TETRAHEDRAL MESH OPTIMIZATION AND GENERATION VIA TOPOLOGICAL TRANSFORMATIONS AND GRADIENT BASED NODE PERTURBATION

By

Christopher Bruce Hilbert

Steve L. Karman, Jr.
Professor of Computational Engineering
(Chair)

James C. Newman III
Professor of Computational Engineering
(Committee Member)

Sagar Kapadia
Research Assistant Professor of
Computational Engineering
(Committee Member)

Vincent C. Betro
Computational Scientist, JICS/NICS
(Committee Member)

TETRAHEDRAL MESH OPTIMIZATION AND GENERATION VIA TOPOLOGICAL TRANSFORMATIONS AND GRADIENT BASED NODE PERTURBATION

By

Christopher Bruce Hilbert

A Dissertation Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy in Computational Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2015

ABSTRACT

A general tetrahedral mesh optimization scheme utilizing both topological changes (i.e. flips) and gradient-based vertex optimization (i.e. smoothing) is demonstrated. This scheme is used in the optimization of tetrahedral meshes created by third-party software as well as a grid generation methodology created for this work. The particular algorithms involved are explained in detail including, an explication of the primary optimization metric, weighted condition number. In addition, a thorough literature review regarding tetrahedral mesh generation is given.

DEDICATION

This work is dedicated to my little girl and boy, Abbie and Zeke, who, though they are only seven and five years old, already make fun of their dad for being a nerd.

ACKNOWLEDGEMENTS

The number of people who have helped me to this point are too many to remember. As with any person who made it this far in their academic career (or any major life endeavor), I have to thank a legion of teachers, students, mentors, friends, and family who have challenged and supported me along the way. In particular let me mention the following: My former teachers and fellow students Nancy Caldwell, Mary Richardson, Jonathan Innes, Sonny Tendian, David Foreman, Bruce Atkinson, James Brown, Wally Edmondson, Nick Currier, and Taylor Erwin. My lifelong friends Zachary Cullis, Paul Klintworth, and Michelle Matson who have always supported me during the challenging times of life. My colleague and friend Ethan Hereth for taking up the slack I left while working on this dissertation. My advisor, Dr. Steve Karman, who took me on as a student even after I crashed his computer. My parents, Gary and Debbie. I could not have had better. My wife for loving me and putting up with a husband who has been in school her whole married life.

Finally, I give praise to the Lord God, Father, Son, and Holy Spirit. I can do all things through Christ who strengthens me . . . even CFD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

A_k, Jacobian matrix of tetrahedra A taken from vertex k

AR, aspect ratio

CN_k, tetrahedral condition number taken from vertex k

c_W, tetrahedral cost based on weighted condition number

δ_n, threshold perturbation distance

E, an edge of a tetrahedron

ϵ, an infinitesimal number

$J(T)$, the Jacobian of a tetrahedron

LSOI, Lawson Style Optimization Insertion

N, a list or set of nodes

$\mathbf{p}$, perturbation vector

T, a tetrahedron

Θ, a list or set of tetrahedra

W, weight matrix in weighted condition number

WCN, tetrahedral weighted condition number

$\mathbf{x}$, coordinates of vertex/node

$\|\cdot\|_F$, Frobenius norm

CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

Grid generation is one of the most vital and difficult parts of the computational simulation process. Acceptable grid quality is a necessary condition to ensure the efficacy of any finite difference, finite volume, or finite element simulation. While grid quality can be an elusive goal, experience shows that element shape plays an important role in affecting the stability of a particular solution procedure as well as affecting phenomenon such as dissipation and diffusion. Grid angles approaching 0° or 180° often cause computational difficulties and, thus, elements with as "regular" a shape as possible are desired for quality solutions. An oft cited reference for the effects of element angles on solution accuracy can be found in Babuska [1]. Shewchuk [2] also offers a good overview on the subject. Of course, grid resolution is equally important. No phenomenon can be accurately modeled without adequate resolution.

Finite volume and finite element methods most often make use of unstructured grids. By far, the most common element type utilized in three-dimensional unstructured grids is the tetrahedron. This prevalence is due largely to the fact that tetrahedra are the simplest of the three-dimensional polyhedra. In that vein, a great deal of academic work has focused on constructing tetrahedral grids. This dissertation focuses on a technique for the creation and quality-driven optimization of such tetrahedral grids.

The research herein tackles two separate but closely related problems in tetrahedral grid construction:

- Given a set of triangles in three-dimensional space that form a closed manifold, construct a tetrahedral grid suitable for computational simulations
- Given a tessellation of some space composed of tetrahedra, improve the quality of the tetrahedralization with respect to its suitability for computational simulations

What follows is a general overview of mesh generation and optimization techniques that are important to developing algorithms to solve the above problems. Every attempt has been made to provide a comprehensive, but not necessarily detailed, treatment of the subject of tetrahedral mesh generation. An introduction to the Delaunay property, two and three-dimensional mesh creation algorithms, topological mesh manipulation, and mesh improvement through vertex smoothing will all be highlighted. There are a number of other sources that give a reasonable overview of tetrahedral mesh creation and improvement techniques. A few such general treatments are given by Bern [3], George [4], and Danilov [5].

1.1 Definitions

In grid generation there are often many terms for the same thing. In this work, we will make use of the following terminology:

- The terms *grid* and *mesh* will be used interchangeably as will the terms *node* and *vertex*.

- A *simplex* will refer to an element, either a triangle or tetrahedron. In that light, triangle and tetrahedron will refer to their irregular (i.e. non-equilateral) forms unless otherwise indicated.

- *Smoothing* will refer to the movement of vertices to improve mesh quality.

- Changes to the mesh connectivity will be formally referred to as *topological changes* and more informally as *swapping* or *flipping*.

- *Quality* will refer, loosely, to element shape and the efficacy of the mesh for computational simulation.

- *Optimization* will refer to mesh quality improvement through whatever means.

1.2 The Delaunay Property

Most of the tetrahedral grid construction techniques are three-dimensional analogues of processes developed to create two and three-dimensional triangle grids. In particular, most tetrahedral grid generation techniques rely on a feature called the Delaunay [6] property as a way of maintaining element quality. The Delaunay property simply ensures that the circumscribing sphere of any tetrahedra in the grid contains no vertices from any other tetrahedra. In two-dimensions, this property is equivalent to stating that no circle circumscribing any triangle contains the vertex of any other triangle. A simple illustration of the Delaunay property in two-dimensions is given in Figure 1.1. Notice the red vertex is inside a neighboring triangle's circumcircle and, thus, the Delaunay property is violated. Note it is possible for points that form two adjacent triangles or tetrahedra to

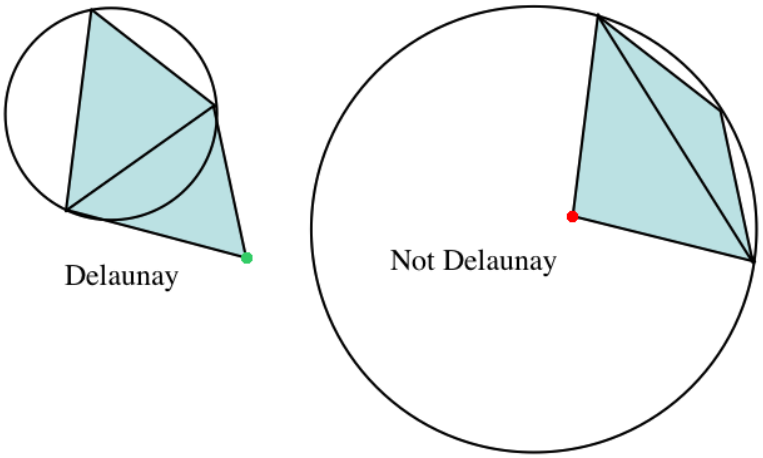


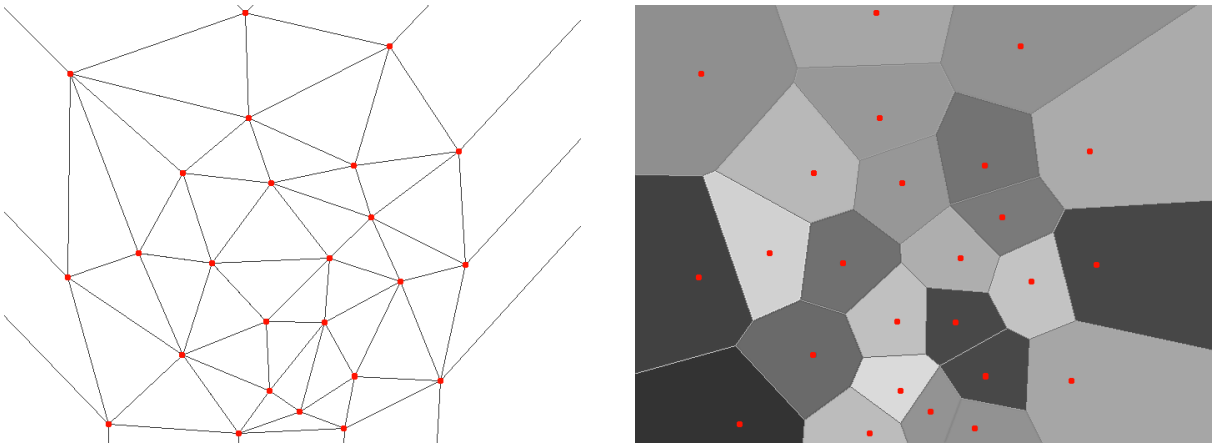
Figure 1.1 Delaunay versus Not Delaunay

be co-circular or co-spherical. Thus the circumcircle/circumsphere will, strictly speaking, contain a non-defining vertex. In these cases, any valid connectivity is taken to meet the Delaunay property.

Delaunay showed that any triangulation with the property described above was the dual mesh of the Voronoi [7] diagram. The Voronoi diagram is the partitioning of a plane with a given set of points into regions all of whose points are closer to one of the given points than any other given point. More formally, a Voronoi region of the plane (or hyperplane) X surrounding a given point $p_i \in \mathcal{P} \subset X$ is

$$R = \left\{ x \in X \middle| dist\,(x, p_i) < dist\,(x, p_k)\ \forall p_k \in \mathcal{P} \right\} \tag{1.1}$$

where $\mathcal{P}$ is the given set of points. This partitioning is also often called a Dirichlet tessellation as it tiles the plane. An illustration of this dual relationship between the Voronoi diagram and Delaunay triangulation is given in Figure 1.2. (These graphs were created with the help of Chew's online



(a) Delaunay triangulation (b) Voronoi diagram

Figure 1.2 A Delaunay Triangulation and Voronoi Diagram of the same set of vertices (in red.)

applet [8].) The Delaunay triangulation/tetrahedralization of a plane/space is unique for a given set of points, excepting the possibility of co-circular/co-spherical points.

Indeed there are many reasons to pursue the Delaunay property as a goal and means of creating triangular and tetrahedral meshes. For example, Sibson [9] has shown that the Delaunay triangulation of a set of points is the most nearly equiangular triangulation attainable. That is, the Delaunay triangulation of a set of points will, in some sense, be the highest quality for that set of points. Also, as will be discussed in Section 1.3, utilizing the Delaunay property during node insertion provides a way to eliminate poor quality elements.

However, there is no direct correlation between maintaining the Delaunay property and the efficacy of a tetrahedral unstructured grid vis-à-vis its utility for computational analysis. While there are several proofs about particular angle-based quality criteria for two-dimensional Delaunay grids, such strong guarantees do not exist for three-dimensional tetrahedralizations. Rajan [10] has shown that a Delaunay mesh of any dimension minimizes the maximum radius of any sphere

circumscribed about any element of the mesh. However, this does not guarantee the optimum angles desired for simulations. For example, four points that are nearly coplanar will have a circumsphere that is relatively small and, thus, contains no other points in the mesh. These tetrahedra are often refereed to as "kites" or "slivers." Such elements do not violate the Delaunay property but, since they have angles between faces that approach 0° and 180°, are poor elements for computational simulation. Figure 1.3 shows such an element from a Delaunay tetrahedralization. Moreover, simply creating a Delaunay mesh does not solve the grid generation problem vis-à-vis a grid's utility for computational analysis. In particular, care must be taken to ensure that the triangles used to define the geometric boundaries of the model being analyzed are present in the final mesh, i.e., the boundaries are recovered. Also, one must insert enough points at the correct locations to achieve an accurate simulation. As such, methods are proposed herein that, while making use of the Delaunay criterion, ultimately do not completely respect that specification.

Figure 1.3 A “Kite” in a Delaunay Tetrahedralization

1.3 Two-Dimensional Algorithms

What follows is a discussion of two-dimensional triangulation algorithms. While this work assumes that a suitable triangulation of the surface defining the space to be tetrahedralized has been generated, most of the three-dimensional mesh creation and optimization strategies are extensions of two-dimensional triangulation algorithms.

1.3.1 Lawson and Bowyer-Watson

Much of the inspiration for this work comes from an early researcher in the problem of triangulation, C. L. Lawson. Lawson [11] introduced one of the earliest algorithms for triangulating a set of points. That algorithm is summarized as follows: First the points are ordered by Euclidean

distance from the point with the least x coordinate. An initial triangle is constructed from the first three non-collinear points in this ordering. Then, one-by-one, the points are connected to the bounding edges of the current triangulation.

Note that, due to the initial ordering of the points, Lawson's algorithm could simply be thought of as a sweep line insertion. The unique part of this algorithm is the edge "swapping" technique. As each triangle is created, that triangle and its edge neighbor are evaluated based upon one of three quality criteria: the max-min angle criterion, the circle criterion, or the Thiessen region criteria. The max-min criterion chooses to swap the common edge between two triangles if the new triangulation increases the minimum included angle as compared the original triangulation. The circle criterion chooses to swap the edge if the circumscribing circle of one the triangles in question contains the opposite vertex of the other triangle. The Thiessen region refers to the region around point P that is closer to point P than any other specified point in the plane. Two given points are said to be Thiessen neighbors if their Thiessen regions contact one another. Lawson swaps edges to insure that all Thiessen neighbors are connected. Most importantly, Lawson shows that these three criteria produce equivalent meshes when used to determine edge swapping.

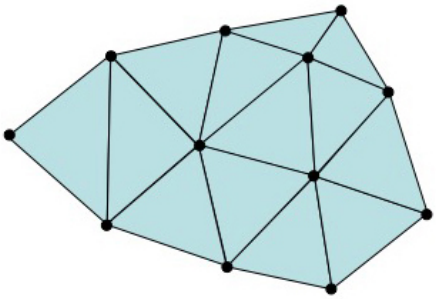
Although not stated by Lawson, the circle criterion should be simply construed as maintaining the two-dimensional Delaunay property. Also, note that the aforementioned Thiessen regions are precisely equivalent to Voronoi partitioning although, again, Lawson does not reference this fact. There is no obvious reason Lawson does not use what have become the more accepted terms for these properties; but Lawson does reference sources (particularly for Thiessen regions) that, apparently, also use these terms.

The more commonly employed triangulation algorithm, and the one that inspires the most three-dimensional tetrahedralization algorithms (see Section 1.4) , was developed independently and published simultaneously in the same journal by Bowyer [12] and Watson [13]. Their algorithm is summarized as follows: Given a Delaunay triangulation, insert a new point into the triangulation by finding all the triangles whose circumcircles contain the new point. Delete these triangles and connect the points to form new triangles appropriately. This algorithm is generally known as "Boywer-Watson" insertion. An illustration of this is given in Figure 1.4.

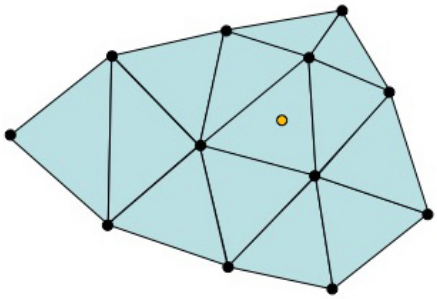
It should be noted that Bowyer does not specify triangle removal in terms of circumcircles but instead in terms of the Voronoi partitioning. Watson specifies an initial simplex into which to insert the first given point while Bowyer simply takes the first several specified points to form the initial Voronoi partitioning. Both authors substantiate that their algorithm works for any dimensionality; but, for obvious reasons, they only give examples in two and three-dimensions.

Lawson's methodology can be applied to point insertion in a manner similar to Bowyer-Watson. Instead of removing all triangles whose circumcircle contains the new point, simply connect the point to the vertices of the triangle in which it is found to form three new triangles (deleting the original triangle.) Subsequently, use a series of flips to ensure the Delaunay property is maintained. A illustration of this process is given in Figure 1.5.

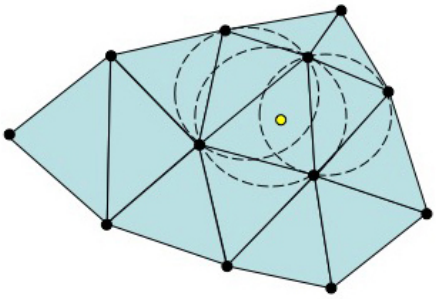
In two-dimensions, Lawson's algorithm will result in the same mesh as Bowyer-Watson, excluding the co-circular case. This result is due to the fact that any two-dimensional grid that is everywhere locally Delaunay will, obviously, be globally Delaunay. Unfortunately, that property does not entirely hold in higher dimensions. More discussion of this quandary can be found in Section 1.5.



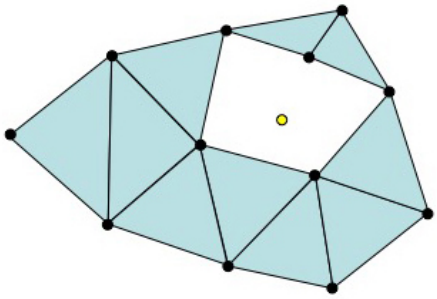
(a) initial mesh



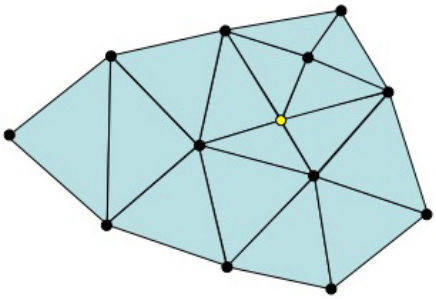
(b) point location



(c) circumcircle tests

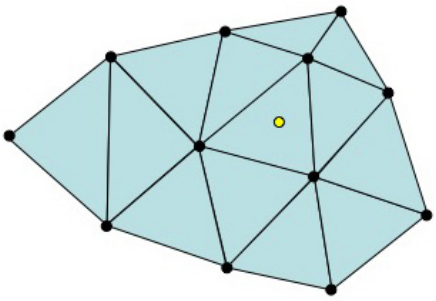


(d) violated triangles deleted

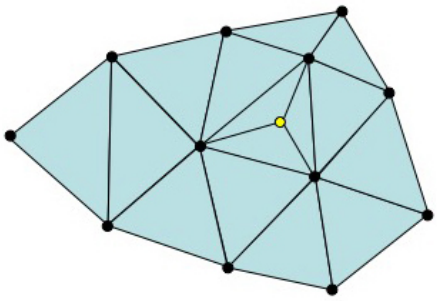


(e) inserted point connection

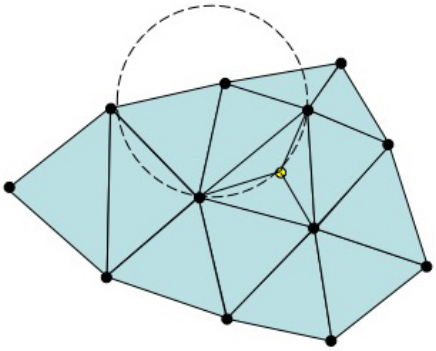
Figure 1.4 Bowyer-Watson Algorithm Example



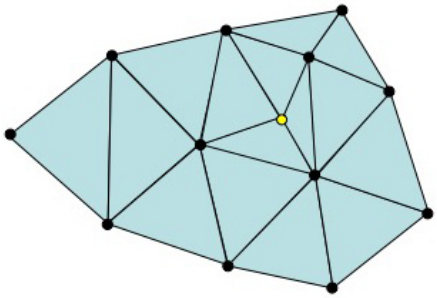
(a) point location



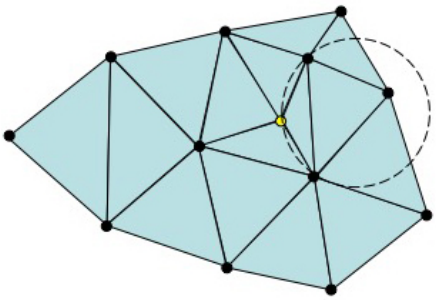
(b) point inserted



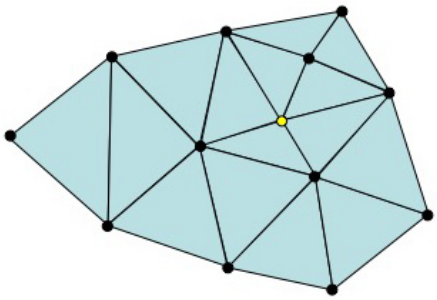
(c) first violated triangle test



(d) first flip



(e) second violated triangle test



(f) second flip

Figure 1.5 Lawson's Algorithm Example

Also note that neither the Bowyer-Watson nor Lawson method address the issue of ensuring particular edges, such as those that compose a set of boundaries, are extant in the final mesh. In that vein, Chew [14] has shown that with a given set of n vertices and a prescribed set of edges, a constrained Delaunay mesh can be constructed in $O(n \log n)$ time. A constrained Delaunay triangulation is a triangulation which respects the set of prescribed edges but is, otherwise, as close to maintaining the Delaunay property as possible. A similar concept of a constrained Delaunay tetrahedralization that respects a set of prescribed triangles is often used for three dimensional meshes. However, neither Bowyer-Watson nor Lawson nor this particular work by Chew prescribe methods to generate the set of points to be triangulated. That is, not everything has been introduced to solve the problem of creating a complete computational mesh.

1.3.2 Two-Dimensional Mesh Creation

Within the literature, there have been many methods proposed for point creation in two-dimensional meshes. Chew [14, 15] presented the first two-dimensional meshing algorithm with provable size and shape bounds. Chew's algorithm is very popular and forms the basis of many other two and three-dimensional Delaunay based procedures. A summary of the algorithm is a follows:

- Choose a parameter h such that all boundary edge lengths are between h and $h\sqrt{3}$.
- Form the constrained Delaunay triangulation of the boundary edges.

- Beginning with this original triangulation, while there exists a triangle with circumradius greater than h, add the center of that triangle's circumcircle to triangulation using a Bowyer-Watson style insertion.

The procedure provably terminates in a triangulation wherein all edges have length less than $2h$ and all angles are between 30° and 120°. This result, due to its guaranteed quality constraint, makes Chew's algorithm extremely appealing, even with the input boundary length restriction. As such, much work has been done to extend the result.

Ruppert [16–18] built on Chew's result by allowing triangles that vary in size. Ruppert's work guarantees a triangulation with angles between 20° and 160° with aspect ratios less than $2/\sin 20° \approx 5.85$. More importantly, unlike Chew's algorithm where all elements are of roughly equal area, the triangles will vary smoothly in size between small and large input boundary segments. It should be noted that, in Ruppert's approach, the boundary segments themselves are refined. Thus, Ruppert's approach does not necessarily recover a prescribed set of segments.

Alternative approaches to generating and triangulating a set of points have been developed. Bern et al. [19] introduced a quadtree methodology that produces triangles with an aspect ratio less than 4 and/or angles with quality bounds (depending upon the particular implementation chosen.) Marcum [20–22] gave details for point insertion based upon the boundary definitions that produce nearly isotropic triangles. See Section 1.4.2 for a more detailed explanation. H. Chen and Bishop [23] proposed a way to mesh surfaces in three dimensions by transforming the circumcircles of the triangles into the parametric space of the surface. This technique produces better results than simply meshing in the parametric space and transforming the mesh back into real coordinates.

1.4 Delaunay Tetrahedralizations

The majority of three-dimensional tetrahedralization algorithms utilize a form of the Bowyer-Watson algorithm and make every attempt to maintain the Delaunay property. There are broad issues that must be overcome to make these mesh generation algorithms viable. Notably, there are efficiency and numerically related robustness issues associated with the point insertions and generation of the set of tetrahedra to be deleted. Little discussion will be given here to these issues here except to say that those sorts of problems have been mostly solved. In particular, Borouchaki and George [24] have developed a highly efficient Bowyer-Watson insertion scheme based upon finding the next point for insertion via a random walk. Shewchuk [25] has also done work towards solving such problems involving infinite precision arithmetic.

More important to this work, one must be concerned with recovering a boundary that consists of a set of triangles. Further, there is the question of how to construct points to generate quality elements on the interior of the mesh. What follows is an overview of these two issues.

1.4.1 Boundary Recovery

The recovery of the prescribed triangular boundary is itself a perplexing problem worthy of an entire dissertation. Perhaps the most distressing fact in the quest for a tetrahedral mesh comes from Ruppert and Seidel [26] who showed that determining if a polyhedra can be tetrahedralized at all without the addition of a specified number of vertices is NP-complete. That is, the problem is theoretically equivalent to the most challenging and intractable problems of computational theory. Thus, there may be triangular boundary configurations that could themselves only be tetrahedralized by checking all possible configurations. This result implies the problem would

take $O(n!)$ operations to solve! Still, from a practical point of view, boundary recovery is well understood.

Paul Louis George et al. [27–29] have presented most of the seminal work on recovering a particular triangular boundary. George's methodologies consists largely of making topological changes to the mesh in order to recover the prescribed triangulation. Shewchuk [30, 31] has also employed an edge swapping based boundary recovery scheme. However, in [27] George has proven that boundary recovery is provably obtainable if one is allowed to insert an arbitrary number of points into the triangulation itself. Si and Gartner [32] introduced an algorithm utilizing a constrained Delaunay approach to insert points and provably recover the boundary. Du and Wang [33] presented work which combined point insertion and topological changes that also provably recovered the boundary. Liu et al. [34] improved upon this procedure by constructing the order in which these points are inserted so that they can be easily removed.

Ghadyani et. al. [35] presented an innovative algorithm based upon element removal called "Last Resort." This algorithm creates a hole in the mesh containing the points of the deleted elements. The number of possible ways to re-mesh this hull is, as alluded to above, $O(n!)$. However, Ghadyani employs a tree structure to make the procedure more tractable, reducing the operations to $O(n)$. For boundary recovery, the hole is merely created at the surface with its hull defined by the prescribed triangulation.

1.4.2 Internal Point Generation

There are a variety of methodologies used to determine where precisely points should be placed internally to the surface triangulation to create a suitable computational mesh. Algorithms

to attack the worst quality element, quad-tree insertion techniques, and boundary dependent point placement are all routinely used. Most three-dimensional point placement algorithms are merely analogues of some two-dimensional algorithm. However, as was stated before, most techniques are based-on, or at least utilize, the Bowyer-Watson point insertion methodology.

Placing new points to guarantee the removal of poor quality elements is commonly utilized. Chew [36] proposed an algorithm for tetrahedralization of a domain based on the developed two-dimensional scheme. However, instead of placing points at exactly the center of the circumsphere of a tetrahedron, a point is randomly placed somewhere within one-half radius a circumsphere. This candidate point is accepted only if it produces no slivers. (See Section 1.2.) If slivers are produced by the new point, a new random point is chosen and the check is repeated until no slivers are produced. This process guarantees the deletion of a poor quality element and, as Chew proved, will create meshes with angles between tetrahedral faces bounded by approximately 14.5° and 151°. Unfortunately, this guarantee requires that the boundary triangulation be convex and meet some stringent quality and length constraints.

Shewchuk [25,37,38] developed a three-dimensional tetrahedral mesh generation algorithm that built upon Ruppert's two-dimensional algorithm. (See Section 1.3.2.) This algorithm can provably produce Delaunay meshes with circumradius-to-shortest-edge ratio no greater than two as long as the input segments and facets are separated by angles of at least 90°. The procedure first splits the prescribed boundary segments and faces, thus refining those segments and faces as was done by Ruppert. Vertices are then inserted at the circumcenter of tetrahedra with circumradius-to-shortest edge ratio above some prescribed bound. Shewchuk also introduced a number of innovations including the use of an "equatorial lens" to protect subfacets of a boundary face from

refinement, an explanation of the use of arbitrary precision floating point arithmetic as it pertains to grid generation, and (most importantly) a large number of proofs about good-quality and good-grading (i.e. smooth transition from elements of small volume to element of a larger volume.) Note that, just as with Ruppert, Shewchuk's original algorithm did not enforce a strict boundary recovery, only broadly defined segments and facets which are themselves refined. However [39] and [30] outline a sweep algorithm utilizing the aforementioned subfacet protection to create a mesh with a given set of facets.

