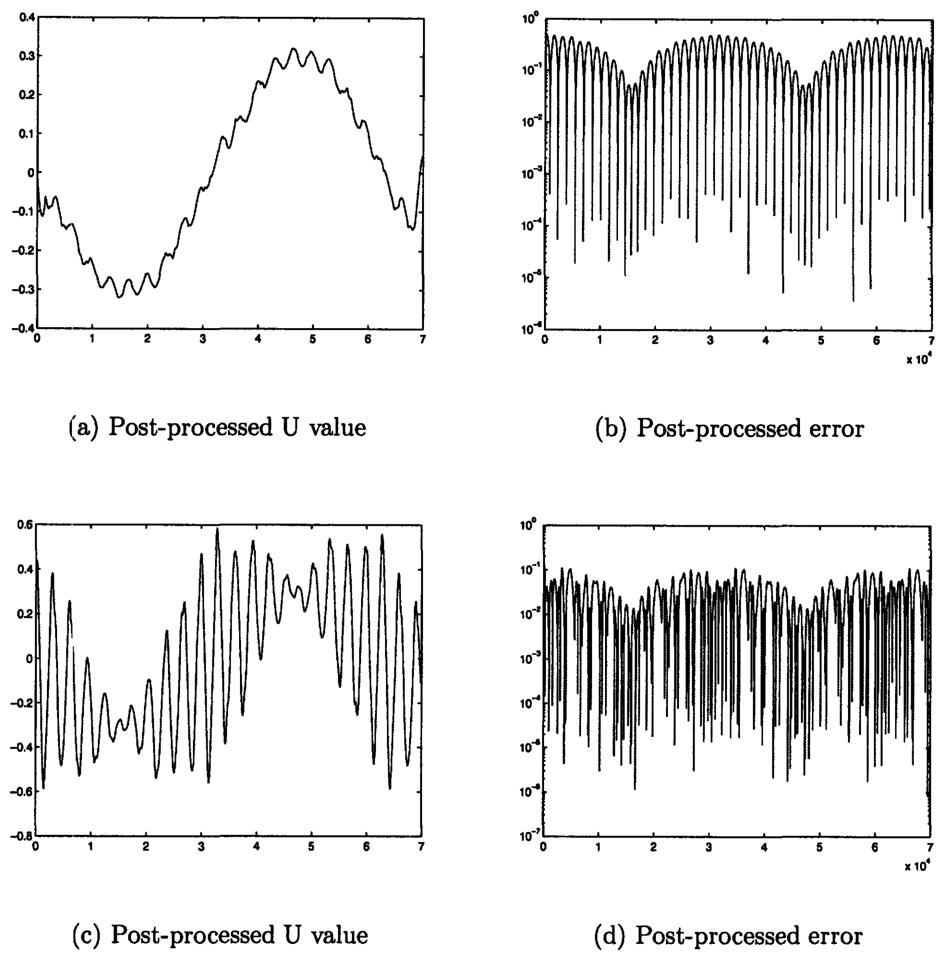


Figure 3-6: Maximum error (in x velocity) vs. characteristic length (in arch-length).


(a) Post-processed U value

(b) Post-processed error

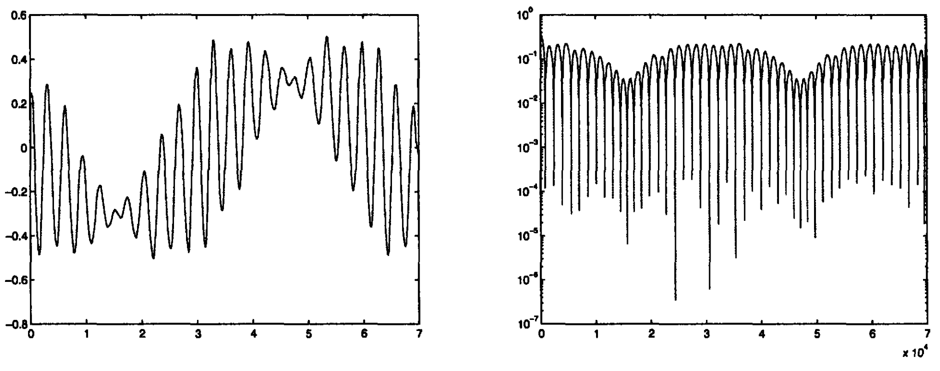
(c) Post-processed U value

(d) Post-processed error

Figure 3-7: Results with char length of .3 on the top and .05 on the bottom.