As a doctoral thesis, Si [40] developed an algorithm much like Shewchuck's that incorporated a Delaunay refinement routine with guaranteed termination. Si [41] has also shown that Shewchuk's boundary angle requirement can be reduced from 90° to $\arccos \frac{1}{3} \approx 70.53°$. A tetrahedral mesh generation code developed by Si, TetGen[1] [42], is freely available online.

There have been many other variations utilizing the Bowyer-Watson insertion technique. Borouchaki et. al [43] generated points by comparing current edge lengths in the mesh with a local step size based upon the boundaries. New points are located along the edges utilizing an algebraic distribution and then inserted using a Bowyer-Watson scheme. Miller et. al. [44] showed a method that places points using a sphere packing algorithm which guarantees a bounded radius to edge ratio. Gosselin and Ollivier-Gooch [45] presented an algorithm that inserts points based on various quality measures with the goal of removing sliver tetrahedra.

Marcum [20–22, 46] has created some very high quality (and visually impressive) meshes using "Advancing Front/Local Reconnection," or AFLR, point insertion techniques. Initially,

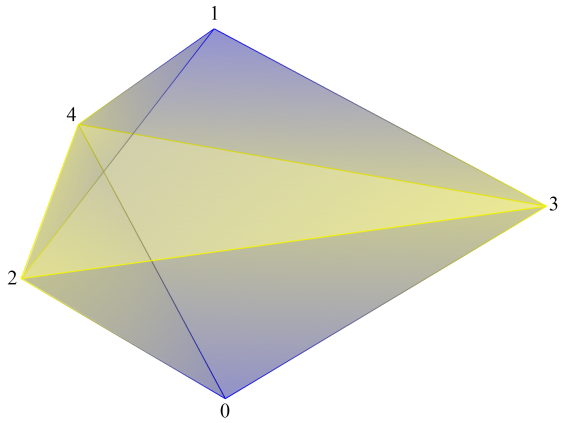
[1] http://wias-berlin.de/software/tetgen/

the triangular boundaries are recovered, somewhat ironically, with a Delaunay procedure. Point distribution functions are assigned to each boundary point representative of the local point spacing. New points are created by marching from the selected face, hence the term advancing front, with procedures implemented to ensure that the newly placed points are not close to existing points. Interestingly, each point is inserted in place and then edge and face flips are used, hence local reconnection, using a "combined Delaunay and min-max type criterion." (See Section 1.5.) Thus, because of the prescribed point placement, meshes are created with mostly isotropic tetrahedra (outside any prescribed boundary layer.) Working with Marcum, Weatherill [47–49] has given a method which produces results similar to AFLR meshes by inserting points in a set of existing tetrahedra based upon local spacing parameters.

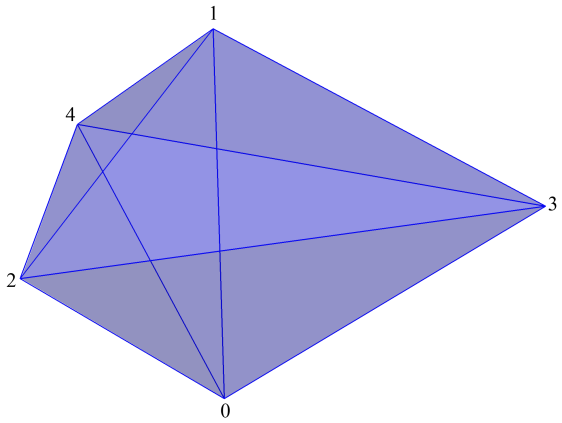
1.5 Edge and Face Removal

In two dimensions, the only topological change involving triangles that can be made is the simple edge swap originally used by Lawson. However, in three-dimensions with tetrahedra, the changes are not so simple. While only two tetrahedra may share a face there is no upper bound for the number of tetrahedra that may surround an edge. Nevertheless, there are practical ways to make flips with tetrahedra.

There are three basic topological changes that can be made in tetrahedral meshes: the 2-3 flip, 3-2 flip, and 4-4 flip. The 2-3 flip takes 2 tetrahedra that share a face and connects the two vertices not on that face with an additional edge. Thus, three tetrahedra are created around that edge and the two original tetrahedra are removed. See Figure 1.6 for an illustration. The 3-2 flip is simply the inverse of the 2-3 flip. An edge surrounded by three tetrahedron is removed to create two



(a) before: 2 – 3 – 4 – 1, 4 – 3 – 2 – 0

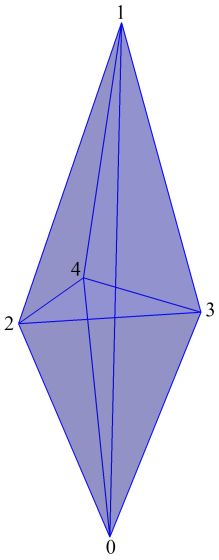


(b) after: 0 – 1 – 2 – 3, 0 – 1 – 3 – 4, 0 – 1 – 4 – 2

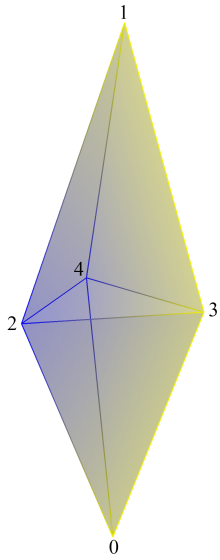
Figure 1.6 The 2-3 Flip

tetrahedra that share a face. The 3-2 flip is depicted in Figure 1.7. Finally, the 4-4 flip removes an edge surrounded by four tetrahedra and adds an edge between two of the four nodes not connected to the edge that was removed. Note there are potentially two valid choices for this new edge. Figure 1.8 shows an example of one choice for the 4-4 flip. George and Borouchaki [50] have shown that the 4-4 flip is necessary in creating an arbitrary topological change in that the 2-3 and 3-2 flips are insufficient to describe some transformations. Removing an edge surrounded by more than four tetrahedra will be considered in Section 2.2.1.

One of the most import aspects of face and edge swapping with regards to tetrahedral mesh generation and optimization is the fact that all such topological changes are local. That is, an edge or face removal or swap only affects simplices connected to that feature. Thus local improvements in mesh quality can be achieved without worrying about changing other elements of the mesh.

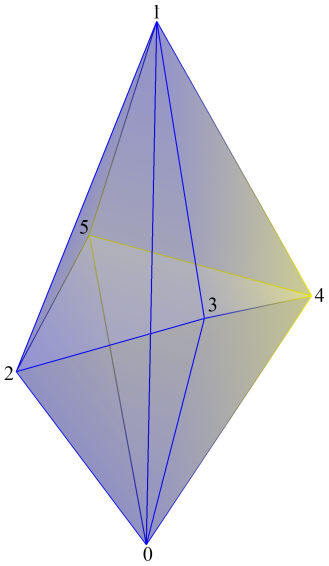


(a) before: 0 – 1 – 2 – 3, 0 – 1 – 3 – 4, 0 – 1 – 4 – 2

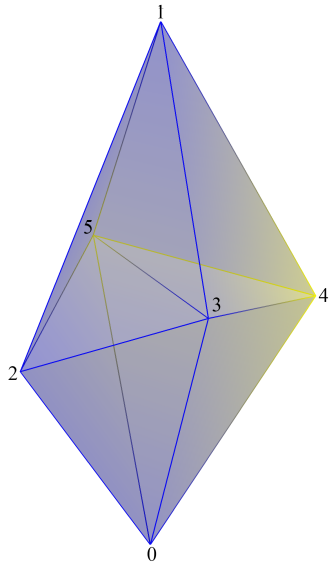


(b) after: 2 – 3 – 4 – 1, 4 – 3 – 2 – 0

Figure 1.7 The 3-2 Flip



(a) before: 0 – 1 – 2 – 3, 0 – 1 – 3 – 4, 0 – 1 – 4 – 5, 0 – 1 – 5 – 2



(b) after: 2 – 3 – 5 – 1, 5 – 3 – 2 – 0, 3 – 4 – 5 – 1, 5 – 4 – 3 – 0

Figure 1.8 The 4-4 Flip

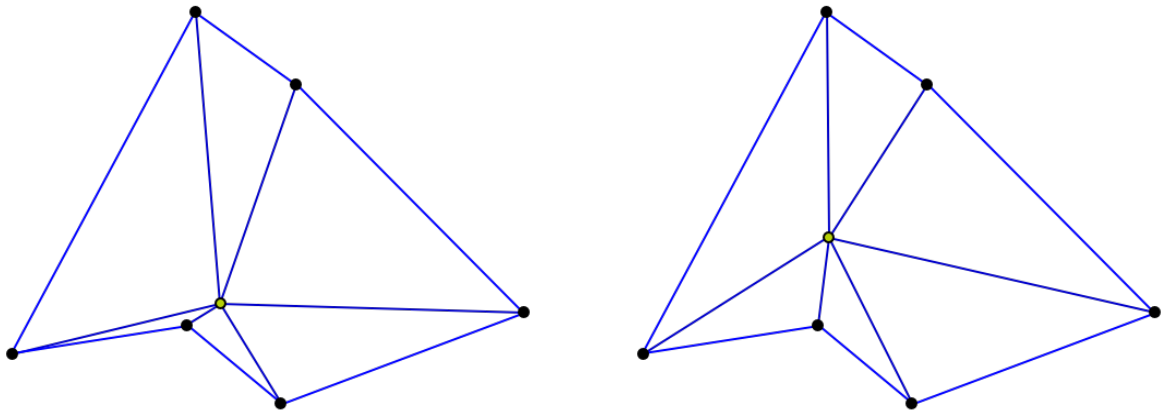
Others have explored topological changes to maintain a Delaunay mesh. Joe [51–53] gave an algorithm utilizing flips that successfully constructs the Delaunay tetrahedralization for any set of points. Moreover, Joe proved that any locally non-Delaunay set of tetrahedra could be transformed into a Delaunay tetrahedralization if no set of four points were coplanar. Unfortunately, Joe shows that these coplanar situations arise in cases that are not overly contrived or pathological. Further, in [54] Joe gave an approach to generating tetrahedral meshes based not upon the Delaunay property but upon maximizing the minimum solid angle. The algorithm gives good results. Overall, Joe's results were early evidence of the efficacy of using topological changes to improve a mesh despite any pathological cases.

Shewchuk [55] gave pseudocode for a "hill-climbing" algorithm designed to improve meshes with topological changes in a manner similar to Joe's algorithm. His approach to removing edges connected to more than four elements employs Klincsek's [56] methodology to construct the optimal triangulation of the ring of edges around the edge to be removed. Since Klincsek's algorithm assumes the points are in the plane, it is not clear how successful this technique will be for non-planar point sets.

1.6 Vertex Smoothing

In practical mesh generation, there is rarely need to constrain vertices except those that define the geometry. As such, it seems a good and natural pursuit to move vertices to improve mesh quality. A simple illustration of the process in two-dimensions is given in Figure 1.9. Note how the sliver triangle in the lower right has been eliminated by the point movement.

Of course, exactly how to achieve a desirable point position is not obvious. There has, however, been a good deal of research in the area of mesh vertex smoothing. Generally, there are two distinct vertex optimization schemes employed for mesh quality improvement: Laplacian smoothing and optimization-based smoothing.



(a) before smoothing (b) after smoothing

Figure 1.9 An Example of Node Perturbation

Laplacian smoothing is based upon taking the average coordinate of the set of vertices connected to a given vertex and moving that vertex to the averaged coordinate. This technique is cast as a Laplacian equation

$$\sum_i w_i \left(\mathbf{x}_i - \mathbf{x}\right) = 0 \tag{1.2}$$

where $\mathbf{x}$ is the position of the vertex in question, the $\mathbf{x}_i$ are the positions of the vertices surrounding $\mathbf{x}$, and the w_i are some prescribed weights. An early reference to Laplace smoothing can be found in Buell [57]. Modern work on this technique seems to have started with Field [58] who shows

that maintaining the Delaunay property after point movement greatly improves the quality in a triangular mesh.

Optimization-based smoothing techniques rely upon minimizing (or maximizing) some function, $\Phi(\mathbf{x})$, which describes some quality measure of the elements surrounding the vertex at $\mathbf{x}$. Also the gradient of the quality function, $\nabla\Phi(\mathbf{x})$, is usually (but not always) employed in finding the minimum or maximum.

As there has been a great deal of work in mesh smoothing over the years with generally good results, it seems necessary to list as much of that work as possible. What follows is as exhaustive of a set of sources as the author could compile, in rough order of increasing interest to the work presented in this dissertation. Indeed, there has been earlier work which attempted to expound upon and categorize the notion of mesh improvement. Ollivier-Gooch [59] gave an overview of several mesh improvement techniques, including mesh smoothing and topological transformations, designed to improve unstructured grids created from randomly placed points. Dompierre et al. [60] proposed a set of benchmarks for tetrahedral optimization based upon several test cases including a unit equilateral tetrahedra. While no attempt is made here to synthesize the research and, indeed, little of what follows was explicitly used in this research, all of the following works are worthy of some consideration.

Parthasarathy and Kodiyalam [61] gave an optimization method based upon minimizing an aspect ratio based global objective function. Canann et al. [62] introduced, in the amusingly titled Optimsmoothing, the idea of minimizing the sum of some quality metric via a conjugate gradient method. Later, Canann et al. [63] presented a smoothing scheme, implemented in ANSYS®, that perturbs one node at at time with both Laplacian smoothing and optimization based scheme.

Amezua et al. [64] proposed a method based on optimizing desired local edge lengths with a quasi-Newton method cast as a finite-element system. Amenta et al. [65] gave a method based upon linear programming for the optimum placement of a newly inserted point. Zhou and Shimada [66] showed an angle-based smoothing method for triangular and quadrilateral meshes based upon a spring analogy. Xu and Newman [67] used a very similar technique but relocated the node with an optimization technique instead of a heuristic technique like Zhou and Shimada. These techniques give demonstrably better results than Laplacian smoothing but an obvious extension to three-dimensions is not apparent.

Consideration of the effect of mesh smoothing on finite element schemes is not highly represented in the literature, but there is some work to be found. Z. Chen et al. [68] proposed an objective function designed for two-dimensional triangular and quadrilateral grids intended for finite-element simulations. The function consists of using the L2 norm and introducing an "angle penalty term" which gave higher quality quadratic elements. L. Chen [69] has introduced a two-dimensional smoothing scheme shown to reduce interpolation error. Since much of the trend in simulation is toward finite element methods, we should expect to see more work of this type.

Occasionally, as is employed in this research, topological changes are applied in addition to node smoothing in order to improve mesh quality. In [70], De L'Isle and George gave an extensive procedure for the optimization of a tetrahedral mesh. Their algorithm consists of removing edges and faces (i.e. flipping,) relocating points one at a time based upon an edge length/in-radius based quality metric and, uniquely, adding and removing points to "suppress" certain edges. In their own words, the algorithm is "lazy" in the sense that it tests each change to see if mesh quality is improved before applying said change. Also, the various optimization techniques are applied in an

arbitrary order. However, the scheme is very successful at improving mesh quality. Further, results are shown wherein the quality metric is modified on a per element basis to obtain certain element shapes and point distributions in different areas of the mesh.

Freitag et al. [71–76] cast the optimization problem in terms of maximizing the minimum angle of the triangles or tetrahedra surrounding a vertex. This optimization was performed via a set of search directions and employed analytical functions and gradient definitions. Further, these works include examples involving both Laplacian and optimization-based smoothing (see [74]) of the same meshes as well as incorporating edge and face flipping. In fact, they note in [73] that it is quite rare for the removal of an edge surrounded by more than seven tetrahedra to yield an improvement in quality. Some of this work has been shown to even be able to "untangle" invalid meshes (see [76].) More importantly, they showed experimentally in [75] that smoothing and flipping yielded considerable savings in solution time.

Klinger [77,78] likewise combined optimization-based smoothing with topological changes. His smoothing algorithm is taken from Freitag (see above) and, like that implementation, works on one node at a time and only accepts the new position of a vertex if the mesh is improved. Further, Klinger's work has implemented the ability to smooth nodes on the boundaries, although no boundary flips are performed. He has also implemented Shewchuk's multi-face removal scheme. Finally, Klinger uses clever vertex insertions as a way of improving the mesh. Klinger's tetrahedral mesh improvement code, Stellar[2], code is available online.

[2]http://www.cs.berkeley.edu/~jrs/stellar/

Freitag and Klinger's work is very close to that presented in this dissertation but differs in a few important ways. First, these works employ optimization methods which work on only one node at a time. Second, they optimize an angle-based function instead of a general quality metric. Finally, none of these techniques are used during the initial mesh generation.

1.7 Overview of Mesh Improvement and Creation Strategies Explored

Most of the author's introduction to the mesh smoothing strategy employed in this work came through Steve Karman, who originally employed an optimization scheme to smooth meshes constructed from an octree. Most of this work was originally presented in [79] and [80]. A continuation of the work that also employed solution adapted smoothing techniques was presented in [81]. Much of this dissertation is a continuation of the research presented there.

This work will use topological mesh transformations including removing edges surrounded by more than four tetrahedra as well as gradient based vertex smoothing to improve mesh quality. Quality will be defined primarily with the metric of Weighted Condition Number although tetrahedral aspect ratio will also be reported. (See Section 2.1.) Detailed explanations of both the topological improvement and node smoothing algorithms are given. (See Sections 2.2 and2.3.) An explanation of the node insertion, boundary recovery, and node creation techniques used to create volume meshes (as opposed to improving existing ones) are overviewed. (See Section 2.5.) Finally, a number of examples of mesh improvement and creation are shown. (See Chapter 3.)

CHAPTER 2

METHODOLOGIES AND ALGORITHMS

This chapter gives an explication of the ideas and procedures relevant to this work. In particular, a quality metric, weighted condition number, used to evaluate a tetrahedron's utility for computational simulation is thoroughly explicated. Also, the several algorithms, most importantly those of topological optimization and node smoothing, are given in detail.

One of the important ideas, both in theory and implementation, that all of the approaches presented herein take is simply that improving the worst tetrahedron of any given configuration takes precedence over the quality of any other tetrahedron involved. While this notion is not unique, it is not necessarily a motivating factor in other tetrahedral creation and optimization algorithms. This motivation for improving the worst tetrahedra is simple: it only takes one bad element to ruin a simulation. Being that the goal of this work is to develop schemes to create and improve meshes intended for simulation, as opposed to some other academic end, it is natural to focus on improving the worst tetrahedron; even if this focus sacrifices the quality of other more well-shaped elements. Practical grid generation experience shows this strategy to be fruitful. Indeed, the author has encountered many grids where just a handful of poor elements prevented large simulations from running successfully.

In that light, both the topological optimization and node perturbation optimization techniques given in this work will focus on improving the worst tetrahedron. These emphases will

be pointed out in their explanations. Even the choice of a quality measure was influenced by this principle.

2.1 A Metric for Tetrahedra Evaluation

In order to evaluate the quality of a tetrahedron and whether it is a candidate for vertex optimization or topological optimization, one must, of course, have some numerical measure generally related to that tetrahedron's suitability for use in simulation. Generally, simulation software prefers elements with shapes whose angles are not far from right, i.e. angles that are too small or too large are undesirable. Precisely what constitutes "too small" and "too large" is dependent upon a variety of factors including the order of the simulation, the nature of the physics, and the particulars of solver implementation. In any case, with a few exceptions to capture specific physics such as boundary layers and shocks, it is usually desirable to have tetrahedra that are as close to equilateral as possible for the simple reason that an equilateral tetrahedron "regularizes" all the angles. That is, a decrease in any angle of the tetrahedra (face-to-face angles, edge-to-edge angles, face-to-edge angles, etc.) will necessarily subtend an increase to some other angle.

2.1.1 Weighted Condition Number

This work has focused on a tetrahedral quality metric that enforces good angles without having to actually measure the angles directly: weighted condition number. Freitag and Knupp gave an overview of this metric in [82] and the explanations and proof presented below are taken from their paper. In short, the weighted condition number of a tetrahedron is a measure of that tetrahedron's "equilateral-ness" or equiangularity.

The (non-weighted) condition number of a tetrahedron is directly related to the condition number of a matrix. Label the vertices of the tetrahedron $\mathbf{x}_n$ for $n = 0, 1, 2, 3$. If we consider one vertex $k \in \{0, 1, 2, 3\}$ and construct the vectors emanating from that vertex, $\mathbf{e}_{k+1,k}, \mathbf{e}_{k+2,k}, \mathbf{e}_{k+3,k}$ as columns of a matrix, we have

$$A_k = \begin{bmatrix} \mathbf{e}_{k+1,k,1} & \mathbf{e}_{k+2,k,1} & \mathbf{e}_{k+3,k,1} \\ \mathbf{e}_{k+1,k,2} & \mathbf{e}_{k+2,k,2} & \mathbf{e}_{k+3,k,2} \\ \mathbf{e}_{k+1,k,3} & \mathbf{e}_{k+2,k,3} & \mathbf{e}_{k+3,k,3} \end{bmatrix} \tag{2.1}$$

To be clear, $\mathbf{e}_{n,m,p}$ indicates the pth component of the vector from the vertex m to the vertex n. Also, the first index, indicating the terminating node of the vector, is taken modulo 4. This construction is also known as the Jacobian matrix of the tetrahedra from the given corner.

Note that if we define

$$M = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{2.2}$$

then, by construction, $A_k = A_0 M^k$, where the superscript on M indicates exponentiation. For example,

$$\begin{aligned}
A_0 M^2 &= \begin{bmatrix} \mathbf{e}_{1,0,1} & \mathbf{e}_{2,0,1} & \mathbf{e}_{3,0,1} \\ \mathbf{e}_{1,0,2} & \mathbf{e}_{2,0,2} & \mathbf{e}_{3,0,2} \\ \mathbf{e}_{1,0,3} & \mathbf{e}_{2,0,3} & \mathbf{e}_{3,0,3} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}^2 \\
&= \begin{bmatrix} \mathbf{e}_{1,0,1} & \mathbf{e}_{2,0,1} & \mathbf{e}_{3,0,1} \\ \mathbf{e}_{1,0,2} & \mathbf{e}_{2,0,2} & \mathbf{e}_{3,0,2} \\ \mathbf{e}_{1,0,3} & \mathbf{e}_{2,0,3} & \mathbf{e}_{3,0,3} \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ -1 & -1 & -1 \\ 1 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} -\mathbf{e}_{2,0,1} + \mathbf{e}_{3,0,1} & -\mathbf{e}_{2,0,1} & \mathbf{e}_{1,0,1} - \mathbf{e}_{2,0,1} \\ -\mathbf{e}_{2,0,2} + \mathbf{e}_{3,0,2} & -\mathbf{e}_{2,0,3} & \mathbf{e}_{1,0,2} - \mathbf{e}_{2,0,2} \\ -\mathbf{e}_{2,0,3} + \mathbf{e}_{3,0,3} & -\mathbf{e}_{2,0,3} & \mathbf{e}_{1,0,3} - \mathbf{e}_{2,0,3} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{e}_{3,2,1} & \mathbf{e}_{0,2,1} & \mathbf{e}_{1,2,1} \\ \mathbf{e}_{3,2,2} & \mathbf{e}_{0,2,2} & \mathbf{e}_{1,2,2} \\ \mathbf{e}_{3,2,3} & \mathbf{e}_{0,2,3} & \mathbf{e}_{1,2,3} \end{bmatrix}
\end{aligned} \tag{2.3}$$

since $-\mathbf{e}_{2,0,1} + \mathbf{e}_{3,0,1} = \mathbf{e}_{3,2,1}$, $-\mathbf{e}_{2,0} = \mathbf{e}_{0,2}$, $\mathbf{e}_{1,0,3} - \mathbf{e}_{2,0,3} = \mathbf{e}_{1,2}$ by vector arithmetic. Further, the Jacobian of a tetrahedron T is defined as

$$J(T) = \det(A_k) \tag{2.4}$$

Note that the choice of vertex k does not affect $J(T)$ since

$$\begin{aligned} \det(A_k) &= \det\left(A_0 M^k\right) \\ &= \det(A_0)\left(\det(M)\right)^k \\ &= \det(A_0) \cdot 1 \\ &= \det(A_0) \end{aligned} \tag{2.5}$$

Note that $J(T) < 0$ for inverted tetrahedra and $J(T) = 0$ for a tetrahedron defined by four coplanar points (i.e. a "flat" tetrahedron.) The Jacobian will be used below in constructing a tetrahedral cost function.

Now, consider a "unit" tetrahedra with one vertex at the origin and each side of unit length along the cardinal axes (i.e. a tetrahedron with vertices $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$.) Then the matrix constructed from the origin is

$$A_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

The condition number of this matrix in the Frobenius norm is

$$\kappa(A_0) = \|A_0\|_F \|A_0^{-1}\|_F = 3 \tag{2.7}$$

since $\|A_0\|_F = \sqrt{1^2 + 0^2 + \ldots 1^2} = \sqrt{3}$ and $A_0^{-1} = A_0$. Note this is also the condition number in the Euclidean norm. Since we are considering a unit tetrahedra and its right angle vertex, it is natural to normalize this metric to one for this case. Thus, condition number of a tetrahedron T taken from its vertex k is defined as

$$\mathrm{CN}_k(T) = \frac{\|A_k\|_F \|A_k^{-1}\|_F}{3} \tag{2.8}$$

for $k = 0, 1, 2, 3$. Note that the higher the condition number, the closer that corner of the tetrahedron is to planer and, thus, the smaller (or larger) the solid angle of that corner.

CN_k has several nice properties for a tetrahedral quality metric. First, since the Frobenius norm is invariant to rotation matrices, any rotation of the tetrahedron will not effect its condition numbers. Also, as will be shown later, CN_k is actually algebraic. That is, it can be defined using only addition, subtraction, multiplication, division, and raising to a rational power. Finally, and most importantly, CN_k is scale invariant. That is, the size of the tetrahedra does not effect the condition number. This property is easily shown. Consider a tetrahedron T whose vertices have all been multiplied by α to create T'. Then, if T's original Jacobian matrices were A_k, T''s Jacobian

matrices are simply αA_k. So

$$\begin{aligned} \mathrm{CN}_k\,(T') &= \frac{\|\alpha A_k\|_F \|(\alpha A_k)^{-1}\|_F}{3} \\ &= \frac{\|\alpha A_k\|_F \|\frac{1}{\alpha} A_k^{-1}\|_F}{3} \\ &= \frac{\alpha \|A_k\|_F \frac{1}{\alpha} \|A_k^{-1}\|_F}{3} \\ &= \frac{\|A_k\|_F \|A_k^{-1}\|_F}{3} \\ &= \mathrm{CN}_k\,(T) \end{aligned} \tag{2.9}$$

by the straight-forward properties of the matrix inverse and matrix norms.

Despite invariance to rotation and scale, by itself condition number is an unsatisfactory metric. First it is smallest for a right angle tetrahedron, not an equilateral one. Second, and more importantly, condition number is not independent of one's choice of vertices. (Simply choose a different vertex for the right angle tetrahedron referenced above and the calculation easily reveals this dependence.) Both of these undesirable aspects can be corrected with a simple transformation.

Consider the equilateral tetrahedron with vertices at $(0, 0, 0)$, $(1, 0, 0)$, $\left(1/2, \sqrt{3}/2, 0\right)$, and $\left(1/2, \sqrt{3}/6, \sqrt{2}/\sqrt{3}\right)$. The Jacobian matrix taken from the origin is

$$W = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{bmatrix} \tag{2.10}$$

and its inverse is

$$W^{-1} = \begin{bmatrix} 1 & -\frac{\sqrt{3}}{3} & -\frac{\sqrt{2}}{2\sqrt{3}} \\ 0 & \frac{2\sqrt{3}}{3} & -\frac{\sqrt{2}}{2\sqrt{3}} \\ 0 & 0 & \frac{3\sqrt{2}}{2\sqrt{3}} \end{bmatrix} \tag{2.11}$$

These matrices will serve as linear transformations of the tetrahedral Jacobian matrices. Define the Weighted Condition Number of a tetrahedron from the vertex k as

$$\mathrm{WCN}_k\,(T) = \frac{\|A_k W^{-1}\|_F \|W A_k^{-1}\|_F}{3} \tag{2.12}$$

Now, we can overcome the deficiencies of the unweighted tetrahedral condition by showing that the weighted condition number of a tetrahedra does not depend on one's choice of k. The proof is a follows:

Define the matrix $R = WMW^{-1}$ for the W, M given above. Then

$$\begin{aligned} R &= WMW^{-1} \\ &= \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -\frac{\sqrt{3}}{3} & -\frac{\sqrt{2}}{2\sqrt{3}} \\ 0 & \frac{2\sqrt{3}}{3} & -\frac{\sqrt{2}}{2\sqrt{3}} \\ 0 & 0 & \frac{3\sqrt{2}}{2\sqrt{3}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{6} & \frac{\sqrt{2}}{\sqrt{3}} \\ \frac{\sqrt{3}}{2} & \frac{1}{6} & \frac{\sqrt{2}}{3} \\ 0 & -\frac{2\sqrt{2}}{3} & \frac{1}{3} \end{bmatrix} \end{aligned} \tag{2.13}$$

Direct calculation shows that $R^{\mathrm{T}}R = I$ and $\det(R) = 1$, so, R is a rotation matrix. Further, if R is a rotation matrix, so is R^{-1}. Note also that $R^n = \left(WMW^{-1}\right)^n = WM^nW^{-1}$ for every integer n. Since, as stated above, the Frobenius norm is invariant under rotations, we have

$$\begin{aligned} \|A_kW^{-1}\|_F &= \|A_0M^kW^{-1}\|_F \\ &= \|A_0W^{-1}R^k\|_F \\ &= \|A_0W^{-1}\|_F \end{aligned} \tag{2.14}$$

Similarly

$$\begin{aligned} \|WA_k^{-1}\|_F &= \left\|\left(A_kW^{-1}\right)\right\|_F \\ &= \left\|\left(A_0M^kW^{-1}\right)^{-1}\right\|_F \\ &= \left\|\left(A_0M^kW^{-1}\right)^{-1}\right\|_F \\ &= \left\|\left(A_0W^{-1}R^k\right)^{-1}\right\|_F \\ &= \left\|\left(R^k\right)^{-1}WA_0\right\|_F \\ &= \|WA_0\|_F \end{aligned} \tag{2.15}$$

Therefore

$$\frac{\|A_kW^{-1}\|_F\|WA_k^{-1}\|_F}{3} = \frac{\|A_0W^{-1}\|_F\|WA_0^{-1}\|_F}{3} \tag{2.16}$$

and, so, $\mathrm{WCN}_k = \mathrm{WCN}_0$ for all $k = 0, 1, 2, 3$ and the proof is complete.

As such, it is perfectly reasonable to drop the weighted condition number's dependence on the choice of vertex and simply say

$$\mathrm{WCN}\,(T) = \frac{\|AW^{-1}\|_F \|WA^{-1}\|_F}{3} \tag{2.17}$$

Note that WCN has a range of $[1, \infty)$ with the convenient property that an equilateral tetrahedron has a WCN of one. Also, the scale invariance and rotational invariance shown for CN_k will also hold for WCN.

2.1.2 Tetrahedral Cost

One important drawback still remains in the use of weighted condition number as a tetrahedral cost metric. Namely, WCN will not identify an inverted tetrahedron. To overcome this deficiency, a cost function has been developed that first calculates the Jacobian of the tetrahedron. With this definition in mind, the cost of a tetrahedra, T, based on WCN is defined as

$$c_W\,(T) = \begin{cases} 1 - J\,(T) & \text{if } J\,(T) \leq 0 \\ 1 - \frac{1}{\mathrm{WCN}(T)} & \text{if } J\,(T) > 0 \end{cases} \tag{2.18}$$

Note $c_w\,(T) \in [0, \infty)$ with $c_W\,(T) = 0$ for an equilateral tetrahedron, $c_W\,(T) = 1$ for a flat tetrahedron, and $c_W\,(T) > 1$ for an inverted or invalid tetrahedron.

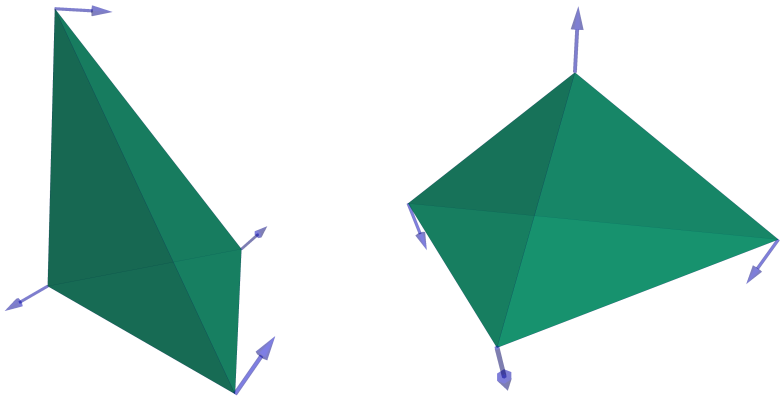
For smoothing purposes, it will also be necessary to calculate the derivatives of $c_W\,(T)$. The gradient matrix of this cost for any tetrahedra will be of size 4×3 with one entry for each derivative with respect to each coordinate of each vertex. While it would be possible to find an

analytic definition of this gradient, there are a number of obstacles involved in such a derivation. First and foremost, such a derivation would be quite tedious as the expansion of WCN itself is daunting to say the least. Also, the derivatives of c_W are not continuous at $c_W = 1$. That is, c_W is a C^0 function. However, these drawbacks are not that important. First, there is no interest in tetrahedra with $c_W(T) \geq 1$ except to move their nodes to a position of validity and, thus, the smoothness of the function around $c_W(T) = 1$ is of no concern. That is, the Jacobian is simply part of the definition to identify when a tetrahedron is inverted. Secondly, there are well understood numerical techniques to calculate the derivative of a function while calculating its value, namely numerical complex differentiation and dual numbers. For this work, dual numbers were chosen as the mechanism to calculate the derivative of c_W while calculating its value. See Appendix A for an explanation of dual numbers.

2.1.3 Examples of Tetrahedral Weighted Condition Number and Cost

In practical mesh generation, it is important to have a sense of the quality metrics used to evaluate a mesh. Further, an understanding of the gradients used in this research is important to understanding the validity and utility of the method. As a way of developing some intuition as to the nature of WCN and c_W, consider the following examples.

First, a right angle tetrahedron with vertices $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ has a weighted condition number of WCN $= 1.22474487$ and cost of $c_W = 0.18350342$. Figure 2.1 shows such a tetrahedron. The vectors depicted are the normalized, inverted gradients of c_W associated with each vertex and, thus, each vector points in the direction of greatest decrease for the cost of the tetrahedra. In simpler terms, moving the node along that vector will, at least for small



(a) origin on the left (b) origin on top

Figure 2.1 A Right Angle Tetrahedron with Normalized, Inverted Gradient vectors of c_W.

perturbations, improve the quality of the tetrahedra. It is readily apparent that these vectors are pointing, at least initially, in the directions that will move the tetrahedron towards equiangularity.

For more examples, consider a perfectly equilateral tetrahedron with vertices $(0, 0, 0)$, $(1, 0, 0)$, $\left(1/2, \sqrt{3}/2, 0\right)$, and $\left(1/2, \sqrt{3}/6, \sqrt{2}/\sqrt{3}\right)$. Of course, such a tetrahedron has a weighted condition number of WCN $= 1.0$ and cost of $c_W = 0.0$. Figure 2.2 shows an equilateral tetrahedron and Figure 2.3 shows several tetrahedra each with an equilateral triangle for a base with its apex having been moved to various positions. Just as in Figure 2.1, these tetrahedra are depicted with the normalized, inverted gradients of c_W. Figure 2.3a shows a tetrahedra whose vertex has been moved over the origin. Note, as was the case for the right angle tetrahedron, the vectors are pointing in directions that will move the tetrahedron toward equiangularity. However, it is interesting to see that the apex vector is not simply pointing directly back towards what would make it the apex of an equilateral tetrahedron but actually indicates a change in the z-coordinate of the apex. This observation shows how the derivatives do not always give a direct path towards equiangularity.

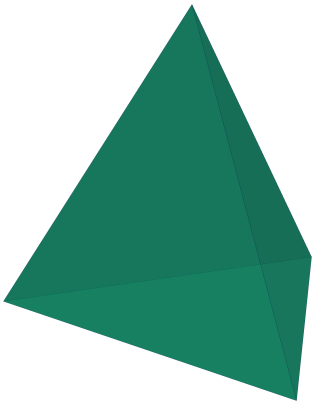
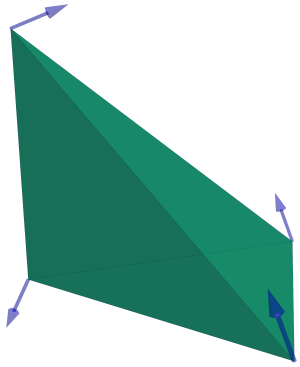


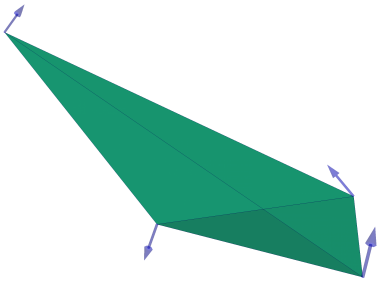
Figure 2.2 An Equilateral Tetrahedron

Figures 2.3c, 2.3d, and 2.3f show equilateral tetrahedra whose apices have been moved up and down the z-axis by factors of 2. For the tetrahedron whose apex has been moved down, the vectors indicate that it should be moved directly upwards to improve c_W. Also, note the vectors on the base have a significant negative z component. As one would expect, the two tetrahedra whose apex has been shifted upwards exhibit exactly the opposite behavior with respect to their cost improvement vectors. More importantly, note that the tetrahedra in Figures 2.3b, 2.3d, and 2.3f are deformed to the extent that visual inspection engenders suspicion of low quality. Analytically, the face angles between the base and sides in Figure 2.3d are only 35.26° and the vertex angles around the apex of the tetrahedron in Figure 2.3f are only 17.34°. Experience shows that small angles lead to poor performance in some analysis software. Moreover, the angle between the base and vertex in Figure 2.3b is 140.8°. As referenced earlier, Shewchuk [2] has shown that large angles lead to large error in finite element analysis.

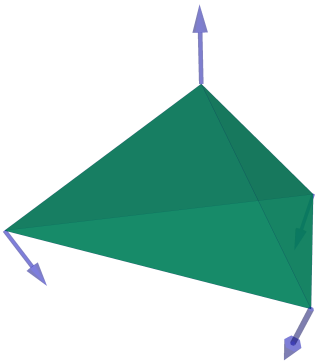
For a more analytical example, consider the tetrahedron with an equiangular base and some third node above that base. That is, a tetrahedron with vertices $(0, 0, 0)$, $(1, 0, 0)$, $\left(1/2, \sqrt{3}/2, 0\right)$,



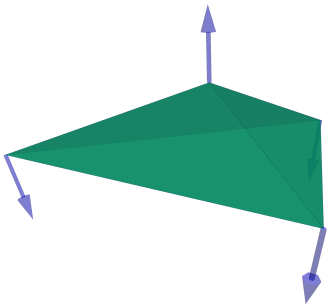
(a) tetrahedron with equilateral base and apex of $\left(0, 0, \sqrt{2}/\sqrt{3}\right)$;WCN = 1.167, $c_W = 0.1429$



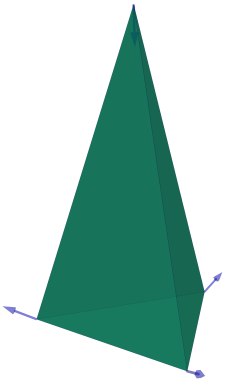
(b) tetrahedron with equilateral base and apex of $\left(-1, 0, \sqrt{2}/\sqrt{3}\right)$;WCN = 2.167, $c_W = 0.5385$



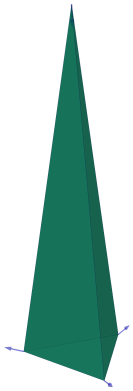
(c) tetrahedron with equilateral base and apex $\left(1/2, \sqrt{3}/6, \sqrt{2}/2\sqrt{3}\right)$; WCN = 1.225, $c_W = 0.1835$



(d) tetrahedron with equilateral base and apex $\left(1/2, \sqrt{3}/6, \sqrt{2}/4\sqrt{3}\right)$; WCN = 2.031, $c_W = 0.5076$



(e) tetrahedron with equilateral base and apex $\left(1/2, \sqrt{3}/6, 2\sqrt{2}/\sqrt{3}\right)$; WCN = 1.225, $c_W = 0.1835$



(f) tetrahedron with equilateral base and apex $\left(1/2, \sqrt{3}/6, 4\sqrt{2}/\sqrt{3}\right)$; WCN = 2.031, $c_W = 0.5076$

Figure 2.3 Various Tetrahedra Related to an Equilateral Tetrahedron

and (x_3, y_3, z_3) where $x_3, y_3, z_3 > 0$ to assure the tetrahedron is valid. So, the A matrix for this tetrahedron is

$$A = \begin{bmatrix} 1 & \frac{1}{2} & x_3 \\ 0 & \frac{\sqrt{3}}{2} & y_3 \\ 0 & 0 & z_3 \end{bmatrix} \tag{2.19}$$

which gives (with the help of the computer algebra system Maple[1] the tetrahedron's WCN as

$$\begin{aligned} \mathrm{WCN}(T) &= \frac{1}{18 z_3} \cdot \sqrt{10 - 6x_3 + 6x_3^2 - 2y_3\sqrt{3} + 6y_3^2 + 6z_3^2} \\ &\quad \cdot \sqrt{-3y_3\sqrt{3} + 9x_3^2 + 9y_3^2 + 18z_3^2 - 9x_3 + 9} \end{aligned} \tag{2.20}$$

Note that even for this case with a single free vertex, the expansion of the WCN is rather cumbersome. Its derivatives would be even more so. Thus the use of numerical differentiation over some exact formulation is justified at least in terms of convenience if not speed of computation. Nonetheless, the expansion is straight-forward and entirely algebraic.

This WCN and its associated c_W are plotted in Figure 2.4. Figures 2.4a and 2.4b graph WCN and c_W for fixed x_3, y_3 and free z_3. Figures 2.4c and 2.4d graph WCN and c_W for fixed z_3 and free x_3, y_3. Note, it is clear is all these plots that the quality of the tetrahedron is best (i.e. c_W and WCN are lowest) when the $(x_3, y_3, z_3) = \left(1/2, \sqrt{3}/6, \sqrt{2}/\sqrt{3}\right)$ and the tetrahedron is perfectly equilateral. More importantly, it is clear that these graphs have that point as a single minimum. While not a picture of the entire space, this simple analysis gives credence to the practice of using a gradient based method for optimization of c_W.

[1] http://www.maplesoft.com/products/maple/)

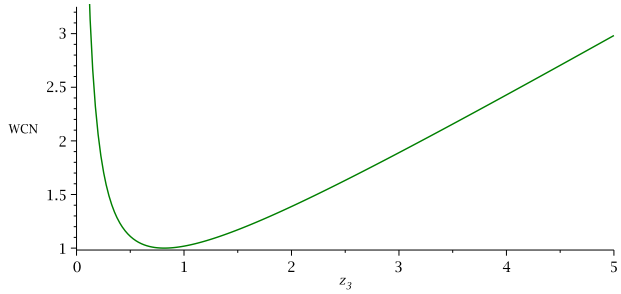
Interestingly, Figures 2.4a and 2.4b also clearly show that the derivatives of $c_W(T)$ are not monotonic. There are clear inflection points and it is obvious the derivative is decreasing as the vertex moves away from the plane of the tetrahedron's base. This observation is borne out in practice and is one of the reasons the gradient vector is primarily used to provide a perturbation direction and not a magnitude. (See Section 2.3.)

As a final investigation of the space of WCN and c_W, consider some arbitrary but valid tetrahedron T. Translate one vertex of T to the origin, rotate one edge onto the x-axis, and scale the tetrahedron such that this edge is of unit length to create a tetrahedron T' with vertices $(0, 0, 0)$, $(1, 0, 0)$, (x_2, y_2, z_2), and (x_3, y_3, z_3) . Since WCN is scale invariant, rotationally invariant, and the translation of T will not effect the vectors that define its edges, then the WCN of T and T' will be equal. With, yet again, a great deal of help from a computer algebra system, one can show that the expansion of the $T's$ WCN is

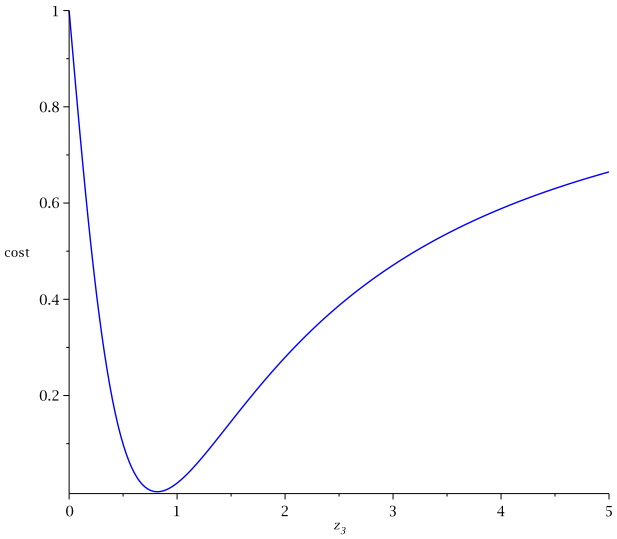
$$\mathrm{WCN}(T) = \frac{\sqrt{\rho_1 \, \rho_2}}{6 \, (y_2 \, z_3 - y_3 \, z_2)} \tag{2.21}$$

where

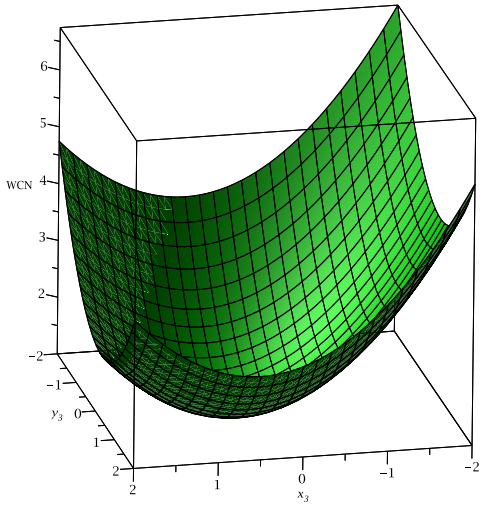
$$\begin{aligned} \rho_1 \;=\; & 6x_2^2 - 4\,x_2\,x_3 + 6\,x_3^2 + 6\,y_2^2 - 4\,y_2\,y_3 + 6\,y_3^2 \\ & +6\,z_2^2 - 4\,z_2\,z_3 + 6\,z_3^2 - 4\,x_2 - 4\,x_3 + 6 \end{aligned} \tag{2.22}$$



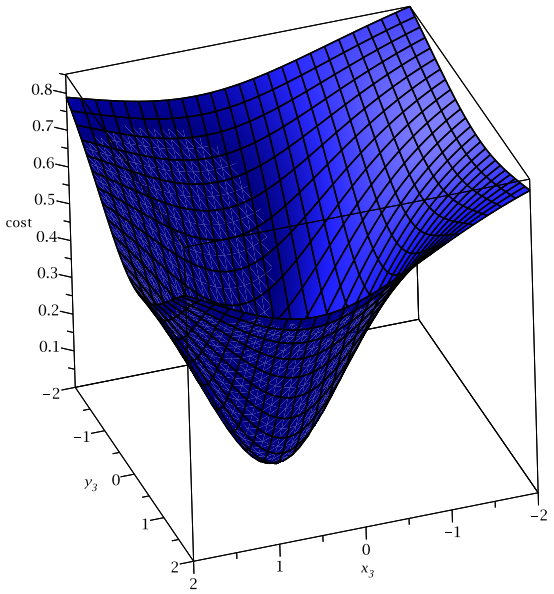
(a) WCN vs. z_3



(b) c_W vs. z_3



(c) WCN vs. x_3, y_3



(d) c_W vs. x_3, y_3

Figure 2.4 Plots of WCN and Cost Verses Position of Apex of Tetrahedron with Equilateral Base

$$\begin{aligned}\rho_2 \;=\;& x_2^2 y_3^2 + x_2^2 z_3^2 - 2\, x_2\, x_3\, y_2\, y_3 - 2\, x_2\, x_3\, z_2\, z_3 + x_3^2 y_2^2 + x_3^2 z_2^2 + y_2^2 z_3^2 - 2\, y_2\, y_3\, z_2\, z_3 \\ & + y_3^2 z_2^2 + x_2\, y_2\, y_3 - x_2\, y_3^2 + x_2\, z_2\, z_3 - x_2\, z_3^2 - x_3\, y_2^2 + x_3\, y_2\, y_3 - x_3\, z_2^2 \\ & + x_3\, z_2\, z_3 + y_2^2 - y_2\, y_3 + y_3^2 + z_2^2 - z_2\, z_3 + z_3^2 \end{aligned} \tag{2.23}$$

Note again, this expansion is cumbersome. More importantly this formulation is, indeed, general and could be used to calculate the WCN of any tetrahedron after translation, rotation, and scaling. So, definitively, one can say that WCN is algebraic and, thus, everywhere differentiable and, moreover, its derivative is everywhere continuous. So, for valid tetrahedra, c_W is a C^1 function.

All of this analysis leads to the practical question of what threshold of c_W is unacceptable for simulation. While a definitive answer will be dependent upon implementation, a pattern does emerge. The three poorer quality tetrahedra in Figures 2.3b, 2.3d, and 2.3f all have WCN ≥ 2.0. In Figures 2.4b and 2.4d, it is easy to see that input values giving $c_W \geq 0.5$ are well past the inflection point and, thus, the gradient is becoming less useful for optimization. These observations and experience with the algorithms have led to taking any tetrahedron with $c_W \leq 0.5$, WCN ≤ 2.0 as acceptable.

2.2 Topological Optimization

Topological optimization of a tetrahedral mesh can be a very powerful mechanism to improve mesh quality locally. A particularly poor quality tetrahedra can be eliminated if some viable topological transformation is performed. Indeed, since there are ten potential flips associated with each tetrahedra, four face flips and six edge flips, the utility of such transformations is easy to grasp.

2.2.1 Higher Order Flips

The three basic flips, 3-2, 2-3, and 4-4, were presented in Section 1.5. However, it is quite possible for there to be more than four tetrahedra arrayed around an edge. In fact, there is no upper bound for the number of elements that can contain any given edge. In this work, the removal of an edge surrounded by more than four tetrahedra is refereed to as a "higher order" flip.

The central problem of removing an edge surrounded by some number of tetrahedra is triangulating the set of points not part of the edge E to be removed. Here we refer to this set of points as the "ring" surrounding E. For a 3-2 flip, the problem is trivial as there are only three points and, thus, only the single triangulation of those points. For a 4-4 flip, there are four points in the ring; so there are two possible triangulations from which to choose. For five or or more, the number of nodes in the rings increases the potential number of triangulations factorially. In fact the number of valid triangulations for a convex polygon with n vertices, as given in [83], is

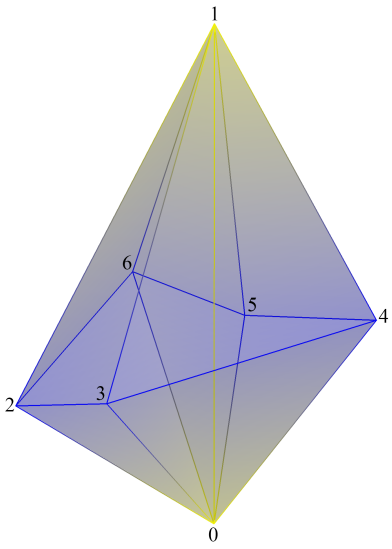
$$\frac{n\,(n+1)\ldots(2n-4)}{(n-2)!} \tag{2.24}$$

Checking every triangulation is obviously impractical. Some solutions to this problem are referenced in Section 1.5. The solution presented here, however, is based upon recasting the triangulation of the ring as another topological optimization problem. The edge E is removed and a triangulation of the ring is constructed using an "ear-clipping" method similar to that described by El Gindy et al. [84]. Each node n in the ring along with its neighbors to the left and right, $n-1$, $n+1$, are used to form a triangle. These triangles are then used to create two tetrahedra, each connected to a different node of E. If these tetrahedra are valid, they are retained and n is removed

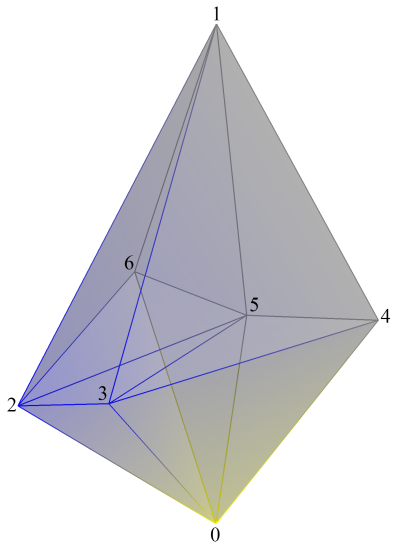
from the ring. If the two tetrahedra are invalid, then move onto the next node in the ring. The procedure is repeated until the ring is completely collapsed. Keep in mind the ring of nodes is by no means necessarily planar; but the validity of any triangulation is judged on the validity of the tetrahedra it subtends, a measure that accounts for its non-planarity.

This process creates a new set of tetrahedra Θ which, as it is guaranteed to be valid, may potentially replace the original set of tetrahedra around E. Θ is then optimized in isolation using only 4-4 flips. Section 2.2.3 details the general topological optimization algorithm. For this case, the optimization is exactly the same as using Lawson's two-dimensional algorithm on the arbitrary triangulation; but, instead of using a circumcircle criterion for edge swapping, edges are swapped based on the tetrahedral cost of the four tetrahedra around the edge.

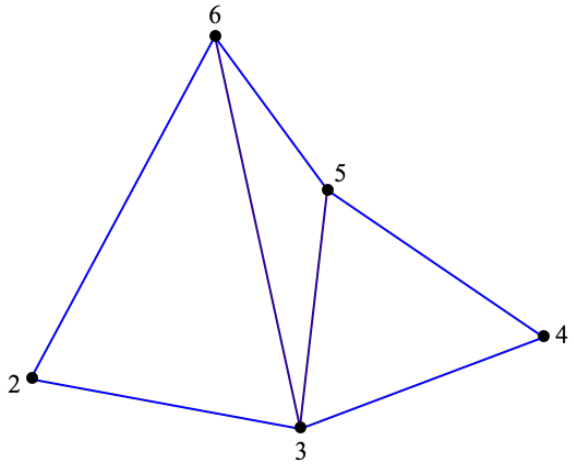
Figure 2.5a and 2.5b show a higher order flip involving five tetrahedra transformed into six tetrahedra. Figures 2.5c and 2.5d show the ring triangulation before and after optimization.



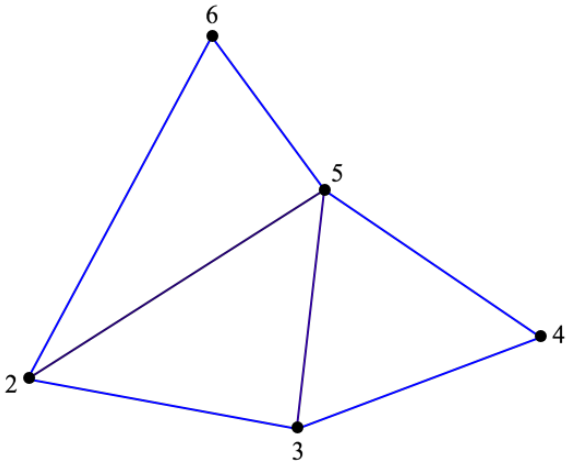
(a) before: 0 – 1 – 2 – 3, 0 – 1 – 3 – 4, 0 – 1 – 4 – 5, 0 – 1 – 5 – 6, and 0 – 1 – 6 – 2



(b) after: 2 – 3 – 5 – 1, 5 – 3 – 2 – 1, 3 – 4 – 5 – 1, 5 – 4 – 3 – 1, 5 – 6 – 2 – 1, and 2 – 6 – 5 – 1



(c) original arbitrary ring triangulation



(d) optimized ring triangulation

Figure 2.5 A Higher Order Flip and Its Ring

2.2.2 Boundary Flips

Although the problem this work attempts to solve expressively requires the boundary triangulation to be retained as part of the final mesh, it is possible to perform topological improvements with tetrahedra connected to the boundary. Consider two adjacent triangles on the boundary and the set of tetrahedra Θ connected to the edge E those triangles share. It is possible to swap that edge amongst the two triangles and re-tetrahedralize Θ. If there are only two tetrahedra connected to E then this edge swap would replace the two tetrahedra with two new tetrahedra. Such a 2-2 flip is easy to implement. If three tetrahedra are connected to E, those could be swapped for four tetrahedra connected to the new edge. The problem becomes more complicated as the number of tetrahedra around the boundary edge increases.