The maximum error improves during the post-processing for characteristic lengths smaller than .18. The reason for this is that if the kernel support is too big then high frequencies of the solution get filtered out. At the same time if the characteristic length is too low, the support is not large enough to reduce the unwanted oscillations. Figure 3-7 shows the two extreme cases, one with characteristic length of .3, the other with .05.

The same test was run on similar vector fields just with lower number of oscillation cycles around the circle. It was observed that when the characteristic length was set to the largest element size, the post-processed maximum error was always lower than



(a) Post-processed U value (b) Post-processed error

Figure 3-8: Results with char length of .14

the non post-processed. Table 3.1 sums up the results by showing the maximum error for the non post-processed (projected) and the post-processed solution, with the maximum element size as the characteristic length.

Figure 3-8 shows the post-processed u values and the error whan the maximum element size was used as the characteristic length for the convolution kernel. It can be seen that the solution is continuous and significantly smoother than the original projected solution (Figure 3-5 b).

Table 3.1: Post-processed error when the characteristic length is set to the maximum element size.

Number of cycles	maximum element size	maximum post-processed error	maximum projected error
2	.154	.068	.104
4	.244	.018	.019
6	.179	.014	.031
8	.108	.021	.052
10	.151	.048	.063
12	.126	.051	.105
14	.124	.078	.151
16	.158	.159	.191
18	.138	.136	.172
20	.101	.137	.249

3.7 Choice of time-integration technique

There are a wide variety of time-integration techniques available [25, 3]. For the purpose of this work there were some restriction on what type can be used. In order to be able to use implicit schemes, the solution needs to have smooth first derivatives. Since the post-processor guarantees only smooth solutions, it is necessary to use an explicit method. Also, one would like to use an integrator that demonstrates a high order of convergence, to reduce the number of timesteps needed for the calculation. The most commonly used method is RK4 or RK45. The main problem that arises with these two techniques is that they sample the field from locations that are not along the streamline. RK4 formula is shown below. Step 3 and 4 does not satisfy the requirement of being along the streamline.

$$\begin{aligned}
k_1 &= hf(t_n, y_n) \\
k_2 &= hf(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_1) \\
k_3 &= hf(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_2) \\
k_4 &= hf(t_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4
\end{aligned}$$

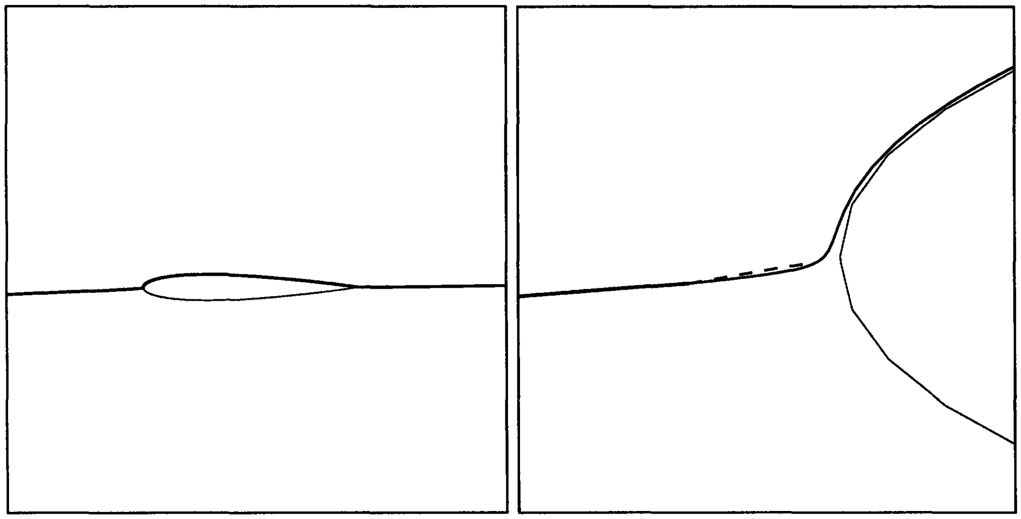
Since the convolution is done along the streamline, calculating the post-processed solution for intermediate stages that are not on the path is inconsistent and may lead to unquantifiable errors. The RK2 method uses one intermediate stage, and that is a half timestep in the future along the streamline. Even though this method has only second-order convergence on continuous fields, this is the only one that can be applied with this post-processing technique. Multistep methods were avoided due to their introduced error in the startup phase.