The solution used here is exactly that employed for higher order flips. The set of tetrahedra Θ are isolated and E is removed. Now the ring of nodes must be closed with the new edge; but, other than that addition, the higher order construction algorithm continues as described above.

2.2.3 Topological Optimization Algorithm

To optimize the mesh topologically, the results presented here consider each potential face and edge flip for each tetrahedra of a particular subset of all tetrahedra in the mesh. This subset of tetrahedra is constructed from tetrahedra that have been recently created and/or modified.

One important drawback to any programmatic implementation of tetrahedral topological transformations is the amount of information that must be constantly stored and updated or, at the very least, determined whenever each flip is considered. For example, before considering face removal, one must obviously be able to determine which two tetrahedra share that face. Similarly,

to consider edge removal one must be able to determine all the tetrahedra around that edge. Further, as each tetrahedra always has four faces and four nodes, the only connectivities with a constant size are an element-to-element lookup and an element-to-node lookup. All other connectives vary in size and, thus, are not amenable to direct memory allocation. For example, each node is connected to an indeterminate number of other nodes and, thus, the size of a node-to-node lookup cannot be predetermined. No matter what implementation strategy is chosen, there will most certainly be a lot of "bookkeeping."

To tackle this problem, the code used for this research has implemented three different connectivity lookups: a node-to-node, node-to-tetrahedra, and tetrahedra-to-tetrahedra connectivity. The node-to-node and node-to-tetrahedra connectivities are updated immediately whenever a tetrahedron is created, destroyed, or changed via topological transformation. These two connectivities are then used to create the tetrahedra-to-tetrahedra connectivity. Also, the set of tetrahedra around a particular edge is determined as needed by intersecting the two node-to-tetrahedra lookups associated with the nodes of the edge. No claim is made about the efficiency of this scheme except that the timing of the cases run is not intractable.

The general topological flip algorithm employed here is as follows: Begin with a list of tetrahedra Θ to consider. As indicated above, this list is populated from tetrahedra recently added to the mesh and with tetrahedra whose nodes have been moved in the smoothing process. (See Section 2.3.) While Θ is non-empty, choose the tetrahedron, $T \in \Theta$ with the greatest C_W (i.e. worst cost) and remove it from Θ. Loop over the faces of T and check if a 2-3 flip with its face neighbor is viable. If the flip is viable *and* the maximum cost of the three new tetrahedra is less than the

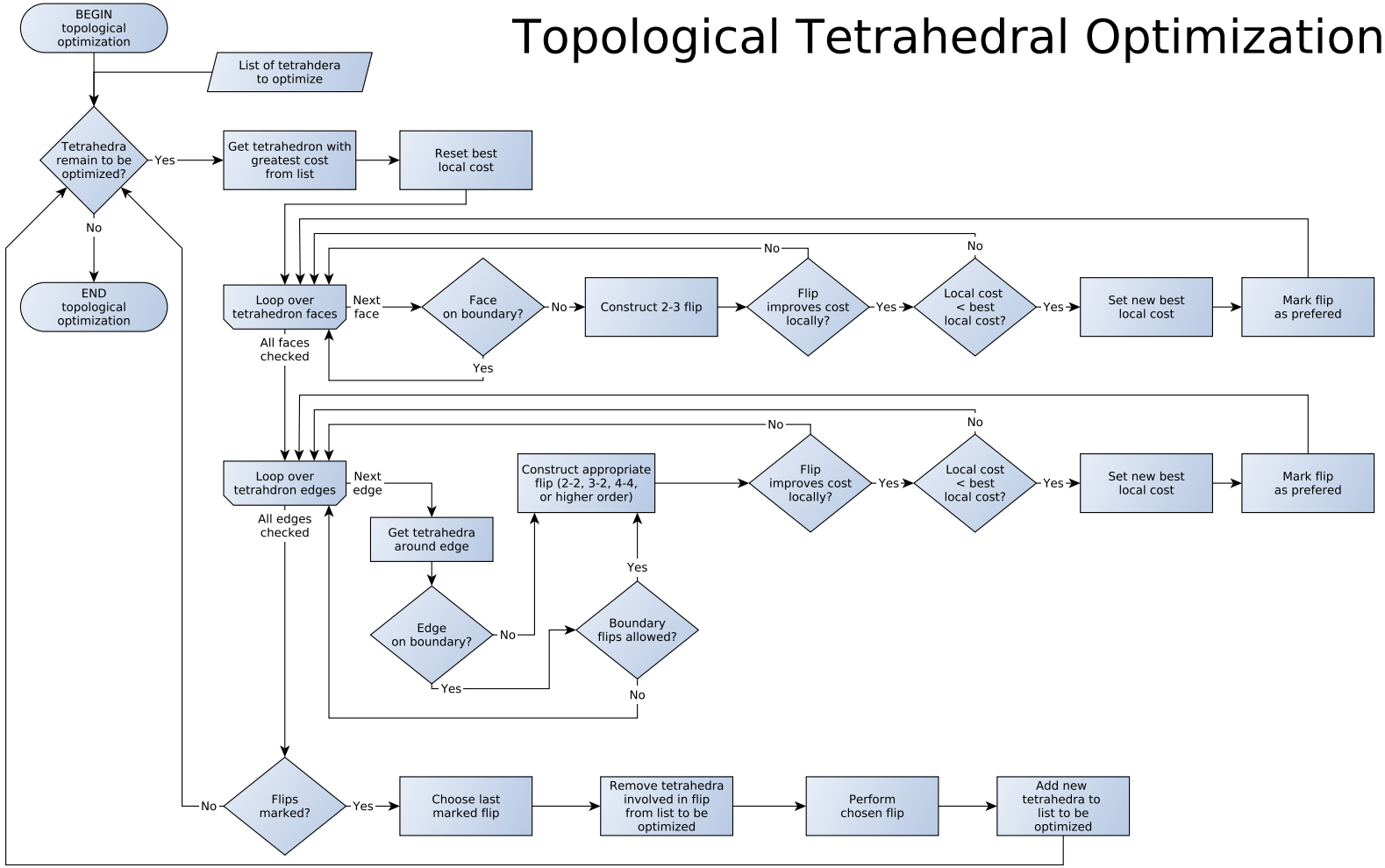


Figure 2.6 Topological Tetrahedral Optimization

maximum cost of the two original tetrahedra *and* the maximum cost of the three tetrahedra is less that the maximum cost of the best flip considered so far, mark the flip as preferred.

Next, loop over all the edges of T. For each edge, collect the ordered set of tetrahedra around that edge. If the edge is on a boundary, consider whether boundary flips are allowed as they may change the boundary triangulation against the desired behavior. Construct the flip appropriate for the number of tetrahedra around the edge. This flip may be 3-2, 4-4, higher order, or 2-2 if on a boundary. As with the face flips, if the flip is viable *and* the maximum cost of the tetrahedra created by the flip is less than the maximum cost of the original tetrahedra *and* the maximum cost of the tetrahedra created by the flip is less that the maximum cost of the best flip considered so far, mark the flip as preferred.

Now that all possible topological changes for T have been considered, the last flip (if any) marked as preferred is performed. In this way, the best flip, in the sense that it reduces the cost the most, is chosen. Finally, any new tetrahedra created are added to Θ to be considered for topological optimization. Also, any nodes involved in the flip are added to a list of candidates for smoothing. (See Section 2.3.) A detailed flowchart of the entire topological optimization process is given in Figure 2.6.

There are an enormous number of algorithmic considerations that must be taken in implementing this scheme. Not all of them are discussed here but a few important issues follow. First, care must be taken to avoid an infinite processes of flipping back and forth between two configurations with the same maximum c_W (such as in a co-spherical distribution of nodes.) This trap can be avoided by ensuring that flips are only marked if they improve the situation by some tolerance. Thus, flips that produce precisely equal maximum costs are not marked.

Second, since we are always choosing the tetrahedron from the list with the worst cost to consider for flipping, it is efficient to keep this list sorted by tetrahedral cost. However, any tetrahedra involved in a flip must be removed from the list. The solution implemented for this work is to keep two C++ stl::map containers, one of which is sorted by cost and contains the integer identifying the tetrahedron and the other which is sorted by the integer and contains the tetrahedron's cost. Thus a bi-mapping has been created between tetrahedral identifier and cost. With this framework, it is quite efficient to remove and add tetrahedra based on cost or identifier.

Finally, looping over tetrahedra, as opposed to faces and/or edges, necessarily causes extra work as, strictly speaking, every flip will be considered multiple times. For example, if every tetrahedra around an edge is in the list of consideration for flipping, then that particular edge flip will be considered for every one of those tetrahedra. This repetition will cause a good deal of unnecessary computation. To avoid this, tetrahedra that have had all of their edges and faces checked for potential flips but had no flips performed are flagged. Anytime any flagged tetrahedron is considered in the flip of another tetrahedron, that flip is ignored (i.e. not marked) before construction. (In consideration of space, this flagging is not referenced in Figure 2.6.) However, anytime a flip is performed, all the tetrahedra that share an edge or face with any of the new tetrahedra are un-flagged so they may be considered in further flips.

2.3 Optimization via Node Perturbation

As indicated in the literature review in Section 1.6, nodal perturbation, or smoothing, can be a powerful tool for improving the quality of the mesh.

Programmatically speaking, mesh smoothing is a much easier operation than topological transformations as the only changes made to the mesh are the locations of the vertices themselves. That is, no connectivity ever changes so all one has to do is update the coordinates of the nodes. The only lookups one requires are the tetrahedron-to-tetrahedron and the node-to-node connectivities and, again, any smoothing algorithm will not change these. Also, in the algorithm presented here, all of the nodes in a specified set are considered for perturbation at once. Thus, there is no need to worry about a "worst" node and, so, no need to keep a sorted list as was the case for tetrahedra in topological optimization. Generally, mesh smoothing is not as burdened with the same machinery as topological optimization.

The general node smoothing algorithm is as follows. Begin with a prescribed number of smoothing sweeps, a set of nodes N marked for perturbation, and some smoothing parameter (i.e. multiplier) β. As long as there are sweeps remaining and the maximum node perturbation is greater than some overall threshold, loop over all tetrahedra. For every tetrahedron, T, that has any of its four nodes marked for perturbation, calculate the cost of T and its cost-gradient contribution to the four nodes. Also, calculate the inscribed radius, r_T, of T. If the cost of T is greater than any other tetrahedra connected to the node, then store this gradient vector as the gradient for the node. In other words, only the gradient from the highest cost tetrahedron surrounding the node is stored. Now, after all tetrahedra are checked, loop over all nodes. For every marked node, reverse the direction of its gradient vector to produce a vector, $\mathbf{p}$, that points in the direction of greatest descent of the greatest cost tetrahedra surrounding the node. For each node, choose a threshold perturbation distance, δ_n, equal to the minimum inscribed radius of the tetrahedra surrounding the

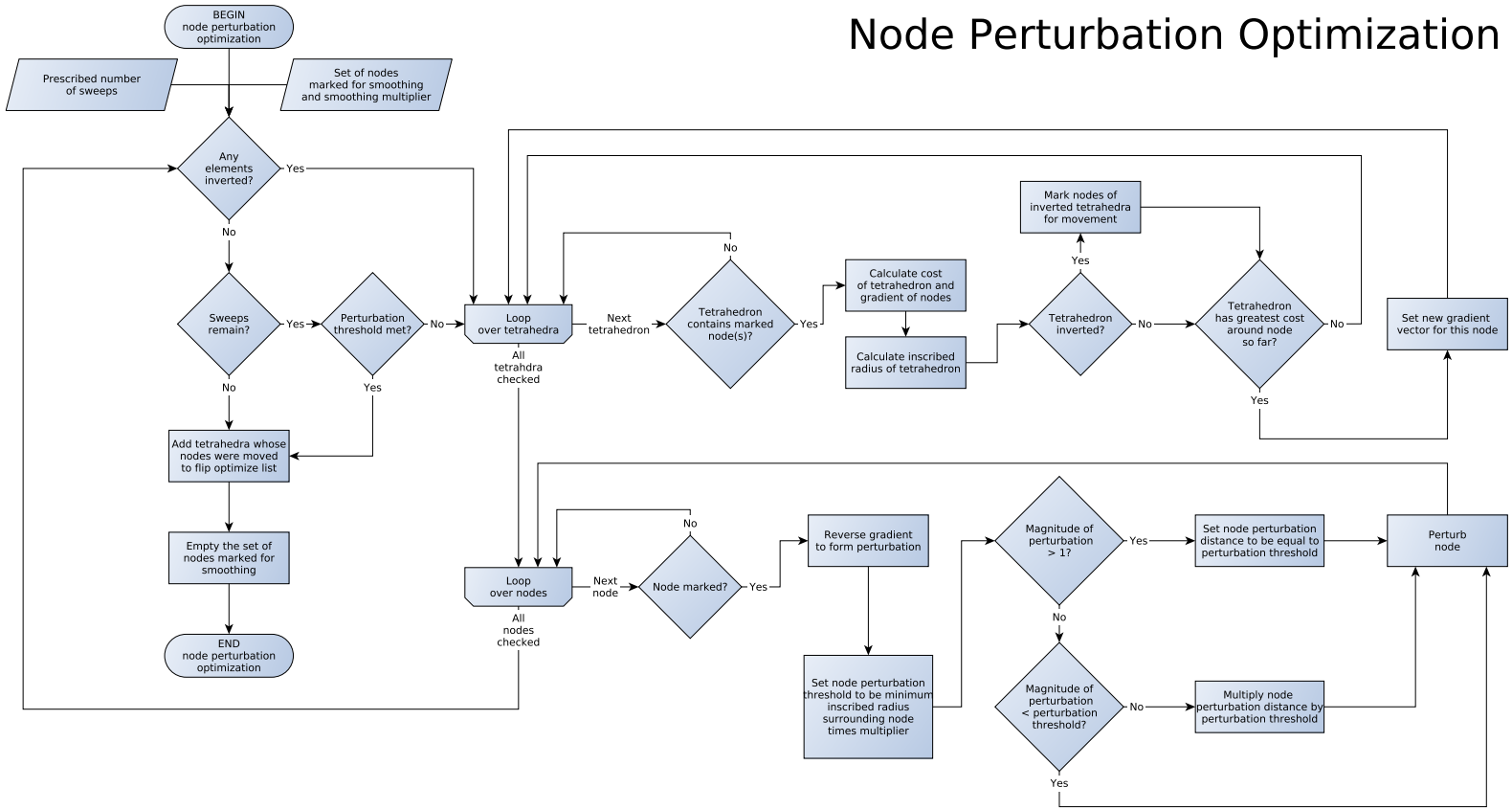


Figure 2.7 Tetrahedral Optimization via Node Perturbation

nodes times the smoothing multiplier. That is

$$\delta_n = \beta \cdot \min\{r_n\} \tag{2.25}$$

where $\{r_n\}$ is the set of inscribed radii of the tetrahedra surrounding the node. If the magnitude of $\mathbf{p}$ is greater than one, then clip its magnitude to the threshold perturbation distance. In symbols,

$$\text{if } \|\mathbf{p}\| > 1 \text{ then } \frac{\delta_n}{\|\mathbf{p}\|}\mathbf{p} \to \mathbf{p} \tag{2.26}$$

If the magnitude of $\mathbf{p}$ is less than or equal to one but still greater than the perturbation threshold, then multiply its magnitude by the threshold perturbation. Again, in symbols,

$$\text{if } \|\mathbf{p}\| \leq 1 \text{ and } \|\mathbf{p}\| > \delta_n \text{ then } \delta_n\mathbf{p} \to \mathbf{p} \tag{2.27}$$

Otherwise, if the magnitude of $\mathbf{p}$ is less than one and less than the threshold perturbation, do not modify $\mathbf{p}$. Finally, perturb the node by $\mathbf{p}$.

Note that the clipping of the perturbation distance by a factor of smallest inscribed radius surrounding the node is necessary to keep this methodology stable. If one simply uses the magnitude of the gradient as a perturbation distance, the method has the tendency to produce invalid tetrahedra and the whole process quickly produces invalid meshes that this gradient methodology cannot "untangle." So, for all intents and purposes, the method described is an under-relaxed algorithm.

One other important consideration taken in this algorithm is whether or not any of the tetrahedra in question are inverted. If so, all of their nodes are marked for perturbation (i.e. added

to N.) Also, tetrahedra whose nodes are nearly coplanar (i.e. kites) will have an inscribed radius very close to zero and, so, the perturbation threshold will be very close to zero. Thus, care must be taken to ensure $\delta_n \gg 0$. Finally, the overall process is not allowed to terminate until all the tetrahedra are valid. This consideration is simply a stopgap measure to keep the algorithm from terminating with an invalid mesh. Otherwise the algorithm terminates after the prescribed number of sweeps is reached. Finally, all tetrahedra connected to the nodes of N are added to a list of tetrahedra as candidates for topological optimization and N is emptied.

A detailed flowchart of this process is given in Figure 2.7.

2.4 Mesh Convergence Cycle

While expressed as separate algorithms, topological optimization and node smoothing are used together to improve the quality of the mesh. The combination is fairly simple but effective. What follows is an explanation of how the procedures are used together as well a some examples.

2.4.1 Mesh Convergence Cycle Algorithm

The algorithm is as follows. An overall number of optimization cycles is chosen. Then all tetrahedra with a c_W greater than some given cost threshold c (usually 0.5) are added to the list of tetrahedra Θ as candidates for topological optimization. Also, all the nodes of each $T \in \Theta$ are added to the set of nodes N to be smoothed. A topological optimization pass is performed which empties Θ. Recall that any flip performed will ensure that all the nodes involved will be added to N and, as a consequence, N may grow in size. After this, a prescribed number of smoothing sweeps are performed. Recall that all the tetrahedra whose nodes have been smoothed are added to Θ and N is emptied. Finally, all tetrahedra with a c_W greater than the given cost c added to the list of

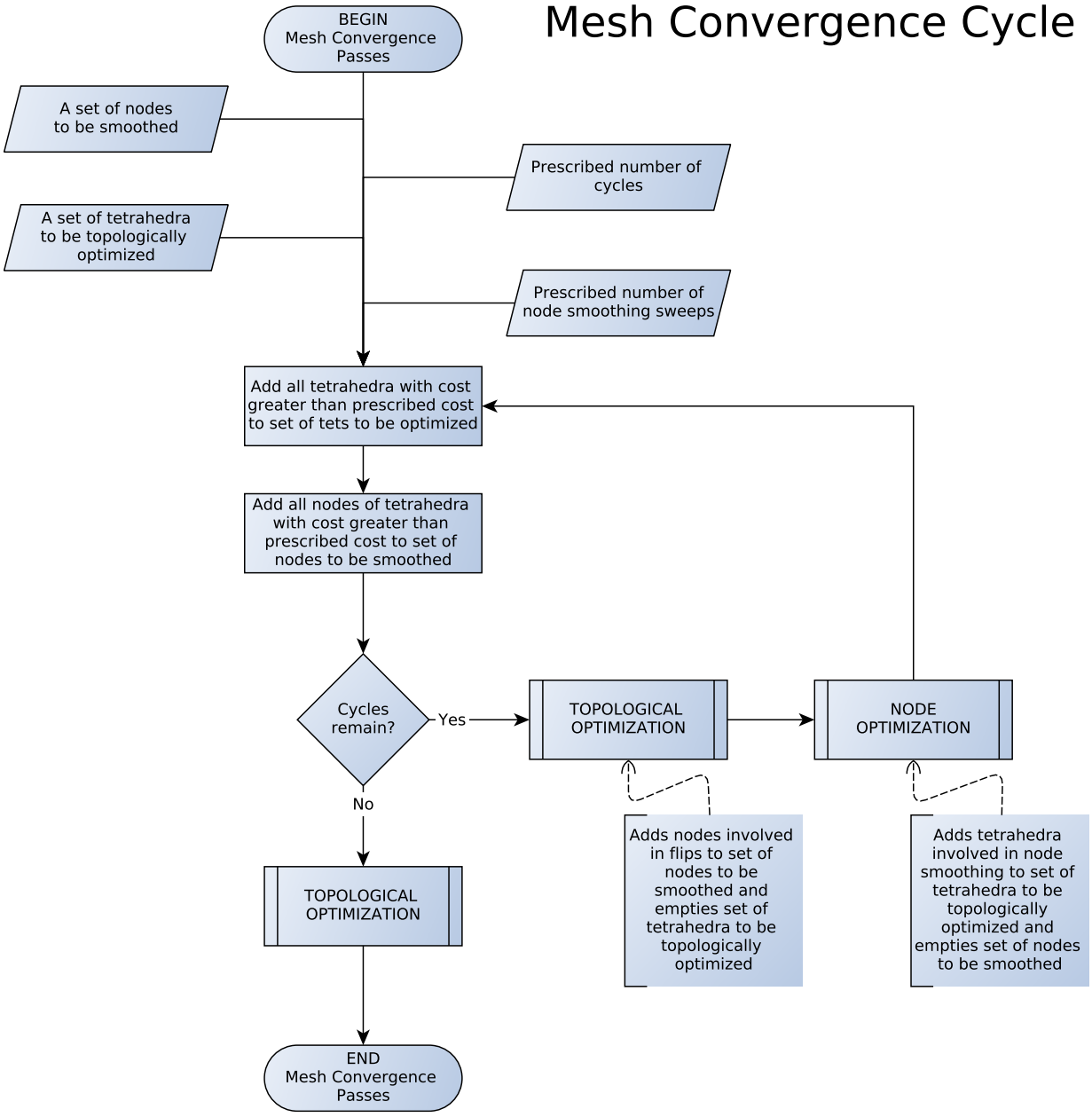


Figure 2.8 Mesh Convergence Cycle

tetrahedra Θ as candidates for topological optimization and the nodes of each $T \in \Theta$ are added to the set of nodes N to be smoothed. The entire process is repeated until the number of cycles is exhausted. A final topological optimization pass is then performed. A detailed flowchart of this algorithm is given in Figure 2.8.

At first brush, it may appear as if the number of tetrahedra slated for optimization and the number of nodes involved in smoothing will only grow. However, note that both Θ and N are emptied at the end of the topological optimization and node smoothing processes respectively. So, at the beginning of each topological smoothing pass, Θ contains only tetrahedra above the prescribed cost threshold c as well as the tetrahedra involved in the previous node smoothing pass. Similarly, at the beginning of each node smoothing pass N contains only the nodes of tetrahedra with cost above c as well as the nodes of any tetrahedra involved in flips from the previous topological optimization pass. In this way, a tetrahedron and its nodes that fall below a cost of c will fall out of consideration in both the topological optimization and node smoothing passes.

The effectiveness of this overall mesh improvement scheme lies in the combination of the two techniques. The topological scheme provides the node smoothing scheme with the best grid connectivity attainable from the given set of nodes. The smoothing scheme provides the topological optimization algorithm with the best node positions available for the given connectivity. The result is an overall algorithm that drives both the connectivity and vertex positions towards a quality mesh.

It should be noted that the name chosen to describe this process is something of a misnomer as it rarely "converges." While there is almost always a step in the algorithm at which no more topological optimizations are performed, the process rarely reaches a stage where the vertex smoothing as described yields movement below some small tolerance. The nodes tend to "jitter"

between acceptable positions until the number of prescribed cycles is exhausted. This behavior is due, at least in part, to basing the choice of gradient vector upon the worst cost tetrahedra. Thus, the worst tetrahedra connected to a node can change and the gradient vector will "ping-pong" directions between smoothing passes. In practice, however, this lack of convergence is of no concern as the number of smoothing sweeps and overall number of convergence cycles is limited.

2.4.2 Mesh Convergence Cycle Examples

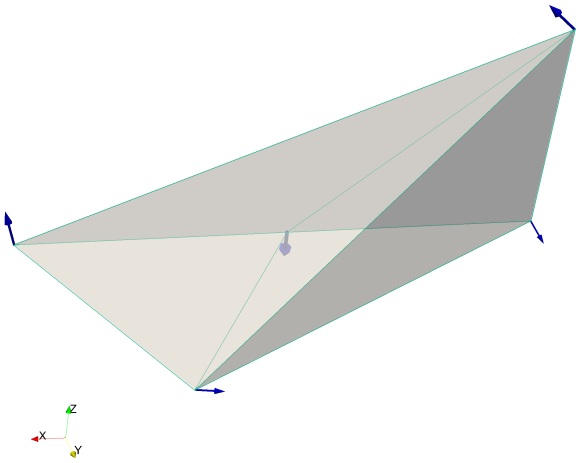
Experience shows this algorithm to be highly effective at improving the quality of tetrahedral grids. Indeed, as will be demonstrated in Chapter 3, this procedure can take meshes with near degenerate tetrahedra and optimize them into grids with completely acceptable quality. What follows are two examples that show the mesh convergence cycle on small cases. These examples are intended to convey a sense of how the algorithm works in practice.

Figures 2.9 and 2.10 give examples of the mesh convergence cycle. Each figure shows a set of tetrahedra either before or at some step in the convergence cycle. Each node is annotated with its scaled, inverted gradient vector. Each figure shows a set of coordinate axes since the perspective varies from picture to picture. The maximum WCN of all the tetrahedra depicted is noted in the figure's caption. In each case, a single node is fixed but all others are allowed to move and the convergence cycle routine is called with 5 smoothing sweeps between each topological optimization pass. Any sort of flip is allowed including boundary flips.

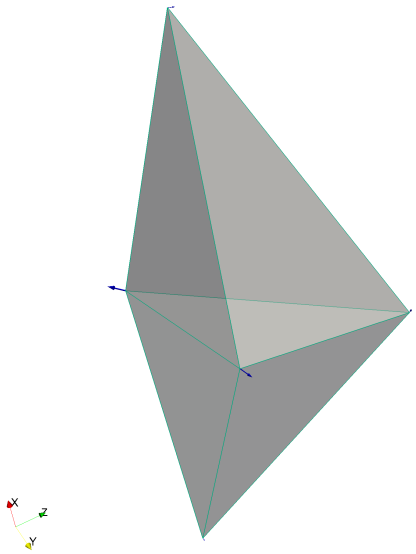
Figure 2.10 depicts two tetrahedra that share a single face. In the optimization process, no flips ever occur (because none would make an improvement) so all the optimization is from vertex smoothing. Note the two tetrahedra quickly converge to two equilateral tetrahedra. In Figure 2.9d

it appears the vectors have disappeared because their magnitude is zero. That is, in this particular instance, the mesh convergence process has, in contrast to the discussion above, actually converged.

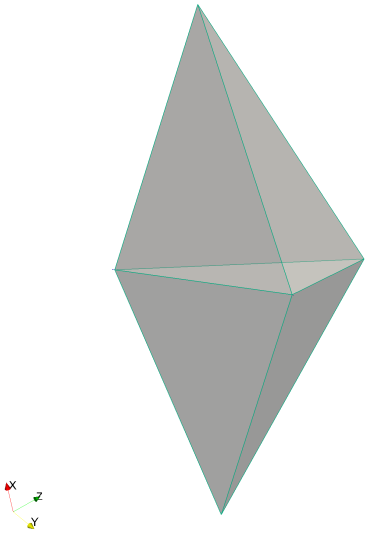
Figure 2.9 shows a small mass of tetrahedra which were constructed mostly at random. This grid begins as nearly invalid. This fact is highlighted by the size of the vectors in Figure2.10b. The mesh improves substantially after one set of flips and, after only three convergence cycles, the maximum WCN has been reduced to a manageable level. Finally, after 100 cycles, the mesh has reached a completely acceptable state.