3.8 Streamline results

For the aforementioned reason the post-processing technique mentioned above was implemented with a Runge-Kutta 2 time integrator.

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 \end{aligned}$$

This method has second order convergence. It requires two post-processing per iteration, since the solution has to be calculated at y_n and $y_n + \frac{h}{2}k_1$.



(a) Streamline close to the airfoil (b) Closer look at the streamlines

Figure 3-9: Streamline results on 2D transonic case. Dashed line is streamline without post-processing. Both streamlines were created with a $\Delta t = .01$

First, streamline results are shown on the same two dimensional transonic airfoil solution as in Section 2.6.1. Figure 3-9 shows two streamlines with the same starting origin. Both of these were traced by an RK2 method with a $\Delta t = .01$, but the dashed line was calculated without post-processing and it gets stuck close to the boundary as this Δt proves to be too large. The calculation time of the post-processed solution took roughly four times longer than the simple integration scheme. If Δt is reduced to .0015 then the non post-processed streamline can also be fully evaluated but this would result in longer total rendering time.

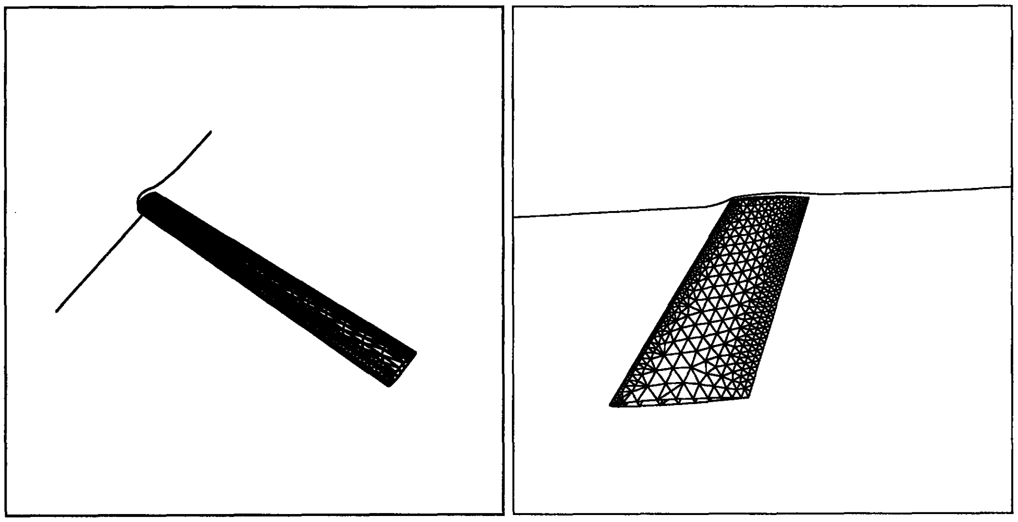

(a) Streamline close to the wingtip (b) Streamline close to the wingtip

Figure 3-10: Streamline results on 3D transonic case. Two different views of the wing and the streamline.

Next, a three dimensional example is shown. Calculations were done on the same case as in Section 2.6.1, a three dimensional wing in $M = 1.5$ conditions. A streamline was traced close to the wingtip and the results are shown on Figure 3-10.

Finally, a two dimensional viscous case was used. Viscid flow was calculated around a NACA 0012 airfoil with $\alpha = 1.25$, $M = .5$ and $Re = 5 * 10^{-3}$. Figure 3-11 shows a streamline close the body, and separation can be observed towards to the trailing edge.

Figure 3-12 shows a streamline in the separated region at the trailing edge. It can be seen that the non post-processed streamline (top two images) spirals inwards towards a critical point. This is a physically impossible solution. However, for the streamline calculation done with post-processing (bottom two images), the streamline spirals out and finally leaves the airfoil. This example demonstrates how, when using ODE integrators, errors can accumulate and produce erroneous results especially in the presence of critical points.

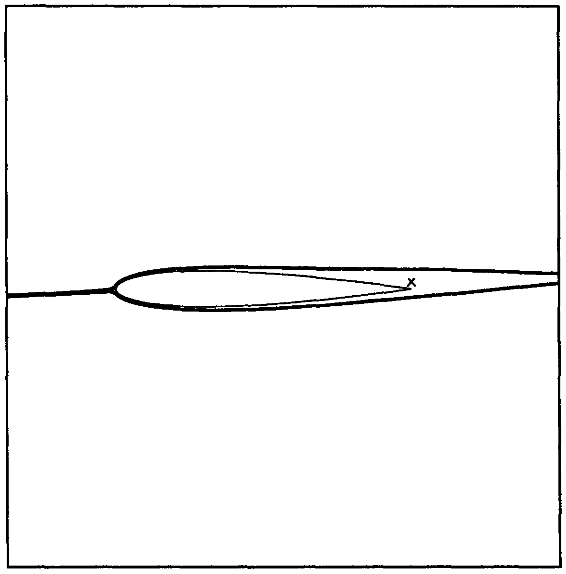
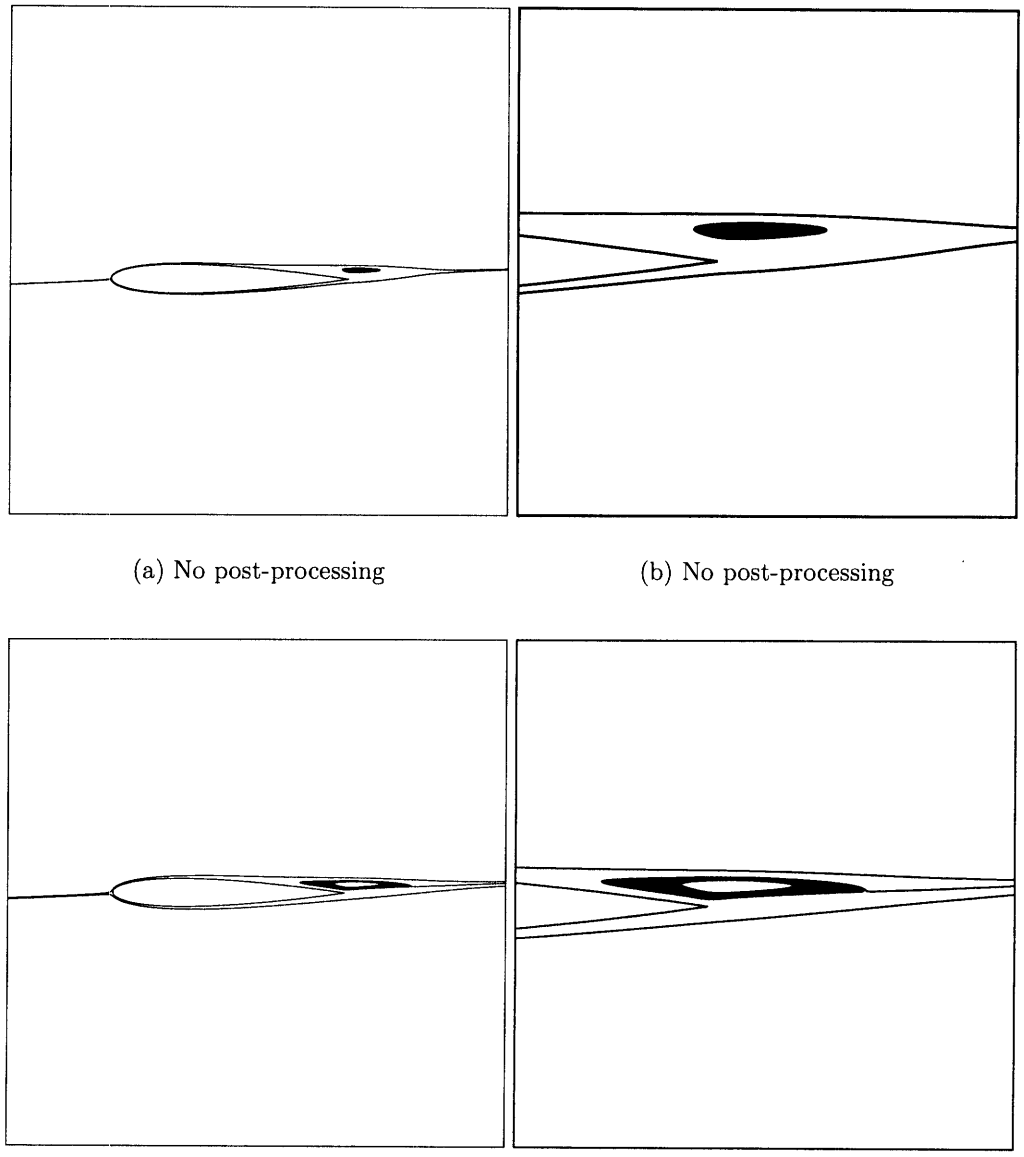


Figure 3-11: Streamline close to the airfoil. Separation can be seen towards the trailing edge. The x location marks the starting point of the streamlines on Figure 3-12.