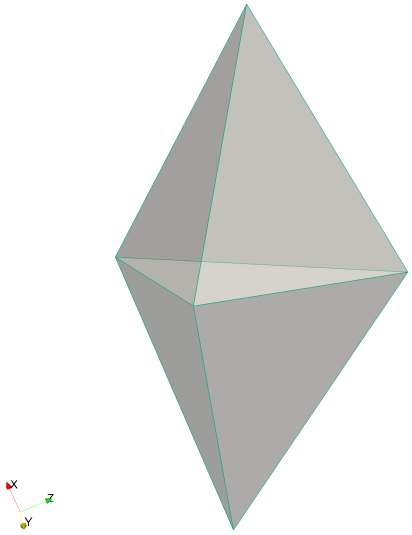
(a) original mesh, max WCN 2.167



(b) after 3 convergence cycles, max WCN 1.224

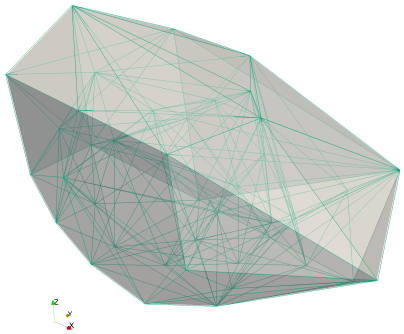


(c) after 5 convergence cycles, max WCN 1.054

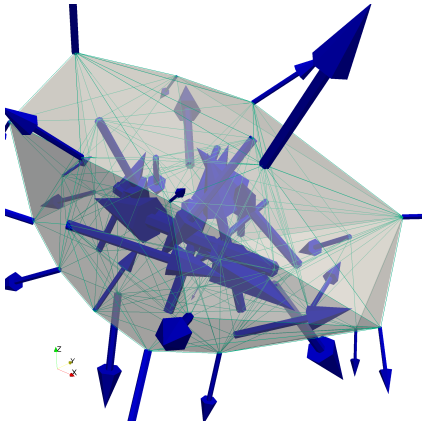


(d) after 20 convergence cycles, max WCN 1.000

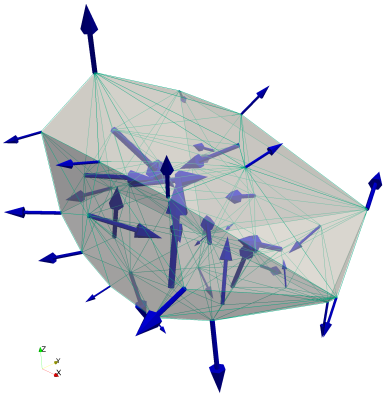
Figure 2.9 Mesh Convergence Cycle, Two Tetrahedra



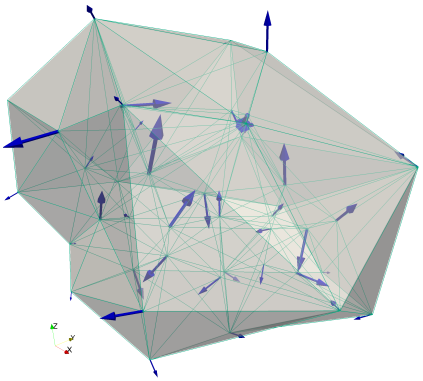
(a) original mesh, vectors omitted, max WCN 1322



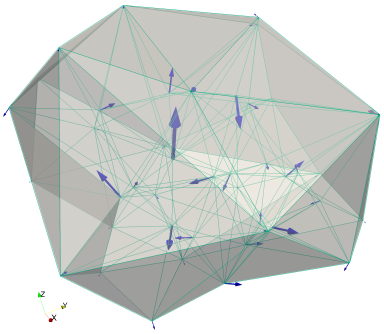
(b) original mesh with perturbation vectors



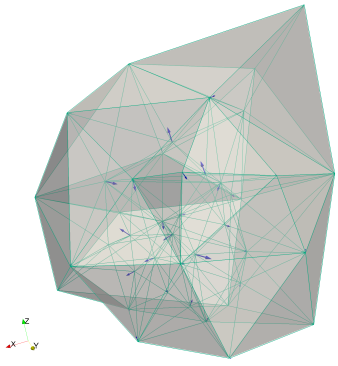
(c) after one round of flips, max WCN 113.3



(d) after 3 convergence cycles, max WCN 3.119



(e) after 8 convergence cycles, max WCN 1.692



(f) after 100 convergence cycles, max WCN 1.450

Figure 2.10 Mesh Convergence Cycle, Forty-two Nodes

2.5 Other Mesh Creation Techniques

The shining jewels of the algorithms presented in this work are most certainly the topological and smoothing procedures themselves. As such, the other mesh creation techniques are, admittedly, somewhat "ad hoc." However, they are not without merit. Given below is an overview of the procedures used in the mesh creation cases presented in Chapter 3.

2.5.1 Point Insertion

Two types of point insertion have been implemented for this work: a traditional Bowyer-Watson style insertion and a three-dimensional Lawson-style insertion. The Bowyer-Watson style insertion is as described in Section 1.3.1. All the tetrahedra with circumspheres that contain the new vertex are deleted and new tetrahedra are created by connecting the new vertex to the vertices of the deleted hull. Since the Bowyer-Watson technique is faster, this technique is used in the initial phases of mesh creation because no smoothing or topological changes have yet been applied that would destroy the Delaunay property.

Once any mesh smoothing or flips have been applied, the grid is no longer Delaunay. While it would still be possible to utilize a Bowyer-Watson style algorithm for insertion as any new vertex will at least be contained in the circumsphere of the tetrahedra that contains the vertex itself, the usefulness of this insertion technique is diminished since all of the guaranteed quality constraints referenced in Chapter 1 no longer apply. So, instead, new vertices are inserted using a Lawson-style scheme. That is, the tetrahedron containing the new vertex is deleted and four new tetrahedra are created by connecting the new vertex to the faces of the deleted tetrahedron. Note it is possible for the new vertex to be on a face or edge in the existing tessellation. In such a case, all the tetrahedra

containing that face or edge are deleted and the new vertex is connected to all the faces of the deleted tetrahedra not containing the new vertex. (N.B. if a new vertex is on an edge of a tetrahedron it is also on two of its faces.)

Immediately after this insertion, the mesh convergence cycle routine is called. (See Section 2.4.) Note that the very first optimization in the convergence cycle is topological. In this way, the Lawson-style scheme mimics the classic two-dimensional Lawson scheme with the addition of node smoothing. The new tetrahedra and new vertex will have been added to the appropriate Θ and N during the insertion process. The quality threshold is set to some high value (usually 0.8) so that not much optimization will occur outside of the region around the new vertex. As a matter of convenience, this operation of Lawson-style insertion immediately followed by topological optimization and vertex smoothing is referred to as Lawson Style Optimization Insertion or LSOI.

2.5.2 Bounding Box and Octree-Based Pseudo-Tiling

Before inserting points, one must have something into which to insert them i.e. some initial grid. It seems natural to create as high quality of an initial mesh as possible. So, in this work, a background mesh consisting of high quality tetrahedra is created into which points from the prescribed boundary triangulation are inserted.

As has been discussed, the highest quality element for our purposes is the equilateral tetrahedron. Unfortunately, there is no way to tile space with regular, equilateral tetrahedra and thus one cannot create the "perfect" background mesh in the sense that all the elements will have maximum quality. However, from a practical point of view, one can come close. The method employed for this work creates an octree-based upon the provided boundary triangulation and uses

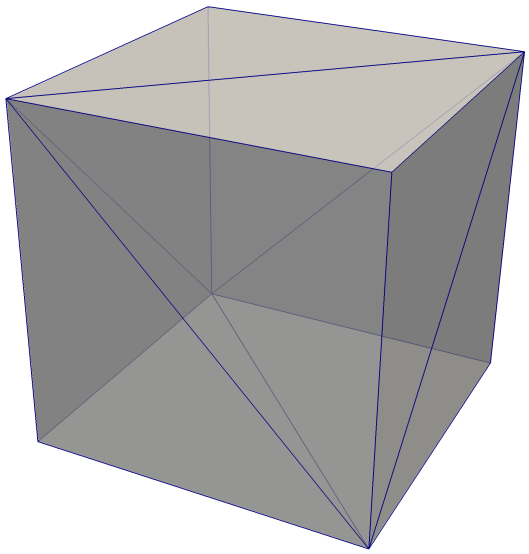


Figure 2.11 Initial Bounding Box

this construct to create points for interior tetrahedra. This method is referred to here as octree-based pseudo-tiling.

First, an eight point bounding box composed of five tetrahedra is created. This box is sized so that it encompasses the extents of the boundary triangulation. This initialization is precisely the same as employed by Watson [13] and virtually every other tetrahedral mesh generation algorithm. The bounding box is depicted in Figure 2.11.

Based on this bounding box, a spatial octree is created to hold the triangles from the prescribed boundary triangulation. Then this octree is itself refined so that none of its leaves have a longer diagonal than twice the longest edge in the boundary triangulation. The octree is regularized so that an octant with any children is ensured to have all eight of its children. Then the octree is refined so that no leaf octant is adjacent to any leaf more than one level larger than

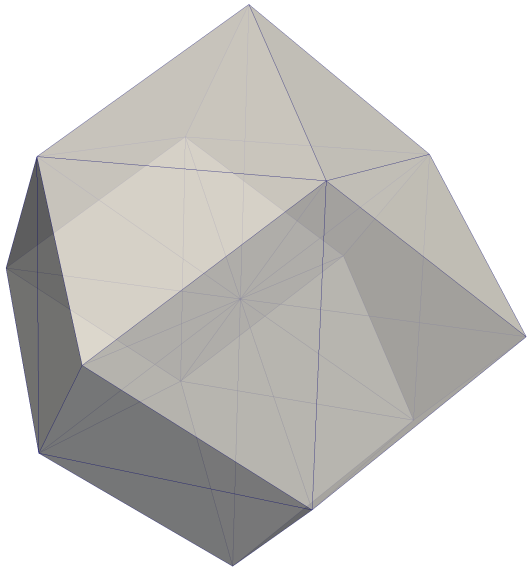


Figure 2.12 Nearly Equilateral Space Tiling Tetrahedra

itself. Finally, points from the lower corner and midpoints of these octants are added to the overall tetrahedralization. This process creates a nice gradation of points based on the spacing that arises from the boundary triangulation. An example of the bounding mesh created in this way on the Three Boxes Case in Section 3.4.2 is shown in Figure 3.13.

The elements immediately subtended by this process consist of a set of nearly equilateral tetrahedra that tile space. An example of this configuration is shown in Figure 2.12. Each of these elements has a WCN of 1.054 which is well within the acceptable range. Elements created from points generated at transitions of spacing in the octree have a higher WCN. However, experience shows that the optimization process readily improves their quality.

2.5.3 Boundary Recovery

As previously stated, one of the goals of this research is to explore the creation of meshes with a prescribed triangular boundary. In other words, the final mesh is required to have a particular set of triangles on the boundary as opposed to simply respecting some polyhedra as is done by Chew, Shewchuk and others. (See Section 1.4.2.) As such, a set of triangle recovery and enforcement techniques have been employed throughout the mesh optimization and creation routines.

First and foremost, if at any time during Lawson-style point insertion or topological optimization an edge prescribed in the triangulation is created, that edge is marked as inviolable and no topological operation is ever allowed to remove it. Similarly, if any triangular face is ever extant in the mesh no edge creation is ever allowed to pierce the face. The only method by which this piercing could occur is via a 2-3 flip and, thus, the inviolable face is easy to protect. Note that such edges or faces could be destroyed with a Bowyer-Watson style insertion but that technique is only employed during bounding mesh creation before any optimization.

After all the points of the boundary triangulation have been inserted and the mesh has been optimized, each boundary triangle is processed to check if it is the face of some tetrahedron in the mesh. If so, that boundary triangle is marked as recovered and the next one is processed. If not, first an attempt is made to insert the edges of the triangle using the topological operations implemented for mesh optimization. If that is successful, an attempt is made to remove edges that pierce the desired face. If either of the edge or face recovery routines fail to recover the desired triangle, that triangle is marked for further processing.

Experience shows that the above technique is able to recover over 97% of the desired triangles before any point insertion is necessary. Of course, all of the triangles must be recovered

for the method to be successful. Before any further boundary recovery, mesh convergence cycles are run on the entire grid. Recall, that any edges and faces that have been correctly recovered will not be changed and so none of the recovery process heretofore completed will be lost. After the cycles are complete, an attempt is made to recover any remaining unrecovered triangles using only topological changes as described above. If this process still fails to recover the triangle, points are inserted near the desired triangle face and the mesh convergence cycle is used to optimize the mesh locally. Then, as before, flips are used in an attempt to recover the triangle. This process may require the insertion of several points over several iterations and, frankly, there is no theoretical guarantee it will recover the desired triangle. However, the algorithms as implemented were successful in all attempts. Each of the cases presented in Chapter 3, several of which have complex boundaries, employed this boundary recovery solution.

2.5.4 Edge Driven Refinement and Simply Partitioned Tetrahedra

After the pseudo-tiling and boundary recovery, it remains to improve the quality of the mesh. However, the LSOI technique has no proven quality guarantees such as those shown for Bowyer-Watson style algorithms by Chew and Shewchuk. (See Section 1.4.2.) So, the chosen methodology for further mesh refinement is based upon a geometric constraint one can be sure converges, namely edge length refinement. Simply, if one chooses to insert points based upon refining internal edge lengths, such a process is guaranteed to converge as those edges above some prescribed length will be subdivided until they are eventually shorter than whatever prescribed length.

To this end, two separate edge refinement techniques have been implemented. These techniques insert new vertices at the midpoints of edges by LSOI. First edges are refined based upon some multiple of the maximum boundary edge length. That is, any internal edge longer than the longest boundary triangle edge has a point inserted at its midpoint. Secondly, internal edges connected to boundary triangles are refined based upon the maximum length of the boundary triangle edges connected to them. In other words, any internal edge connected to a boundary node has a point inserted at its midpoint if its length is greater than some multiple of the boundary edges connected to that node.

Experience showed that refining the mesh along edges using the LSOI was successful. The most elementary example of this technique can be seen in Section 3.3. Other methodologies to choose points for insertion were explored, such as refinement based on tetrahedral cost, but none were as successful as techniques based on edge length.

In practice, edge refinement is conducted by gathering a set of midpoints and then inserting them one at a time via LSOI as opposed to inserting the new vertices as they are created. The set of midpoints is created by looping over tetrahedra, checking edges lengths, and ensuring edges are not marked for refinement twice with the techniques discussed in Appendix B. Thus, it is possible that LSOI will have destroyed or moved the edge some node n was created to bisect before it is inserted. In such a case, LSOI will first place the point by subdividing whatever tetrahedron in which the n is found. So, initially, n will only be connected to four other nodes and four tetrahedra. If no flips are immediately performed, n will be smoothed and tend towards the center of the four vertices to which it is attached.

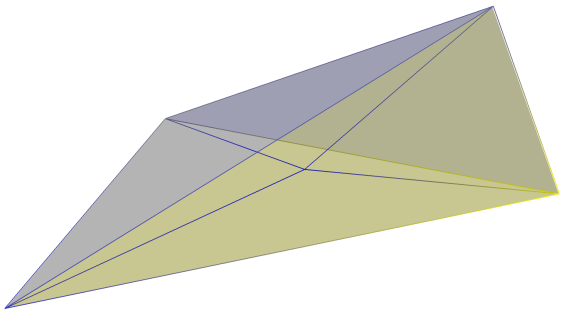


Figure 2.13 A Simply Partitioned Tetrahedron

Here, this construct is referred to as a simply partitioned tetrahedra and n is referred to as a simply connected node. Figure 2.13 shows such a configuration. The four tetrahedra connected to n will not be of the best quality. However, it is very easy to find n after the fact as it will be connected to exactly four other nodes and not be part of a boundary triangle. As part of the mesh creation process, such simply connected nodes and their associated tetrahedra are deleted and replaced with a single tetrahedra.

2.5.5 Overall Mesh Creation Algorithm

For clarity's sake, the mesh creation steps used in this research are summarized, without much explanation, as follows: First, the bounding box with octree pseudo-tiling is created via a Bowyer-Watson insertion. Then, the nodes from the boundary triangles are inserted via Bowyer-Watson. The mesh convergence process is called, the boundaries are recovered, and the mesh

convergence cycle is called again. Edges connected to boundaries are refined and the mesh convergence cycle is called. Edges longer than the longest boundary edge are refined and the mesh convergence cycle is called. Finally, simply partitioned tetrahedra are removed and the mesh convergence cycle is called for the final time.

Generally speaking, this process is successful. Some of its advantages and drawbacks are discussed in Chapter 4. Chapter 3 is replete with examples of this process.

CHAPTER 3

RESULTS

The actual C++ program used to optimize and create meshes with the methods described in Chapter 2 is called by the uncreative but utilitarian name Tetmesh. Tetmesh is approximately 9800 lines of code excluding personal libraries not solely devoted to this project of over 4500 lines. (These counts exclude comments and blank lines.) Volume meshes solely created using Tetmesh will be designated with this name.

Surface meshes and volume meshes used for pure optimization procedures were produced in the commercial grid generation software Pointwise[1]. Some grids used for optimization were created with Si's open source software TetGen[2]. (See Section 1.4.2.) Grid visualization and graphs related to grid quality were produced with the open-source application ParaView[3]. What follows are examples of the mesh optimization and generation process utilizing these tools.

[1] http://www.pointwise.com
[2] http://wias-berlin.de/software/tetgen/
[3] http://www.paraview.org

3.1 Aspect Ratio

Along with weighted condition number, the other quality metric reported in the cases below is aspect ratio, sometimes denoted AR. The aspect ratio of a tetrahedron T is defined as

$$\mathrm{AR}\,(T) = \frac{r_c}{3r_i} \tag{3.1}$$

where r_c is the radius of the sphere containing all four vertices of T (i.e. the radius the circumsphere) and r_i is the radius of the largest sphere that may be inscribed within T. The factor of 3 in the denominator ensures that an equilateral tetrahedron will have AR = 1.

This metric is reported for two reasons. First, such reporting demonstrates that quality metrics other than WCN also show improvement under these mesh optimization techniques. Second, and most importantly, the grids from Pointwise and TetGen are created with metrics more closely associated with AR. In particular, the TetGen meshes were created with a quality flag that refines the grid based upon a ratio of a tetrahedron's circumradius to shortest edge. This measure is similar to AR. While there is no technical information publicly available on Pointwise's mesh generation technique, it is clear the method is based on the Delaunay property and generally produces meshes with low AR.

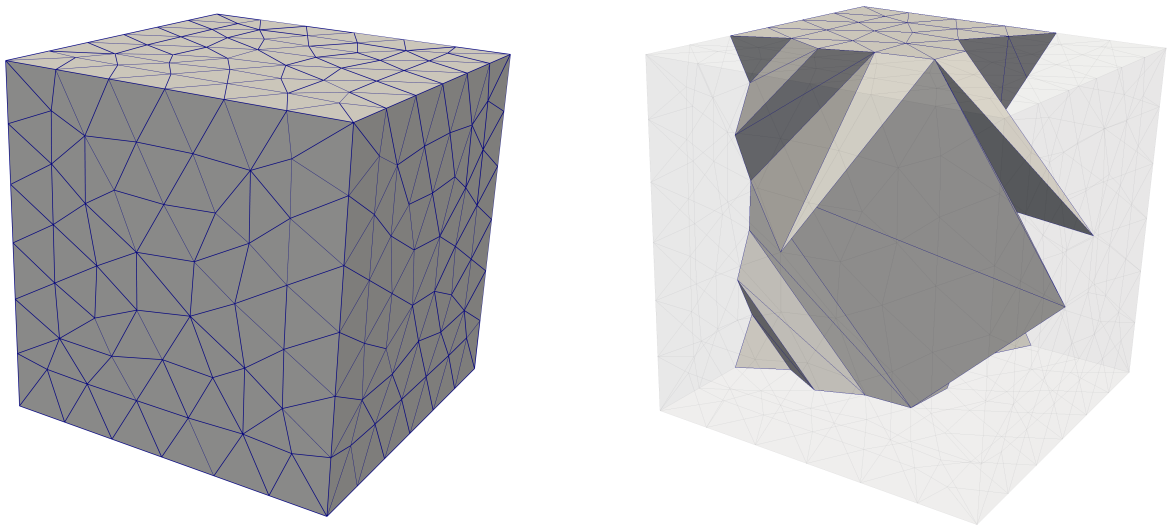
In general, the optimization techniques, though completely based upon WCN, also lower the AR of elements of the mesh. Such improvement is in spite of there being no direct mathematical relationship between WCN and AR other than they both favor equilateral tetrahedra. This fact is submitted in support of the choice of WCN as a quality metric.

3.2 Numerical Experiment

To show the general efficacy of LSOI, a numerical experiment was conducted by randomly placing 172 nodes in a box with a triangulated surface mesh. The number of randomly placed nodes was chosen as 172 since Pointwise created a mesh with that many internal nodes using its default settings. The box in question, which is in fact cubic, has itself 284 surface nodes and 564 boundary triangles. First, the boundary of this triangulation was recovered without the addition of any points. The geometry and recovered boundary are shown in Figure 3.1.

After boundary recovery the 172 random nodes were placed in this grid in two ways. First, they were inserted with Bowyer-Watson and then the mesh convergence cycle was called on the entire grid. Secondly, they were inserted with LSOI and an overall mesh convergence cycle was called. This process was repeated 100 times for each case. Relative frequency plots for the WCN of all tetrahedra in all 100 meshes are given in Figure 3.3.

As expected, Bowyer-Watson insertion of the random points results in horrible meshes. An example is given in Figure 3.2. Note the spike in frequency at the end of the plot. This jump indicates over 8% of the tetrahedra created over the 100 meshes have WCN ≥ 4. LSOI produces a reasonable mesh with most of the tetrahedra having WCN ≤ 2. After further optimization, these LSOI meshes have improved with all tetrahedra having WCN ≤ 3. Surprisingly, the best meshes are those with the Bowyer-Watson insertion and optimization. No such mesh has any tetrahedra with WCN > 2.5 and the majority of elements have WCN ≤ 2. More discussion of these results is given in Chapter 4.



(a) geometry (b) initial mesh

Figure 3.1 Box for Random Insertion

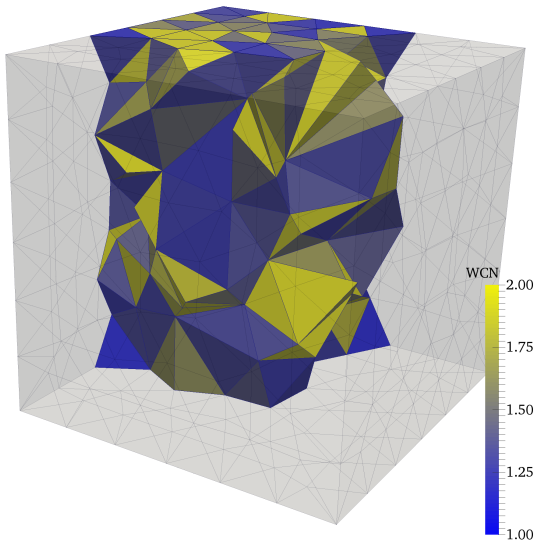


Figure 3.2 A Poor Quality Mesh from Bowyer-Watson Random Point Insertion

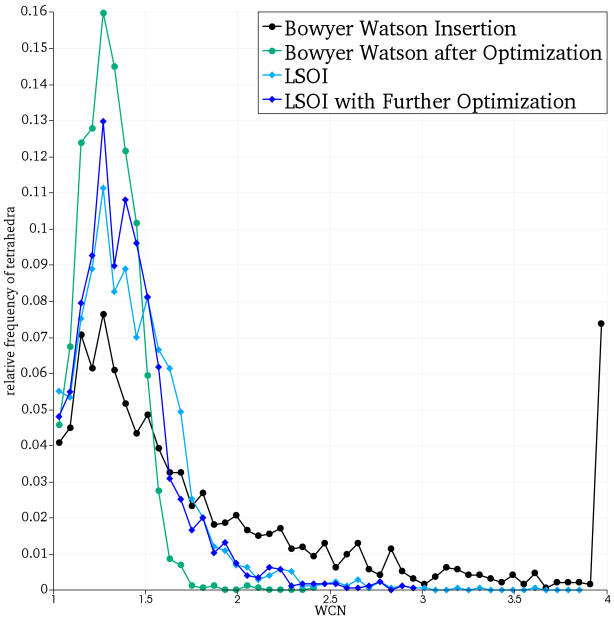


Figure 3.3 Random Insertion Methodologies Relative Frequency Plots

3.3 Edge Refinement with LSOI

For further experimentation, the internal edges of the same geometry shown in Figure 3.1 were subdivided via LSOI until 172 points were inserted. (Again, this number was chosen to match the number of internal nodes in the Pointwise mesh.) Note that, unlike in the general creation procedure, each point is inserted before the next one is chosen. A relative frequency plot of WCN comparing this mesh to the one generated in Pointwise is shown in Figure 3.4.

Generally, the LSOI mesh compares well with the Pointwise mesh. Both have a maximum WCN<2. Further, just a few more points would have made the LSOI mesh higher quality. Figure 3.5 shows the LSOI mesh for this case along with one of its worst tetrahedra. The mesh would likely improve if this tetrahedra, which is on the boundary, were refined along its longest edge.

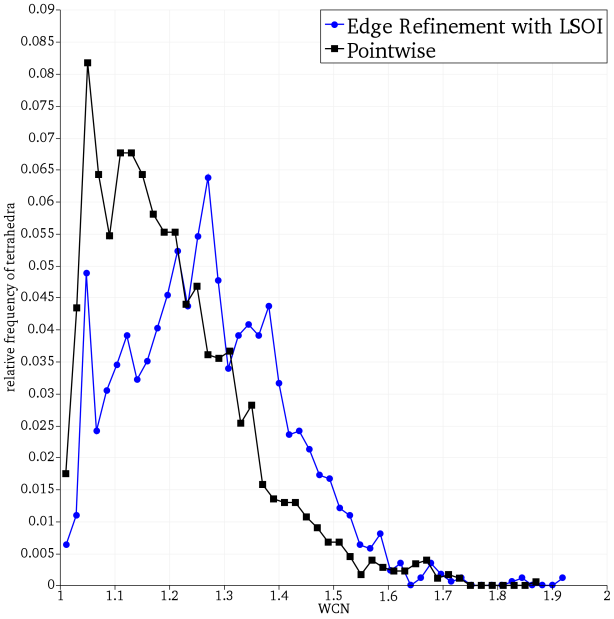
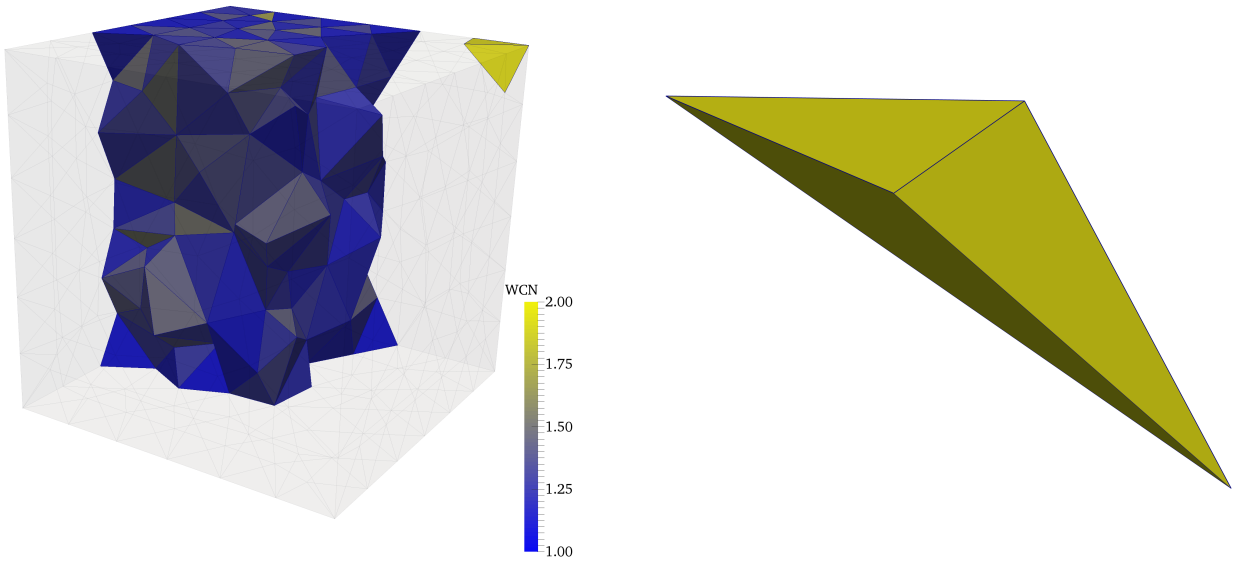


Figure 3.4 Pointwise versus Edge Refinement by Lawson Style Optimization Insertion Relative Frequency Plots

Note the LSOI mesh has a spike in frequency around WCN $\approx$ 1.3. Also, there are more tetrahedra with $1.4 \leq$ WCN ≤ 1.6 in the LSOI mesh than in the Pointwise mesh. As it turns out, this is a common outcome of the optimization process presented in this work. Recall, both the topological and vertex smoothing schemes focus on improving the worst quality elements and very high quality elements are often sacrificed in process.