(a) No post-processing (b) No post-processing

(c) Post-processed (d) Post-processed

Figure 3-12: Streamlines at the separated region. Streamline starting location is the same for both cases. $\Delta t = .1$

Chapter 4

Conclusions and Future Work

4.1 Conclusions

First, a visualization-error based anisotropic mesh adaptation algorithm was presented for both unstructured triangular/tetrahedral body-conforming and simplex cut cell meshes. The technique, while introducing a pre-processing step to the visualization, reduces the total element number needed for accurate higher order CFD visualization. The reduction in element number results in less memory usage and faster rendering time. The computational power saved depends on the order of the solution polynomial. While it is not necessary for lower order solutions, it can prove extremely effective for high order solutions.

An isotropic refinement technique was also introduced for simplex cut-cell solutions. This method has the advantage of neglecting refined elements that are completely outside the flow domain. By not passing these cell informations to the visualizer, the technique reduces rendering time.

A post-processing technique was also introduced for accurate streamline calculations. The method uses an on-the-fly accuracy conserving, smoothness improving post-processor with a Runge-Kutta second order method to trace streamlines. By eliminating the inter-element discontinuities from DG solutions, the order of convergence of time-integrators can be conserved. With this method, incorrect integration close to critical points can be avoided and accurate streamlines can be traced even with bigger time steps. Even though, the post-processor is computationally expensive, in some situations it is necessary for the time integrator not to fail due to discontinuities in the flow solution.

4.2 Future work

During the development of these techniques, a few ideas came to mind in order to either improve this work or add as possible continuation of it to different problems.

1. **Embedded surface adaptation**

   For three dimensional cut cell results, the visualization on the surface of the body is done on the quadratic patches. An additional embedded surface is added for all of these boundaries and the solutions is visualized on these. Since both the geometry and the solution can be higher-order, the standard visualization technique's linear rendering introduces errors. The two dimensional adaptation technique can be applied on these surfaces in order to both improve the accuracy of the body and the solution visualization.

2. **Parallelize adaptation algorithm**

   The pre-processing step of the mesh adaptation can be time consuming for high order solutions. Since the algorithm adapts one element at a time without any influence from any of its neighboring cells, the process can be parallelized in a simple fashion.

3. **Better time integration technique**

   The currently used RK2 solver is explicit and has second order convergence. For streaklining, it has been proven that implicit methods, specially backwards differentiation, is more effective. In order to be able to use these algorithms, a continuous solution is not enough, the first derivative of the solution needs to be smooth also. By applying a modified post-processor, continuity on the first derivative can be achieved and more accurate time-integrators can be implemented for the streamliner.

4. **Better sampling for the post-processor**

   The convolution requires the solution at quadrature points along the streamline. Currently the location of the quadrature points are calculated as the linear interpolation of the section of the streamline. The solutions however is the real higher-order value at the location. In order to improve the calculation of the quadrature point coordinates, a higher-order curve fitting method could be applied to the sample points of the streamlines.

Bibliography

[1] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2d Euler equations. *Journal of Computational Physics*, 138(2):251–285, 1997.

[2] J. H. Bramble and A. H. Schatz. Higher order local accuracy by averaging in the finite element method. *Mathematics of Computation*, 31:94–111, 1977.

[3] R. Burden and J Faires. *Numerical Analysis.* PWS Publishing Company, 1993.

[4] B. Cockburn, M. Luskin, C.-W. Shu, and E. Suli. Enhanced accuracy by post-processing for finite elemen methods for hyperbolic equations. *Mathematics of Computation*, 72:577–606, 2003.

[5] B. Cockburn and C.-W. Shu. Runge-kutta discontinuous Galerkin methods for convection-dominated problems. *Journal of Scientific Computing*, pages 173–261, 2001.

[6] R.A. Crawfis and N. Max. Textured splats for 3d scalar and vector field visualization. In *Visualization '93*, pages 261–272. IEEE Computer Society Press, 1993.

[7] S. Curtis, R. M. Kirby, J. K. Ryan, and C.-W. Shu. Post-processing for the discontinuous galerkin method over non-uniform meshes. *SIAM Journal of Scientific Computing*, 2007.

[8] S. Curtis, M. Steffen, R. M. Kirby, and J. K. Ryan. Investigation of smoothness-enhancing accuracy-conserving filters for improving streamline integration through discontinuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

[9] D.L. Darmofal and R. Haimes. An analysis of 3-d particle path integration algorithms. *Journal of Computational Physics*, 123:182–195, 1996.

[10] W.C. de Leeuw and Jarke J. van Wijk. Enhanced spot noise for vector field visualization. In *Visualization '95*, pages 233–239. IEEE Computer Society Press, 1995.

[11] W.C. de Leeuw and J.J. van Wijk. A probe for local flow field visualization. In *Visualization '93*, pages 39–45. IEEE Computer Society Press, 1993.

[12] B. Delaunay. Sur la sphere vide. *tdelenie Matematicheskikh i Estestvennykh Nauk,*, 7:793–800, 1934.

[13] K. J. Fidkowski. A simplex cut-cell adaptive method for high-order discretizations of the compressible navier-stokes equations. Doctoral thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2007.

[14] K.J. Fidkowski, T.A. Oliver, J.Lu, and D.L. Darmofal. p-multigrid solution of high-order discontiunous Galerkin discretizations of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 207(1):92–113, 2005.

[15] C. W. Gear and O. Osterby. Solving ordinary differential equations with discontinuities. *ACM Transactions on Mathematical Software*, 10(1):23–44, 1984.

[16] A. Glassner. *An Introduction to Ray Tracing.* Academic Press, 1989.

[17] H. Hou and H. Andrews. Cubic splines for image interpolation and digital filtering. *IEEE Transactions on acoustics, speech, and signal processing*, 26:508–517, 1978.

[18] J.D. Anderson Jr. *Fundamentals of Aerodynamics.* McGraw-Hil, 1984.

[19] K.-L. Ma and P. J. Smith. Virtual smoke: An interactive 3d flow visualization technique. In *Visualization '92*, pages 46–52. IEEE Computer Society Press, 1992.

[20] D. Mitchell and A. Netravali. *Reconstruction filters in computer-graphics.* Proceedings of the 15th annual conference on Computer graphics and interactive techniques, 1988.

[21] M. S. Mock and P. D. Lax. The computation of discontinuous solutions of linear hyperbolic equations. *Communications on Pure and Applied Mathematics*, 31:423–439, 1978.

[22] T. Moller, R. Machiraju, K. Mueller, and R Yagel. Evaluation and design of filters using taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.

[23] T. Plewa, T. Linde, and V. G. Weirs. *Adaptive Mesh Refinement - Theory and Applications.* Springer Verlag, 2005.

[24] F.J. Post and T. van Walsum. Iconic techniques for feature visualization. In *Visualization '95*, pages 288–295. IEEE Computer Society Press, 1995.

[25] W. H. Press, S. A. Teukolsy, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1997.

[26] J.W. Purvis and J.E. Burkhalter. Prediction of critical mach number for store configurations. Technical Report 11, 1979.

[27] D.J. Quattrochi. Hypersonic heat transfer and anisotropic visualization with a higher order discontinuous galerkin finite element method. Masters thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2006.

[28] J.-F. Remacle, N. Chevaugeon, E. Marchandise, and C. Geuzaine. Efficient visualization of high-order finite elements. *International Journal for Numerical Methods in Engineering*, 69(4):750–771, 2006.

[29] J. K. Ryan, C.-W. Shu, and H. Atkins. Extension of a post-processing technique for the discontinuous galerkin method for hyperbolic equations with application to an aeroacoustic problem. *SIAM Journal on Scientific Computing*, 26:821–843, 2005.

[30] J.K. Ryan and C.-W. Shu. On a one-sided post-processing technique for the discontinuous galerkin methods. *Methods and Applications of Analysis*, 10(2):295–308, 2003.

[31] J.J. van Wijk. Rendering surface-particles. In *Visualization '92*, pages 54–61. IEEE Computer Society Press, 1992.