(a) interior of grid with worst tetrahedron in upper right corner

(b) worst tetrahedron

Figure 3.5 Edge Refinement by Lawson Style Optimization Insertion

3.4 Optimization and Creation Cases

A number of simple to complex cases are presented below to demonstrate the methods so far described. Meshes are generated in Pointwise and, in some cases, TetGen to be utilized for optimization. Also meshes are generated in Tetmesh with the procedure outlined in Section 2.5.5. In each case, the same geometry and boundary triangulations were used to create the Pointwise and TetGen grids as well as the mesh created purely with Tetmesh. All of the boundary triangulations themselves were created in Pointwise using its Delaunay-based or advancing-front-based surface meshing technique unless otherwise noted.

Grids generated with Pointwise used the default settings with no user intervention. On the other hand, it was necessary to modify the quality parameter input so that the TetGen meshes would have a comparable number of points to the Pointwise mesh. In each of these cases, said

quality parameter had to be set well above the default. Still, as will be seen, the meshes generated by TetGen are of relatively low quality. While unintentional, this outcome was useful in that it shows the efficacy of the optimization algorithm. In any case, the low quality of the TetGen meshes should not be seen as an assailment against that code or its author. It is safer to assume the author of this work simply did not know how to properly operate TetGen.

The Pointwise and TetGen meshes were first optimized with topological changes only. Then, the meshes were optimized with the entire mesh convergence cycle utilizing 40 cycles each with 40 mesh smoothing sweeps between topological optimizations. No boundary flips are allowed in any of the optimization cases. Also, it is worth emphasizing that the optimization procedure employed on the Pointwise and TetGen meshes neither inserts nor deletes vertices. Meshes created with Tetmesh utilized the procedure outlined in Section 2.5.5 and, just as with the pure optimization cases, all of the mesh convergence cycles in those creations utilized 40 cycles each with 40 mesh smoothing sweeps between topological optimizations. Also, no boundary flips are allowed except in the Stanford Bunny case in Section 3.4.6. These details will not be belabored in the discussions below.

For each of these cases, a number of relative frequency plots of WCN are shown as well as a number of tables listing certain percentiles of WCN and AR. Every effort has been made to consistently format these items. Note that in each individual case the relative frequency plots all have the same scales although they vary between cases.

3.4.1 Simple Sphere

First, as a simple introductory case, consider meshes generated inside a sphere with a surface as shown in Figure 3.6. This surface mesh has 513 nodes and 1,022 triangles. Relative frequency plots for the optimized Pointwise and TetGen meshes are given in Figure 3.8. Table 3.1 gives the statistics for both the Pointwise and TetGen optimization cases. Figure 3.9 is a relative frequency plot comparing the Pointwise, TetGen, and Tetmesh generated grids and Table 3.2 give the statistics for that Tetmesh grid.

As was intimated above, the TetGen grid is of poor quality especially given the simplicity of the surface triangulation. Note it has a maximum AR greater than 6. Further, as can be seen from the jump in the relative frequency plot, well over 2% of the tetrahedra have a WCN $\geq$ 2.5. An illustration of all the tetrahedra generated by TetGen with WCN $\geq$ 2.0 is given in Figure 3.7. The Pointwise grid, on the other hand, is of much higher quality. In fact, from a practical point of view, there is little reason to optimize this grid.

Even though the Pointwise grid may not need it, the optimization techniques are able to dramatically improve both grids. The maximum WCN is reduced to below 1.4 for the Pointwise grid and below 1.5 for the TetGen grid. Indeed the TetGen grid has gone from unacceptable to quite nice. Note that in both cases, topological optimization alone is not enough to bring satisfactory improvement in grid quality. Indeed, flips have little effect on the Pointwise mesh at all. However, flipping is enough to at least make the quality of the TetGen mesh tolerable.

Without yielding any particular insight, the mesh creation process in Tetmesh has produced a grid comparable if not better than the original Pointwise mesh. Note the number of nodes in each

mesh differs by less than 9%. The quality of the Tetmesh grid is nearly that of the fully optimized Pointwise and TetGen grids.

Finally, note the spikes in the relative frequency graphs around WCN of 1.3 in both the pure optimization cases as well as the Tetmesh created grid. This behavior is exactly the same as noted in Section 3.3. Indeed, this phenomenon will be seen over and over again in the cases that follow.

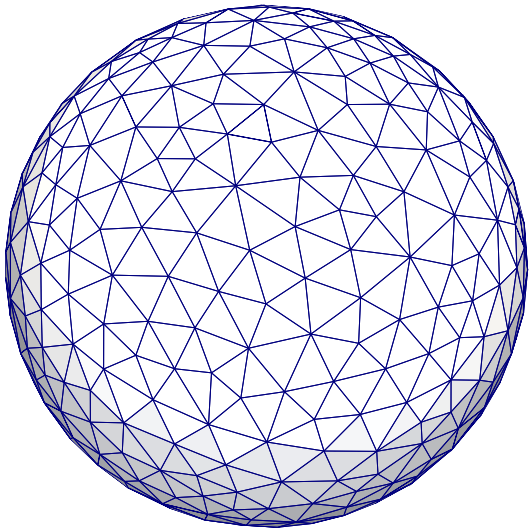


Figure 3.6 Simple Sphere Geometry and Surface Grid

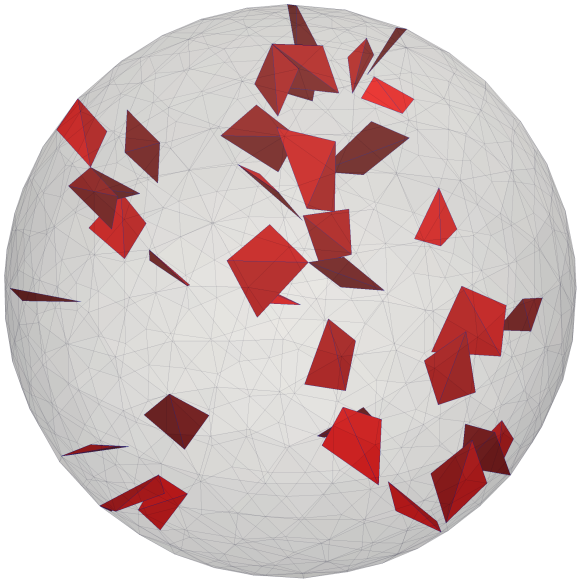


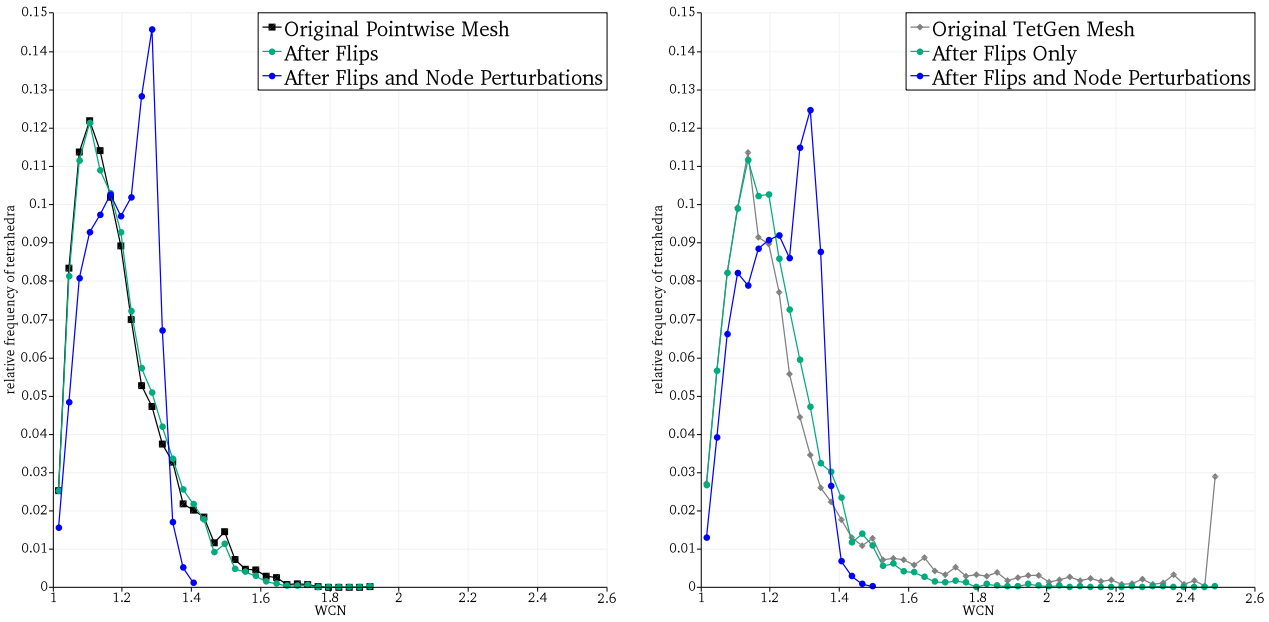
Figure 3.7 Simple Sphere Poor Quality Elements in TetGen Mesh

Table 3.1 Simple Sphere Pointwise and TetGen Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original Pointwise Grid: 5,508 tetrahedra, 1,196 nodes			
Weighted Condition Number	1.002080	1.085016	1.133701	1.162054	1.193901	1.287428	1.920569
Aspect Ratio	1.002549	1.101091	1.161922	1.194867	1.234125	1.345487	2.024688
				After Flips Only: 5,490 tetrahedra, 1,196 nodes			
Weighted Condition Number	1.002080	1.085564	1.136373	1.164958	1.196001	1.284117	1.920569
Aspect Ratio	1.002549	1.102337	1.163650	1.197530	1.237103	1.344637	2.154153
				After Flips and Point Perturbation: 5,489 tetrahedra, 1,196 nodes			
Weighted Condition Number	1.007541	1.108690	1.169153	1.198764	1.229639	1.276362	1.397497
Aspect Ratio	1.008416	1.129427	1.202376	1.238658	1.270282	1.323309	1.721800
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original TetGen Grid: 5,154 tetrahedra, 1,097 nodes			
Weighted Condition Number	1.001030	1.100670	1.157017	1.189563	1.225011	1.363782	6.242445
Aspect Ratio	1.001268	1.123973	1.196112	1.236434	1.287736	1.454263	6.333880
				After Flips Only: 4,883 tetrahedra, 1,097 nodes			
Weighted Condition Number	1.001030	1.100953	1.156915	1.186288	1.215249	1.300929	2.803747
Aspect Ratio	1.001268	1.124653	1.195577	1.234605	1.280088	1.395390	2.809321
				After Flips and Point Perturbation: 4,886 tetrahedra, 1,097 nodes			
Weighted Condition Number	1.002412	1.119992	1.191016	1.224359	1.257695	1.311589	1.482044
Aspect Ratio	1.003190	1.143769	1.230470	1.271725	1.308008	1.374352	1.915600

Table 3.2 Simple Sphere Tetmesh Grid Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
	Tetmesh Grid: 4,818 tetrahedra, 1,100 nodes						
Weighted Condition Number	1.005530	1.132415	1.198142	1.227794	1.255214	1.309516	1.592277
Aspect Ratio	1.007014	1.160040	1.241294	1.278358	1.310812	1.390752	2.148685



(a) Pointwise (b) TetGen

Figure 3.8 Simple Sphere Optimization Relative Frequency Plots

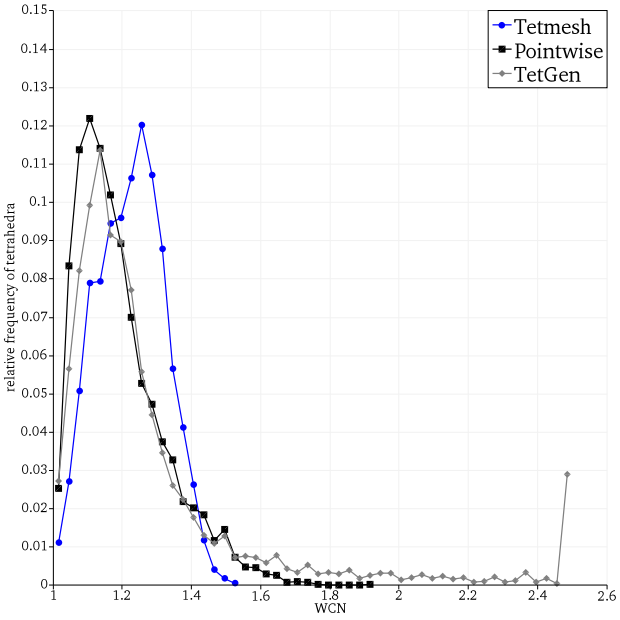


Figure 3.9 Simple Sphere Tetmesh vs. Pointwise and TetGen Relative Frequency Plot

3.4.2 Three Boxes Test Case

As another simple test case, an outer cube with two interior cubes with off-center positioning was constructed. This case is useful for demonstration since it has interior surfaces as do many practical grid generation problems. The surface mesh used here has 5,053 nodes and 10,094 triangles. Importantly, the inner cube discretization is much finer than the outer cube. That is, the spacings of the surface mesh are not roughly uniform as they were on the sphere case above. This surface spacing differential requires a spacing gradation in the tetrahedral mesh in order to have a quality grid. Otherwise, there will be large "jumps" in element size that are not conducive to good simulations (or element quality itself.) The surface geometry is depicted in Figure 3.10.

For this case, only a Pointwise generated grid has been optimized. Relative frequency plots of the Pointwise optimized mesh and the grid created by Tetmesh are given in Figures 3.11 and 3.12 respectively. The listing of grid statistics are to be found in Tables 3.3 and 3.4.

The results for pure optimization are very similar to those on the sphere case. The Pointwise mesh is of acceptable quality, but flipping and node smoothing significantly improve the grid. Topological changes by themselves do not have much effect.

Once again, the mesh creation scheme creates a grid comparable to that of Pointwise's with slightly higher quality. Figure 3.13 shows the octree pseudo-tiling tiling for this case. Note how it creates a simple gradation from the inner to the outer boundaries. The final meshes are depicted in Figures 3.14 and 3.15. These grids are obviously of high quality but note that one of the lower quality elements is on the boundary. This phenomenon will be examined later.

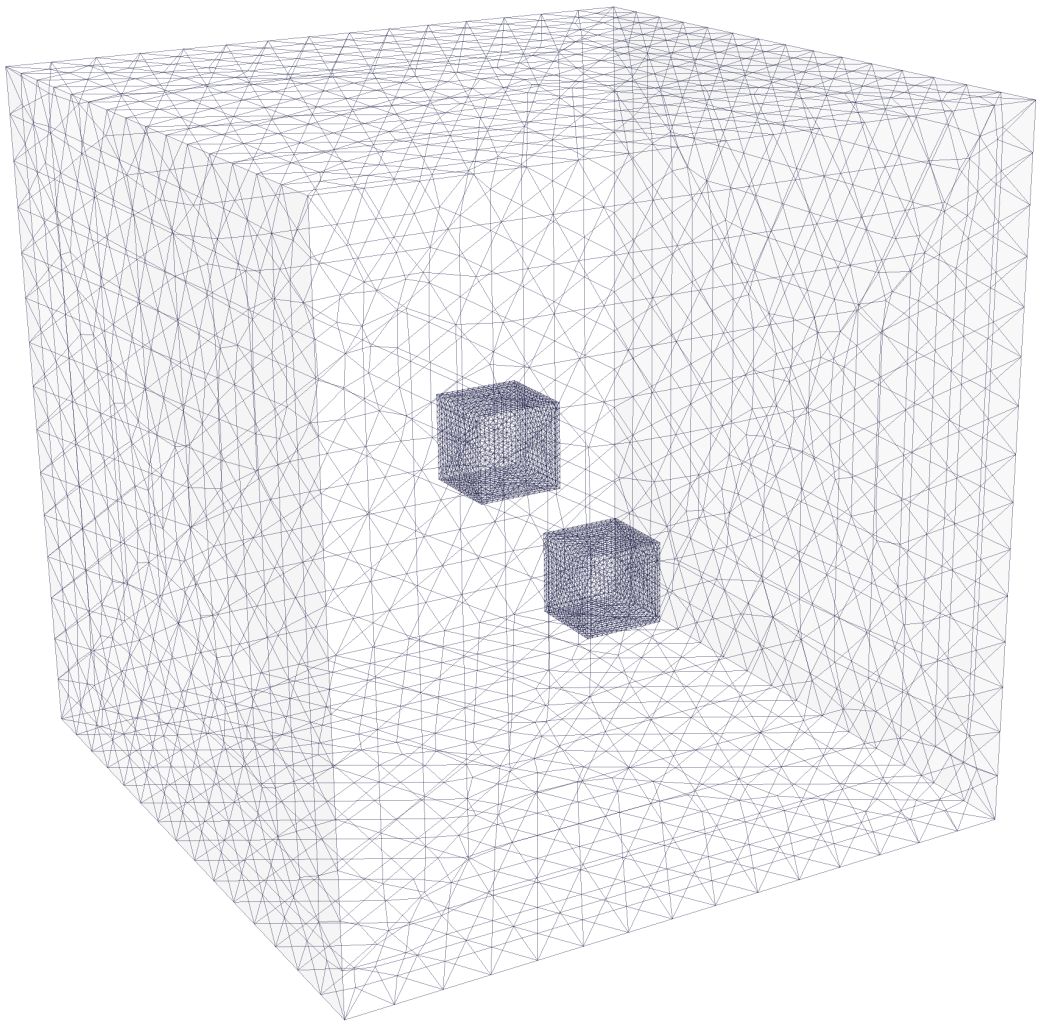


Figure 3.10 Three Boxes Case Geometry and Surface Grid

Table 3.3 Three Boxes Case Pointwise Mesh Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original Pointwise Grid: 140,266 tetrahedra, 25,822 nodes			
Weighted Condition Number	1.000005	1.089869	1.143163	1.172338	1.205842	1.301549	2.315684
Aspect Ratio	1.000006	1.107443	1.173024	1.208577	1.249300	1.366580	2.534831
				After Flips Only: 139,866 tetrahedra, 25,822 nodes			
Weighted Condition Number	1.000005	1.092028	1.146974	1.176985	1.209879	1.298306	2.315684
Aspect Ratio	1.000006	1.109605	1.177493	1.213688	1.254092	1.365022	2.681166
				After Flips and Point Perturbation: tetrahedra, 25,822nodes			
Weighted Condition Number	1.000380	1.119865	1.182970	1.213670	1.240652	1.279388	1.534027
Aspect Ratio	1.000435	1.142673	1.218070	1.250941	1.277307	1.330296	1.894667

Table 3.4 Three Boxes Case Tetmesh Grid Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Tetmesh Grid: 83,972 tetrahedra, 16,779 nodes			
Weighted Condition Number	1.000009	1.134308	1.198504	1.226480	1.254746	1.317876	1.952879
Aspect Ratio	1.000010	1.160623	1.237476	1.272555	1.308964	1.400661	3.170570

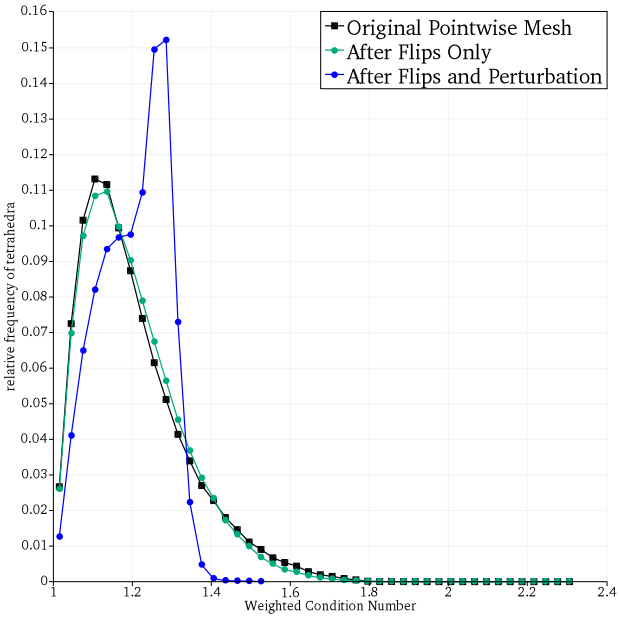


Figure 3.11 Three Boxes Case Optimization Optimization Relative Frequency Plots

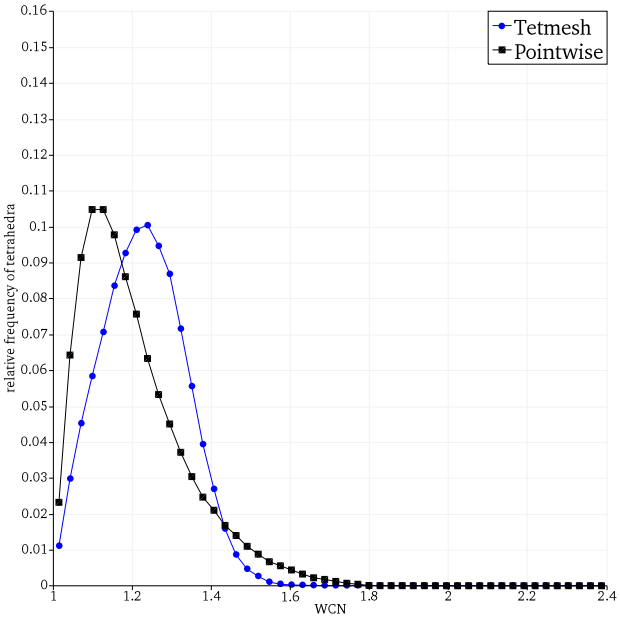


Figure 3.12 Three Boxes Case Tetmesh vs. Pointwise Generated Grid Relative Frequency Plots

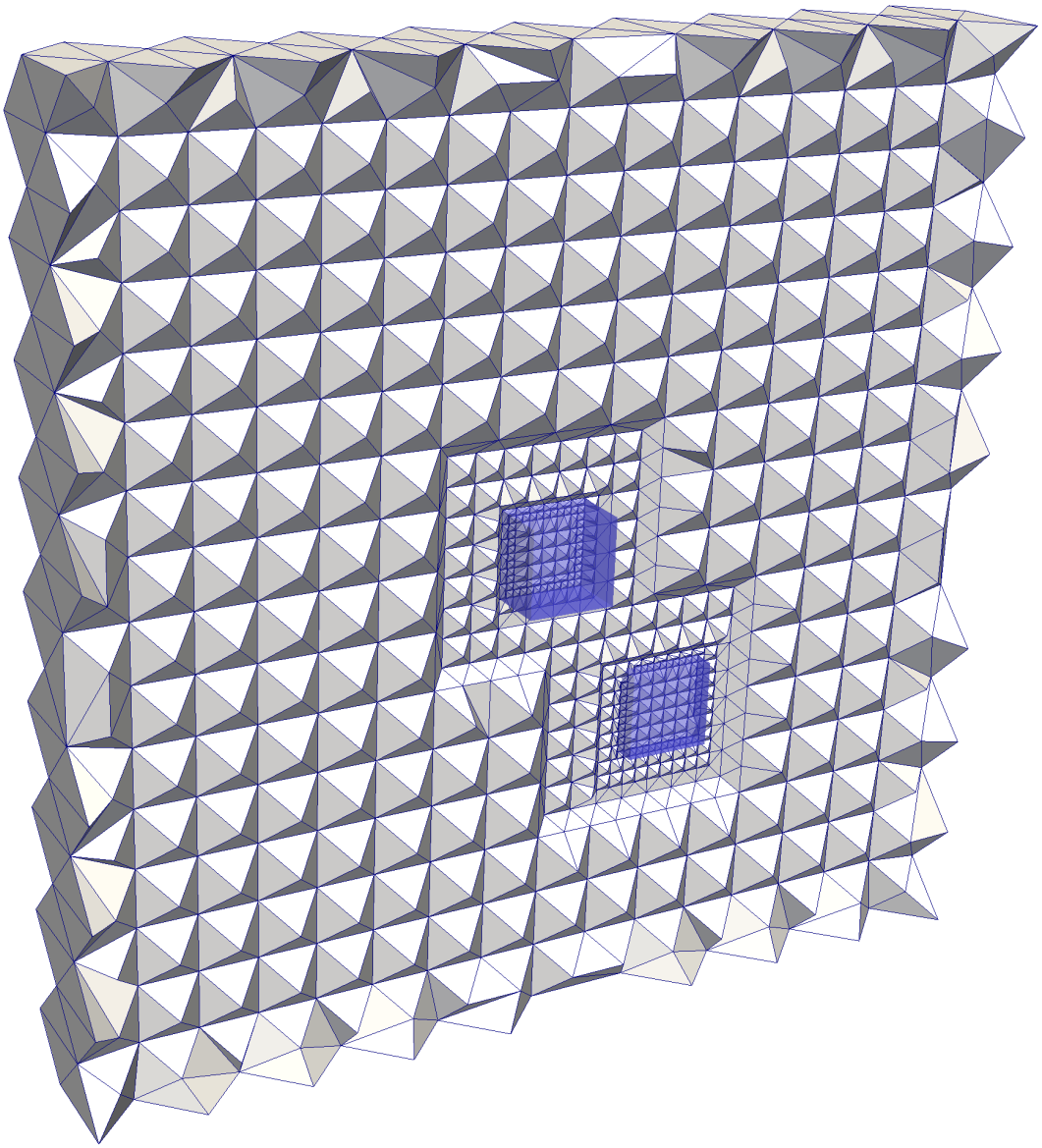


Figure 3.13 Three Boxes Case Octree Pseudo-Tiling

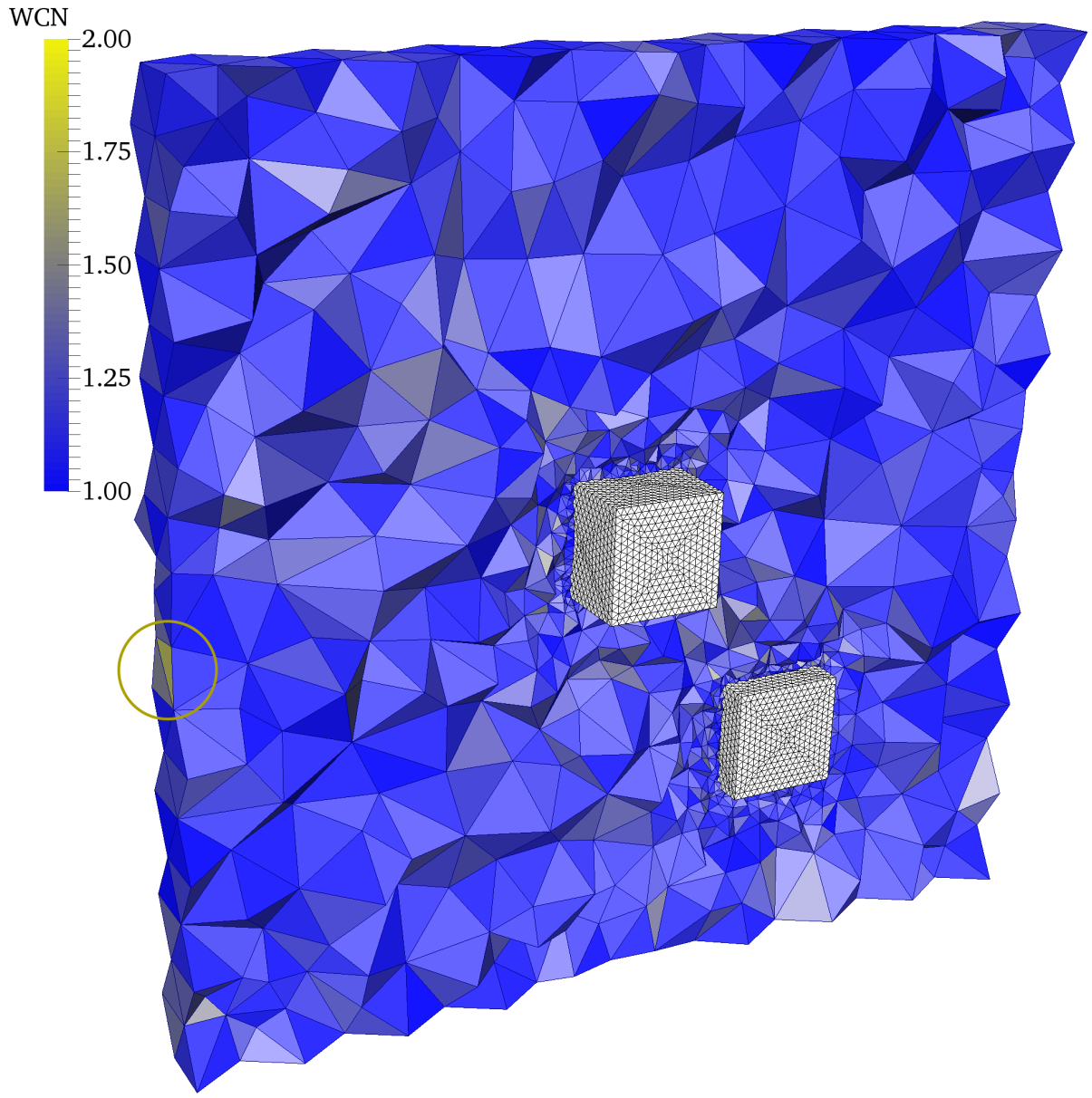


Figure 3.14 Three Boxes Case Tetmesh Grid with Callout of Poorer Quality Tetrahedron on Surface

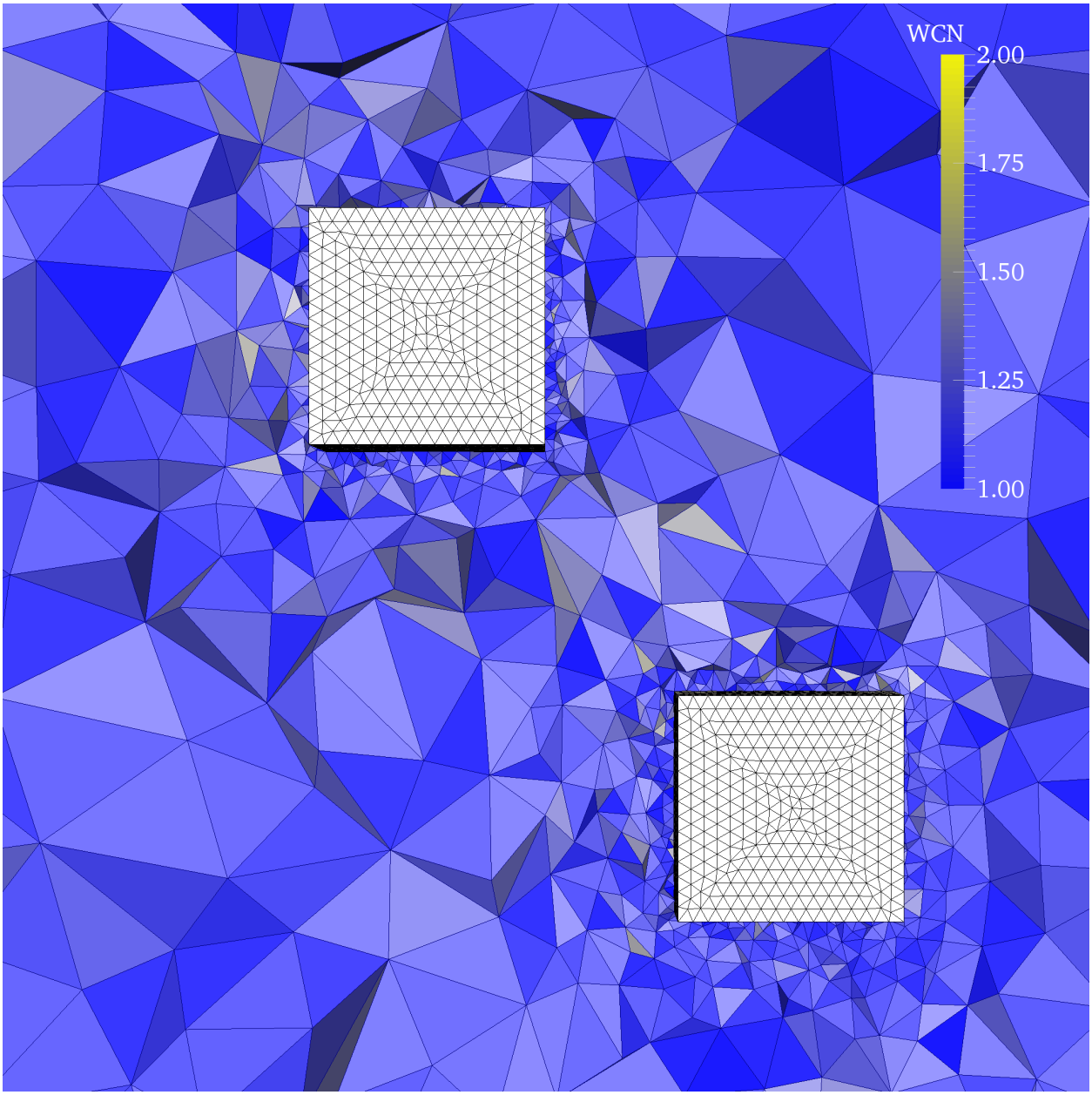


Figure 3.15 Three Boxes Case Tetmesh Grid (close-up)

3.4.3 ONERA M6 Wing

Of course, if the methodologies developed here are intended for practical mesh generation they should be generated on practical geometries. To that end, a mesh on NASA's ONERA M6 Wing geometry was constructed. The ONERA M6 is a standard CFD test case consisting of a particular wing mounted in a wind tunnel. Information about the test case can be found in [85].

The usual CFD geometry with the wing attached to a symmetry plane has been employed. The surface mesh, including the farfield and symmetry planes, consists of 42,766 nodes and 85,528 triangles. The wing's surface mesh, as well as part of the symmetry plane, is shown in Figure 3.16.

Both a Pointwise and TetGen generated grid have been optimized for this case. Relative frequency plots of the Pointwise and TetGen optimized meshes as well as the grid created by Tetmesh are given in Figures 3.17 and 3.18 . The listing of grid statistics are in in Tables 3.5 and 3.6.

Again we see the optimization methodologies are able to improve mesh quality in both the Pointwise and TetGen grids. A depiction of the several tetrahedra with WCN $\geq$ 2.0 is shown in Figure 3.19. Note all of the these lower quality tetrahedra have been optimized away.

The grid created solely in Tetmesh, while far superior to the TetGen grid, is of roughly the same quality as that of the Pointwise mesh. Figure 3.20 shows all the the elements with WCN $\geq$ 2.0 created by Tetmesh. Fortunately, they are all far away from the actual wing geometry. It is believed these poorer quality elements are caused by a flaw in the octree based pseudo-tiling initialization. Note the slices of the grid in Figure 3.22 shows a high quality mesh near the wing but Figure 3.21 shows elements that are slightly stretched. This abnormality was caused by the extents of the mesh being non-cubic and, thus, the octree used to create vertices for insertion was irregularly shaped.

It is believed this oversight led to the lower quality elements. Still, the overall mesh is acceptable and, again, comparable to the Pointwise grid. In any case, this problem could easily be corrected by simply force the bounding box to be cubic.

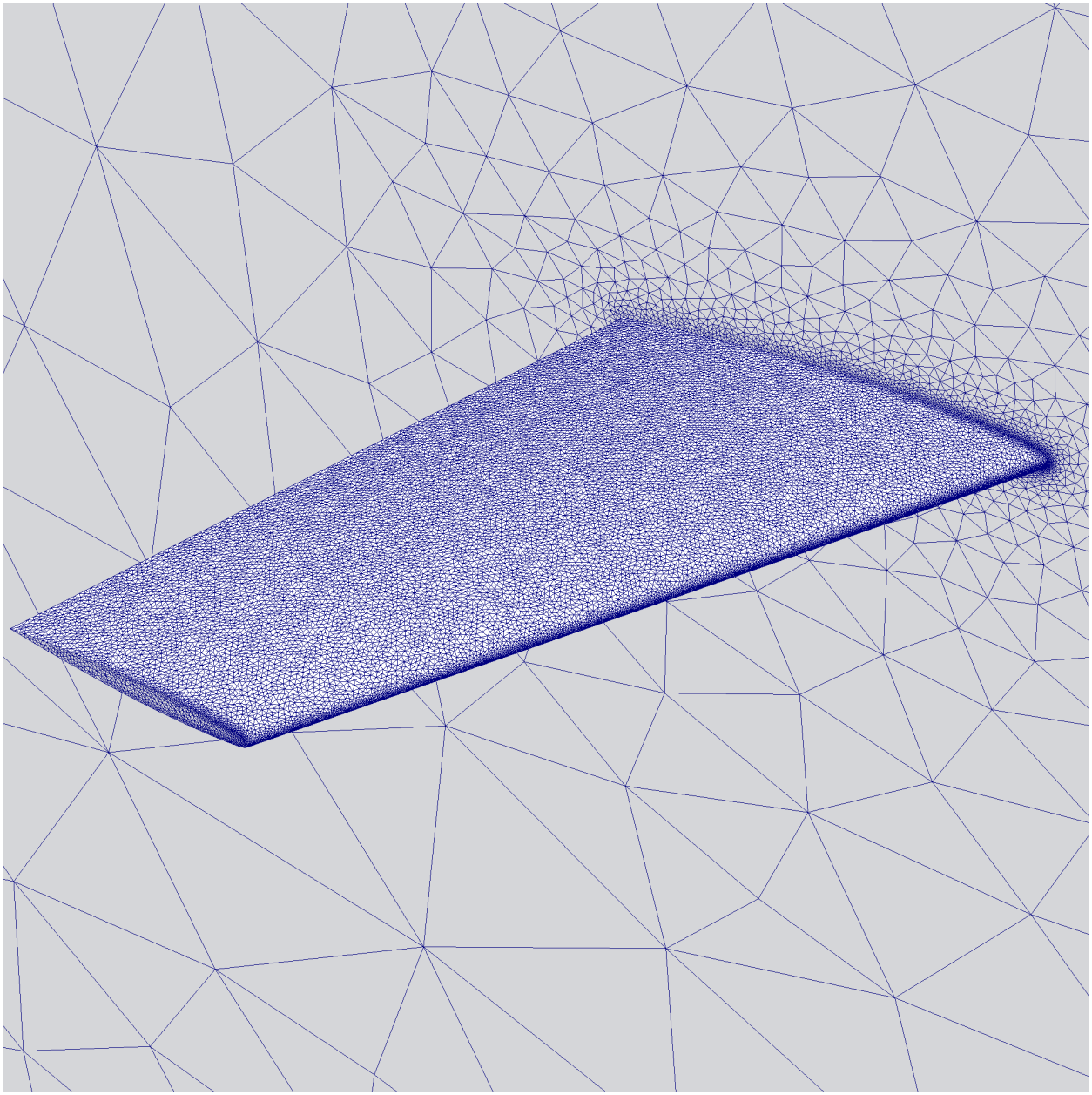


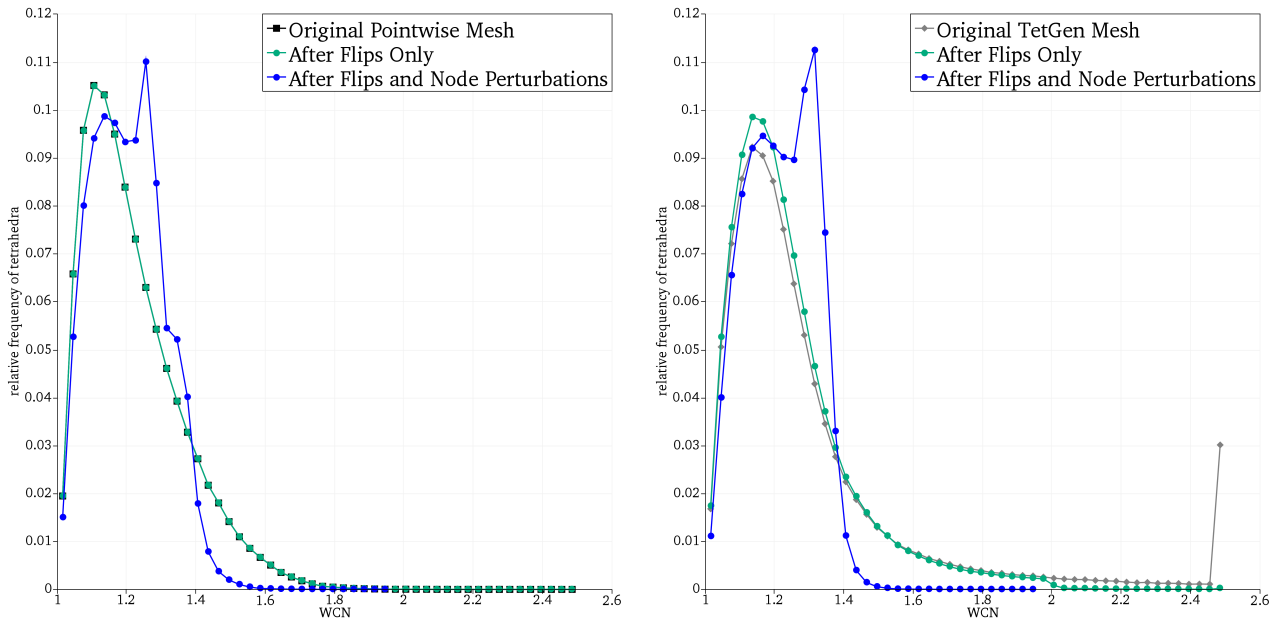
Figure 3.16 ONERA M6 Geometry and Surface/Symmetry Plane Grid

Table 3.5 ONERA M6 Pointwise and TetGen Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original Pointwise Grid: 606,222 tetrahedra, 123,261 nodes			
Weighted Condition Number	1.000550	1.095559	1.153315	1.185411	1.222607	1.326973	2.498528
Aspect Ratio	1.000673	1.115065	1.186290	1.226583	1.274023	1.409723	4.391156
				After Flips Only: 606,221 tetrahedra, 123,261 nodes			
Weighted Condition Number	1.000550	1.095560	1.153319	1.185417	1.222621	1.326993	2.278920
Aspect Ratio	1.000673	1.115066	1.186295	1.226589	1.274032	1.409752	4.022698
				After Flips and Point Perturbation: 605,302 tetrahedra, 123,261 nodes			
Weighted Condition Number	1.000414	1.107063	1.168144	1.199940	1.232332	1.291159	1.956504
Aspect Ratio	1.000499	1.128394	1.202912	1.240682	1.275086	1.360772	2.636634
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original TetGen Grid: 637,028 tetrahedra, 123,070 nodes			
Weighted Condition Number	1.000025	1.111545	1.177396	1.212631	1.254451	1.402932	6.691065
Aspect Ratio	1.000033	1.136131	1.221359	1.268818	1.324521	1.510058	7.740472
				After Flips Only: 611,860 tetrahedra, 123,070 nodes			
Weighted Condition Number	1.000025	1.108377	1.169882	1.201491	1.237525	1.345122	4.617606
Aspect Ratio	1.000033	1.132391	1.212036	1.255039	1.303673	1.447530	4.744747
				After Flips and Point Perturbation: 605,647 tetrahedra, 123,070 nodes			
Weighted Condition Number	1.000994	1.120304	1.184536	1.217089	1.250632	1.309661	1.956530
Aspect Ratio	1.001117	1.145157	1.225914	1.266191	1.304250	1.376819	3.552701

Table 3.6 ONERA M6 Tetmesh Grid Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Tetmesh Grid: 655,987 tetrahedra, 133,361 nodes			
Weighted Condition Number	1.001318	1.144199	1.214204	1.244265	1.272418	1.338183	2.457210
Aspect Ratio	1.001536	1.174092	1.256200	1.292197	1.330666	1.435704	4.518301



(a) Pointwise (b) TetGen

Figure 3.17 ONERA M6 Optimization Relative Frequency Plots

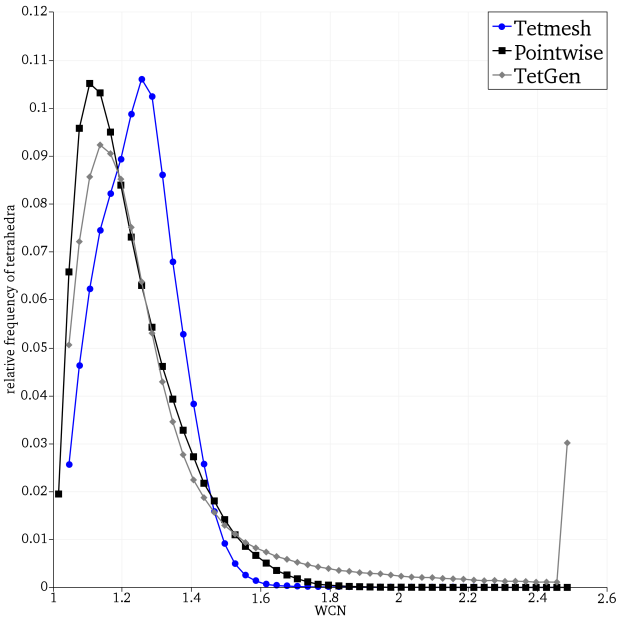
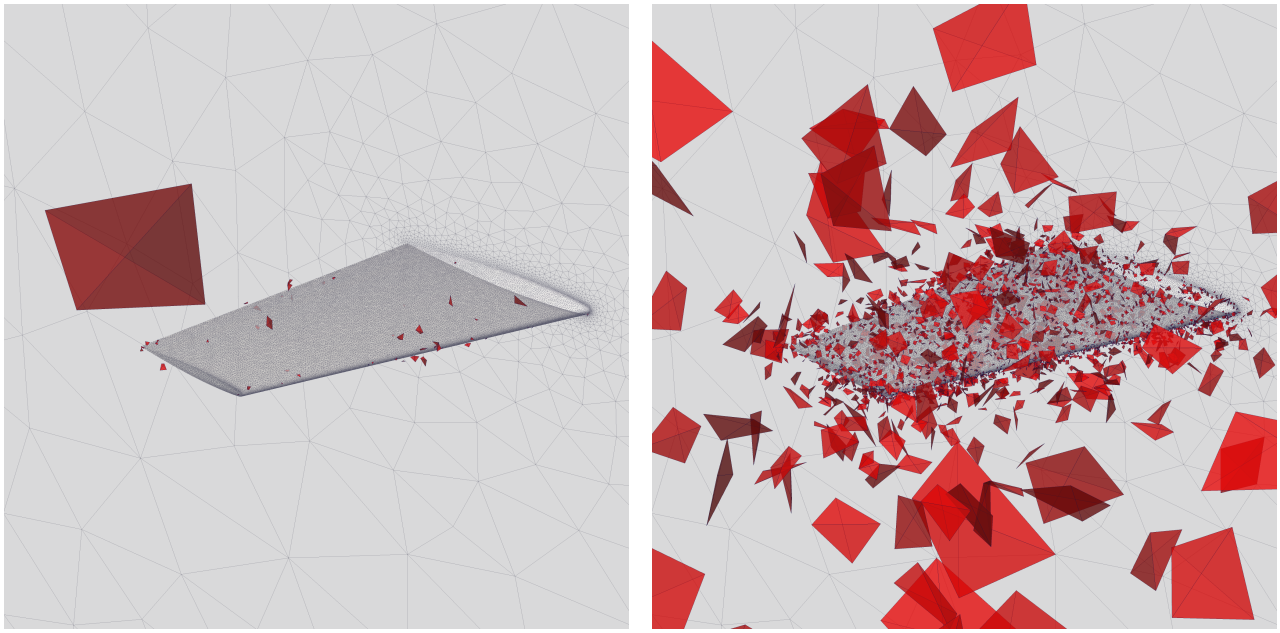


Figure 3.18 ONERA M6 Tetmesh vs. Pointwise Generated Grid Relative Frequency Plots



(a) Pointwise (b) TetGen

Figure 3.19 ONERA M6 Tetrahedra with WCN $\geq$ 2.0

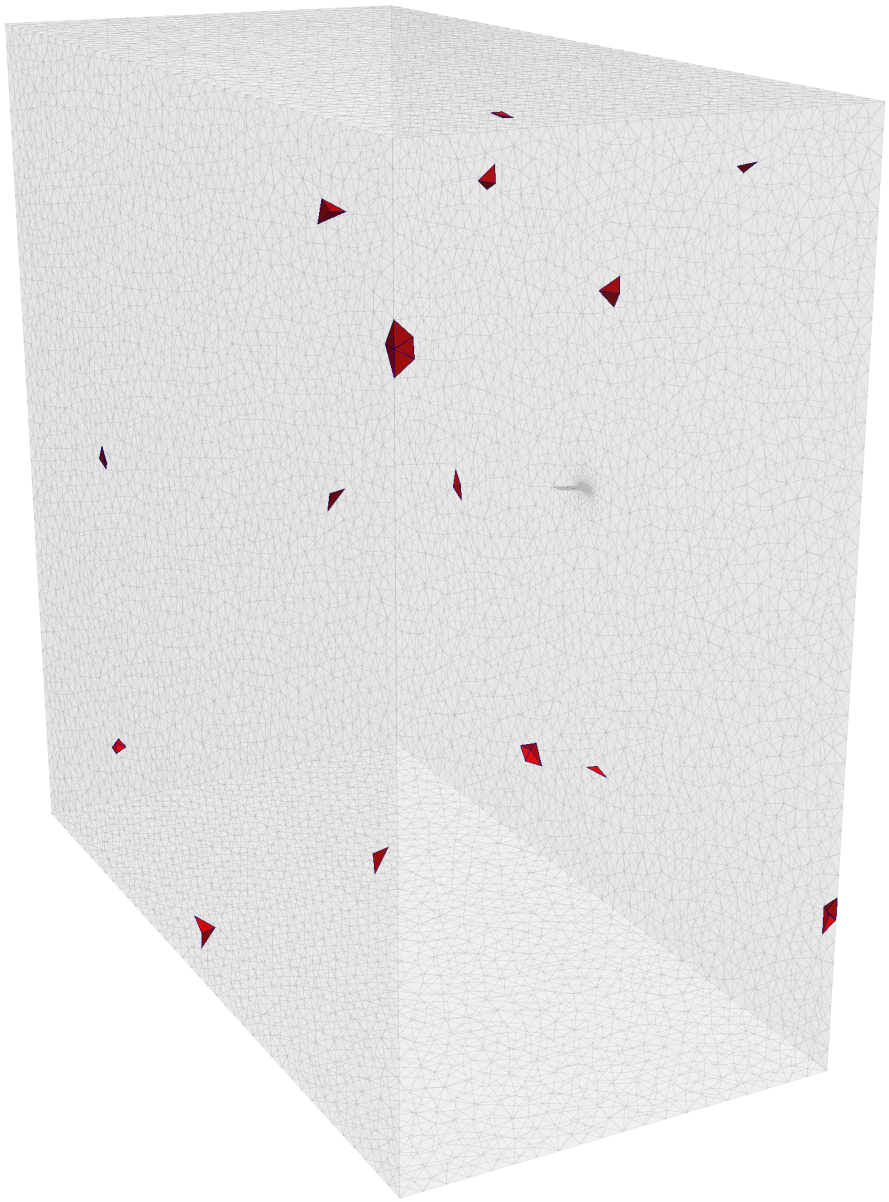


Figure 3.20 ONERA M6 Tetmesh Grid WCN $\geq$ 2.0

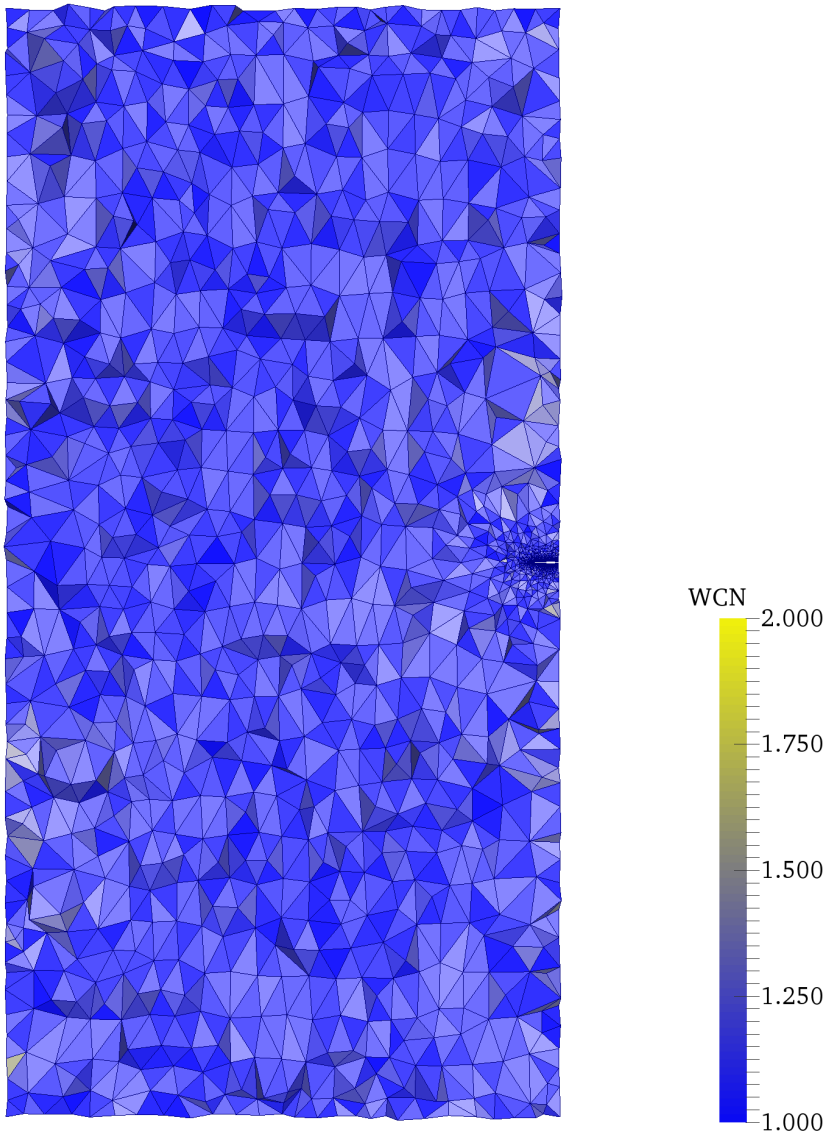


Figure 3.21 ONERA M6 Tetmesh Grid

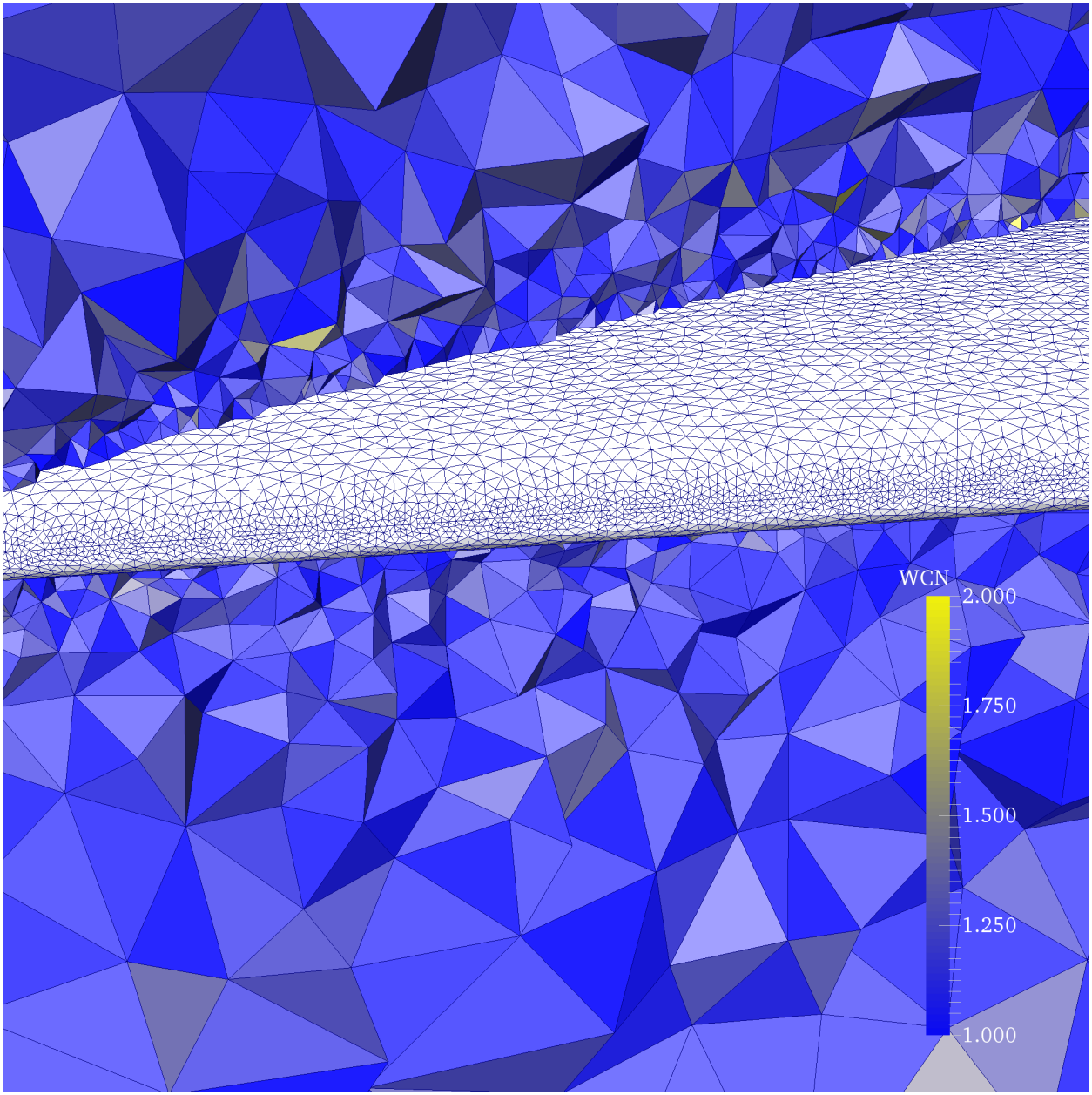


Figure 3.22 ONERA M6 Tetmesh Grid (close-up)

3.4.4 Notional Spaceplane

So far, none of the meshes generated or optimized are of production size. That is, they contain fewer nodes and elements than would be used in analysis of a typical geometry. For that matter, none of the geometries thus far have been particularly complex. So, as an example, a mesh on a notional spaceplane from popular culture has been optimized and generated. While an amusing exercise, this geometry has a couple of qualities that make it a good mesh generation test case. First, there is a great amount of detail in the surface geometry which makes both the surface meshing and boundary recovery challenging. Second, to capture all the geometric detail necessarily requires a large surface mesh.

The geometry for meshing consists of the spaceplane placed in a farfield. The surface mesh including the farfield has 257,440 nodes and 514,896 triangles. Figure 3.23 shows the spaceplane surface mesh along with a close-up of the parabolic aerial. These images are designed to give a sense of the amount of detail present in the surface.

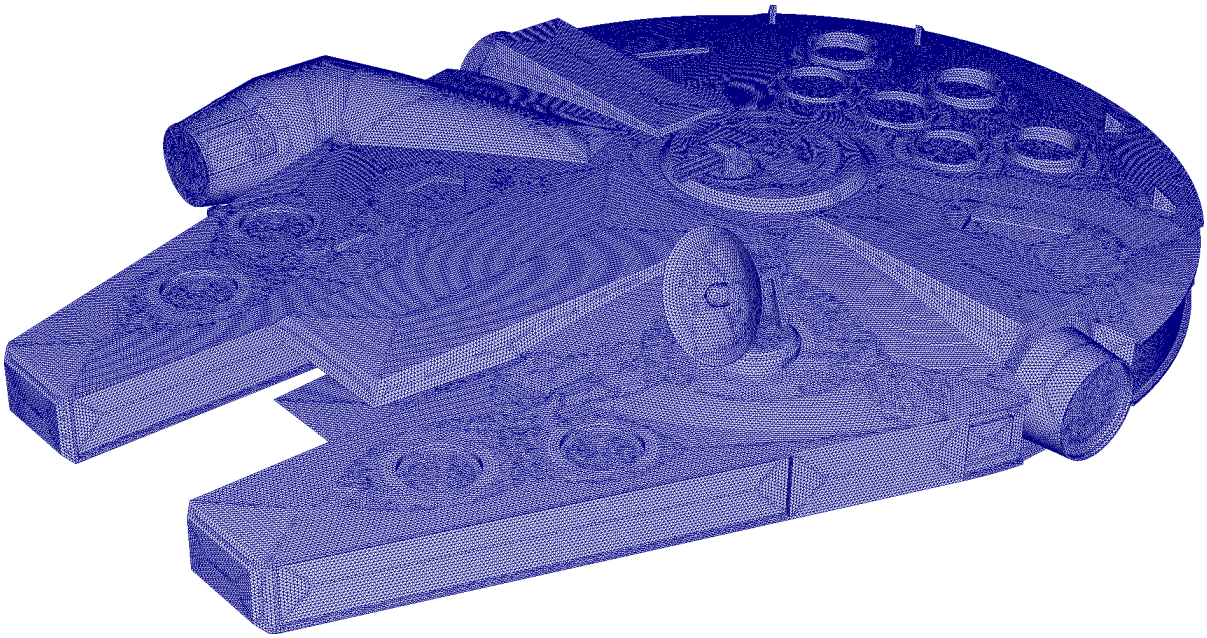
As above, relative frequency plots of the Pointwise optimized mesh and the grid created by Tetmesh are given in Figures 3.24 and 3.25 respectively. The listing of grid statistics are to be found in Tables 3.7 and 3.8. Note that the relative frequency plots do not seem to reflect the maximum WCN indicated in the grid statistics tables. This seeming discrepancy will be discussed below.

Despite the mesh size and complex geometry, results for this case are very similar to those of the cases presented so far, including the characteristic spike in the number of tetrahedra at WCN ≈ 1.3 in both the optimized Pointwise mesh and Tetmesh grid. Figure 3.27 shows a visual comparison between the Pointwise mesh the Tetmesh created grid. It is clear the elements in

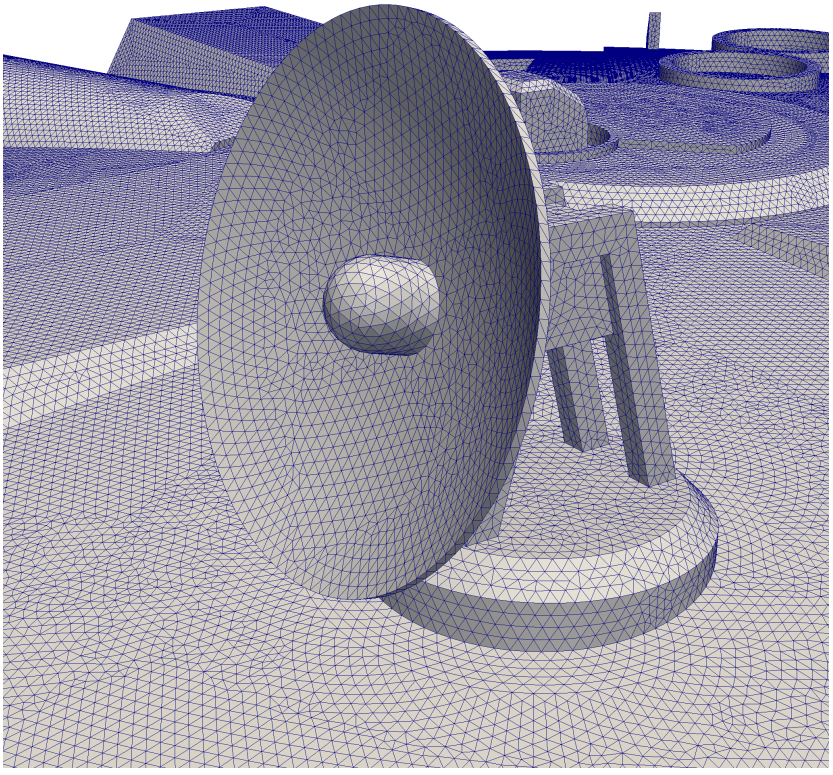
the Tetmesh grid grow larger more quickly as the distance from the surface increases than do the elements in the Pointwise grid.

Note, as indicated above, that the tables of statistics indicate the relative frequency plots are somewhat deceptive. There are, in fact, several tetrahedra in both grids that have WCN $\geq$2.0. There are so few of these elements, though, that they do not affect the character of the plot as they did in the several TetGen meshes presented above. These few poorer quality elements are actually a result of the surface triangulation. However, as can be seen in the visualization, the Pointwise grid has a large number of poorer quality elements away from the geometry.

Figure 3.26 shows a view of one such poor quality tetrahedra on the surface. This tetrahedra is connected to a poor quality surface triangle that arose from awkward intersection of two parts of the geometry. In fact, the depicted tetrahedra is composed of four points on the boundary and, thus, can not be improved through vertex smoothing as all four nodes are fixed. There are several other places in the surface triangulation with similar poor quality triangles. The lower quality elements in both the Pointwise and Tetmesh grids are a result of these sort of difficulties presented by the boundary triangulation.



(a) overview



(b) close-up of aerial

Figure 3.23 Notional Spaceplane Geometry and Surface Grid

Table 3.7 Notional Spaceplane Pointwise Mesh Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
	Original Pointwise Grid: 5,729,951 tetrahedra, 1,082,113 nodes						
Weighted Condition Number	1.000000	1.093555	1.148102	1.178196	1.213021	1.312953	5.418629
Aspect Ratio	1.000000	1.112464	1.180705	1.217464	1.260560	1.384427	6.133898
	After Flips Only: 5,729,909 tetrahedra, 1,082,113 nodes						
Weighted Condition Number	1.000000	1.093557	1.148107	1.178201	1.213028	1.312972	5.418629
Aspect Ratio	1.000000	1.112467	1.180709	1.217471	1.260569	1.384453	6.133898
	After Flips and Point Perturbation: 5,721,003 tetrahedra, 1,082,113 nodes						
Weighted Condition Number	1.000000	1.104396	1.162973	1.193079	1.223987	1.280723	5.418629
Aspect Ratio	1.000000	1.125463	1.196251	1.231485	1.265760	1.328695	7.111226

Table 3.8 Notional Spaceplane Tetmesh Grid Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
	Tetmesh Grid: 2,981,051 tetrahedra, 638,185 nodes						
Weighted Condition Number	1.000000	1.146312	1.224766	1.258686	1.290897	1.362112	7.292558
Aspect Ratio	1.000000	1.178182	1.274926	1.317717	1.361056	1.469953	24.853966

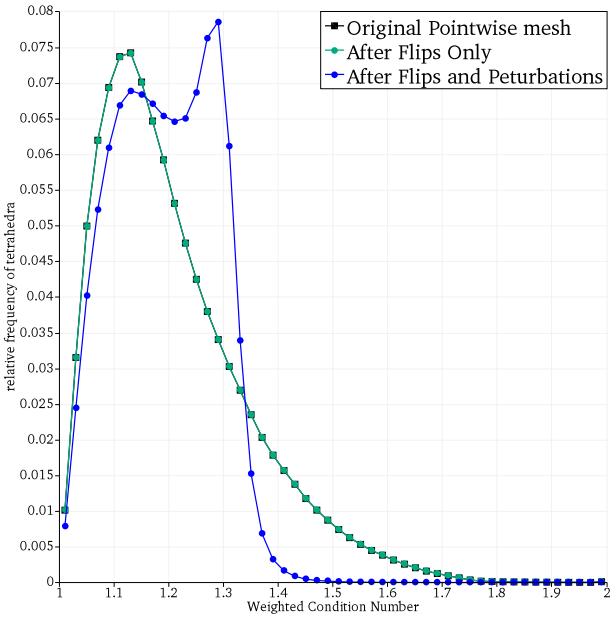


Figure 3.24 Notional Spaceplane Optimization of Pointwise Mesh Relative Frequency Plots

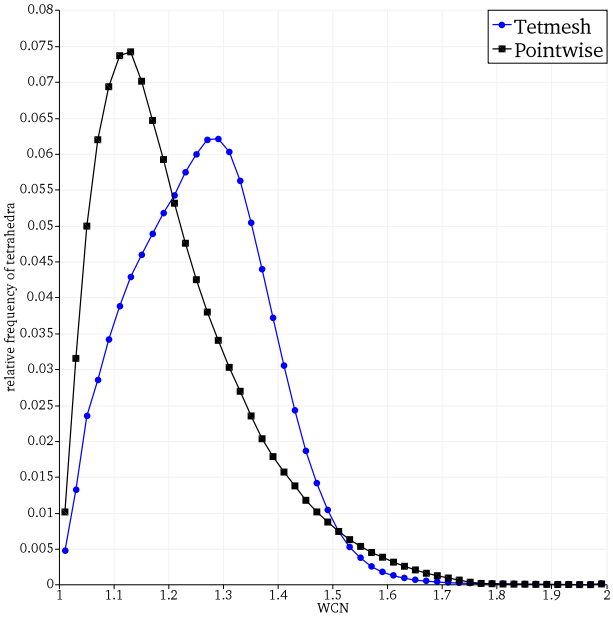
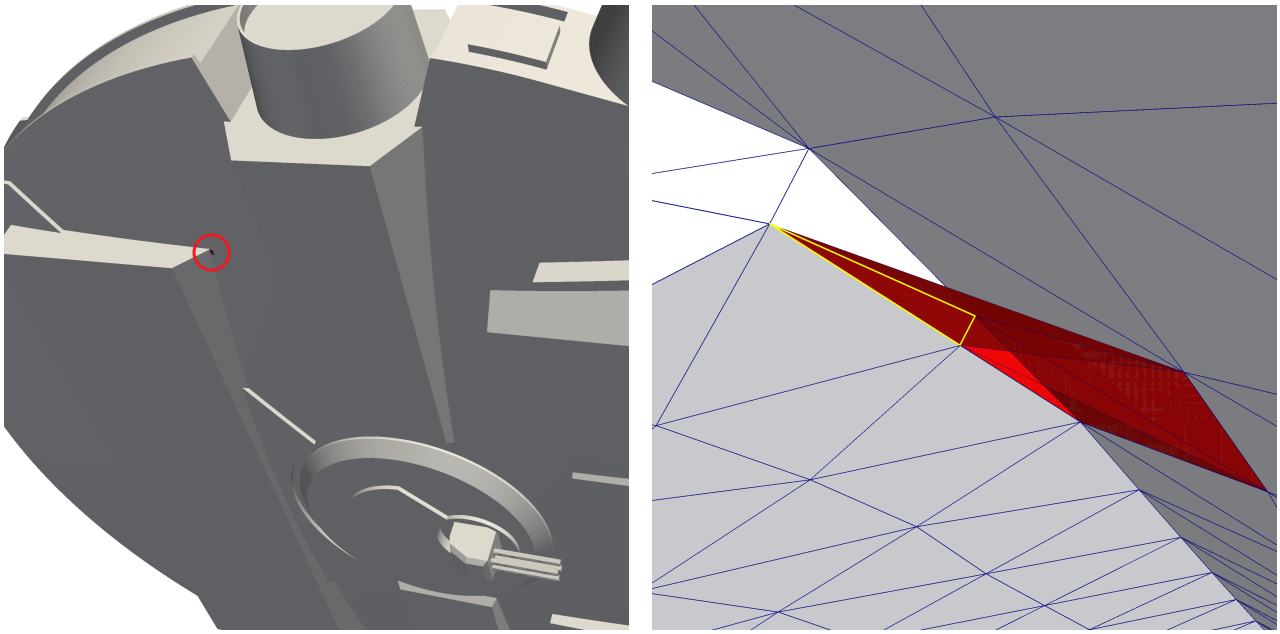
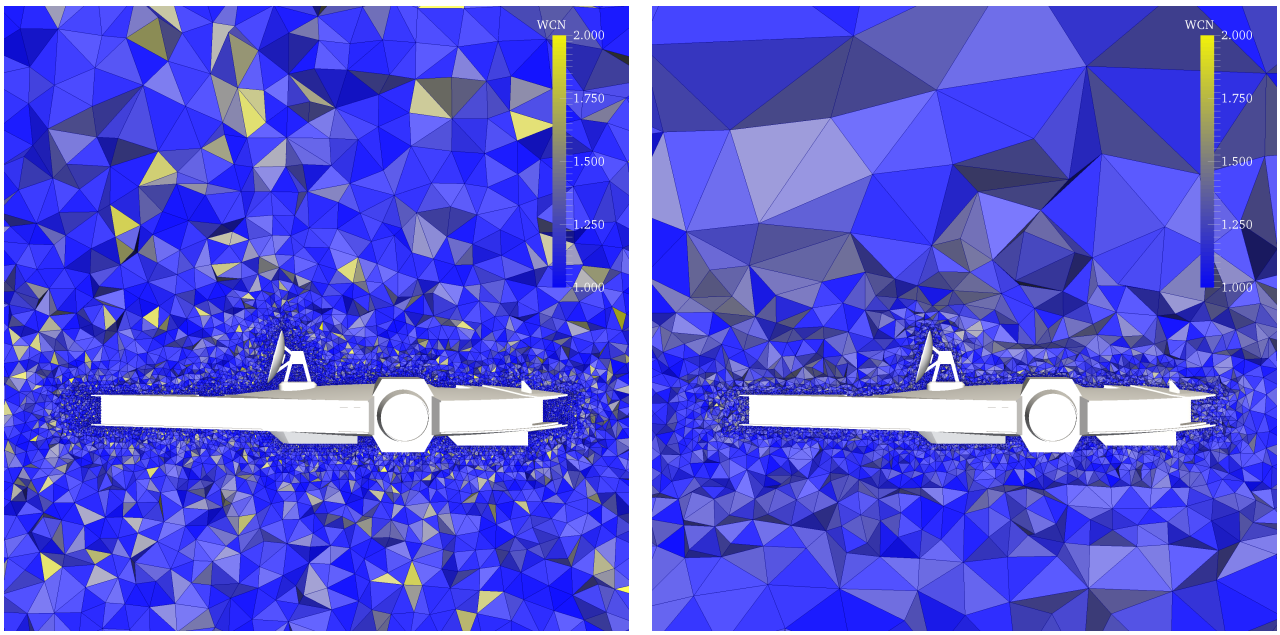


Figure 3.25 Notional Spaceplane Tetmesh versus Pointwise Relative Frequency Plots



(a) locating view (b) close up view

Figure 3.26 Notional Spaceplane Poor Quality Tetrahedra on Surface



(a) Pointwise (b) Tetmesh

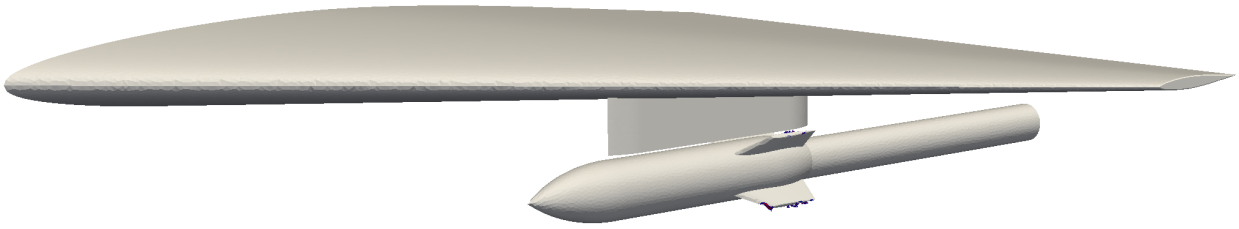
Figure 3.27 Notional Spaceplane Visual Comparison of Pointwise and Tetmesh Grids

3.4.5 Eglin Wing Pylon Store

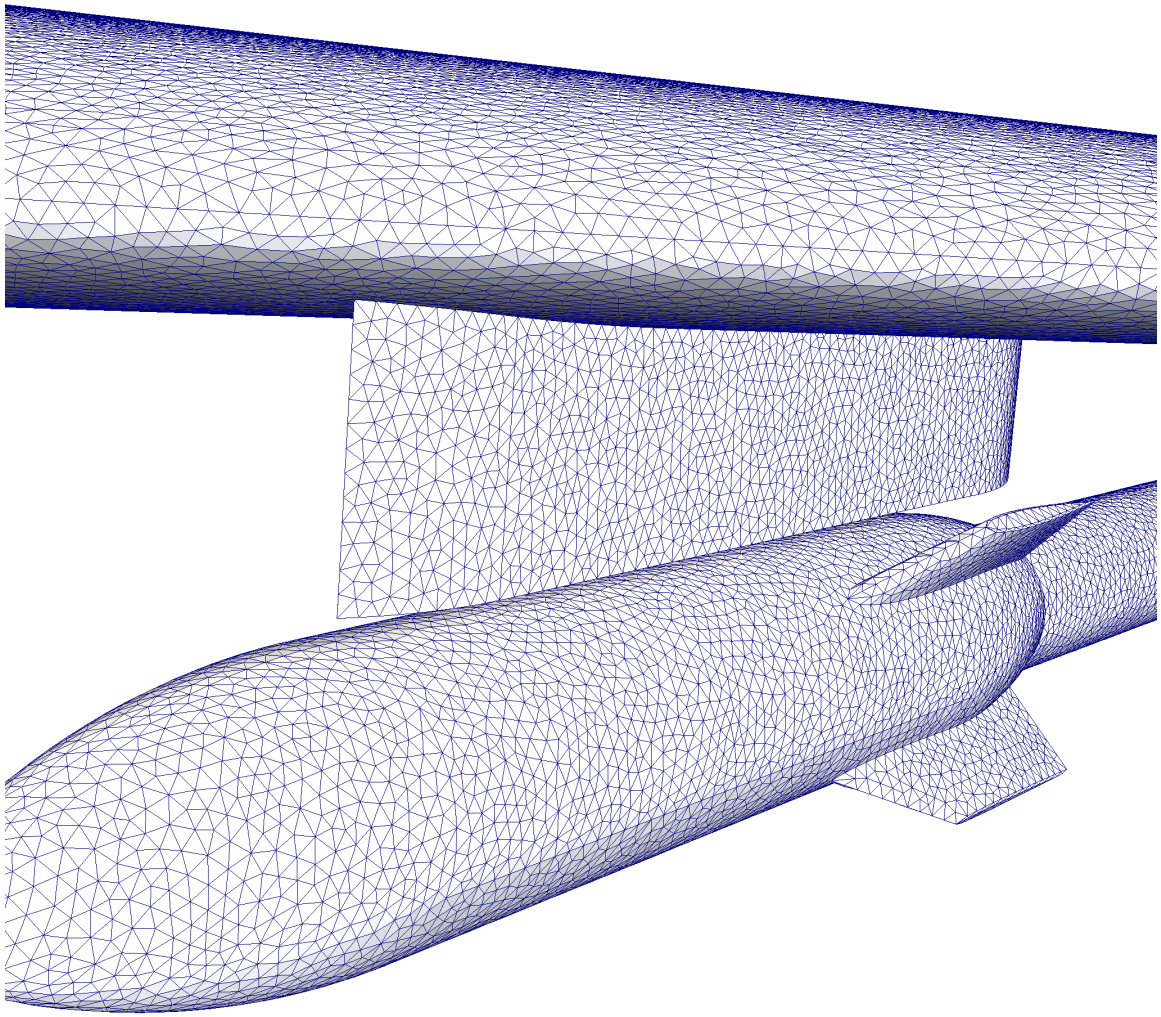
A more realistic example of a complex (but not quite as detailed) geometry is the Eglin Wing Pylon Store, named for Eglin Air Force Base near Pensacola, Florida. Wind tunnel tests were conducted on this configuration at Arnold Engineering Development Center (AEDC) in 1990 and provide high quality experimental results useful in CFD validation. Full details can be found in [86].

The geometry is a simple wing with a pylon attached to a symmetry plane with a separating store close under the pylon. The entire surface mesh used for this work has 52,720 nodes and 105,432 triangles. The geometry, along with a close-up of the surface mesh, is shown in Figure 3.28. Relative frequency plots of the Pointwise optimized mesh and the grid created by Tetmesh are given in Figures 3.29 and 3.30 respectively. The listing of grid statistics are to be found in Tables 3.9 and 3.10.

The results for this case are very similar to those of the notional spaceplane. Optimization of the Pointwise mesh and the grid created by Tetmesh follow all the patterns noted above. Figure 3.32 shows a visual comparison between the Pointwise and Tetmesh grids and, just as before, the Pointwise grid has better gradation in the size of the tetrahedra but a few scattered, poorer quality elements. Further, as before, there are a few poor quality elements not obvious from the frequency plots. Again, these are caused by a few suspect boundary triangles. Figure 3.31 shows just such a tetrahedra on the surface of one of the store fins. Note the very long, poor quality triangle.



(a) geometry excluding symmetry



(b) close-up of surface grid

Figure 3.28 Wing Pylon Store Geometry and Grid

Table 3.9 Wing Pylon Store Pointwise Mesh Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original Pointwise Grid: 789,057 tetrahedra, 158,478 nodes			
Weighted Condition Number	1.000275	1.095215	1.152815	1.184767	1.221854	1.325739	11.032357
Aspect Ratio	1.000354	1.114379	1.185359	1.225413	1.272464	1.406905	16.675972
				After Flips Only: 789,060 tetrahedra, 158,478 nodes			
Weighted Condition Number	1.000275	1.095221	1.152830	1.184785	1.221887	1.325791	11.032357
Aspect Ratio	1.000354	1.114386	1.185381	1.225439	1.272499	1.406965	16.675972
				After Flips and Point Perturbation: 787,548 tetrahedra, 158,478 nodes			
Weighted Condition Number	1.000459	1.110380	1.173361	1.205207	1.236903	1.295330	9.534389
Aspect Ratio	1.000512	1.132579	1.209290	1.246474	1.280882	1.360050	20.190287

Table 3.10 Wing Pylon Store Tetmesh Grid Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Tetmesh Grid: 643,971 tetrahedra, 136,467 nodes			
Weighted Condition Number	1.001496	1.154670	1.231885	1.265349	1.297091	1.365175	9.534387
Aspect Ratio	1.001837	1.188303	1.283355	1.325846	1.368312	1.473711	20.200915

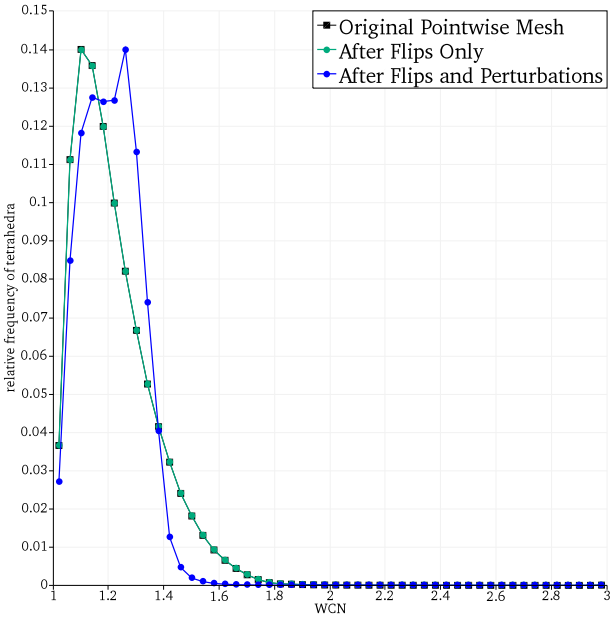


Figure 3.29 Wing Pylon Store Optimization of Pointwise Mesh Optimization Relative Frequency Plots

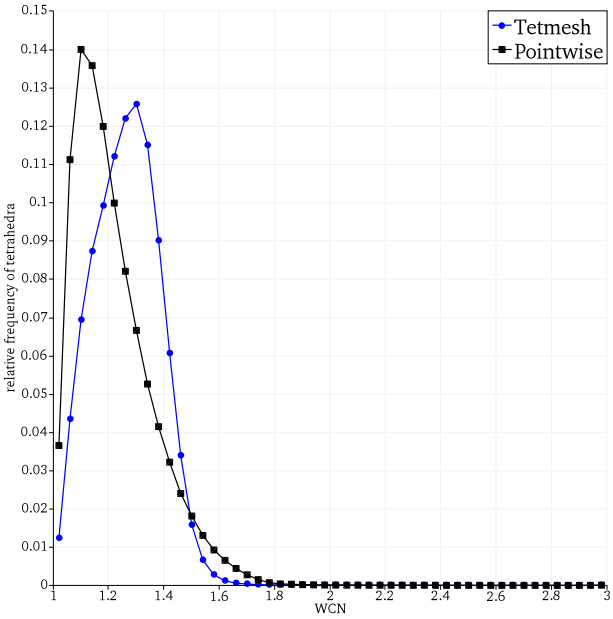


Figure 3.30 Wing Pylon Store Optimization of Pointwise Mesh

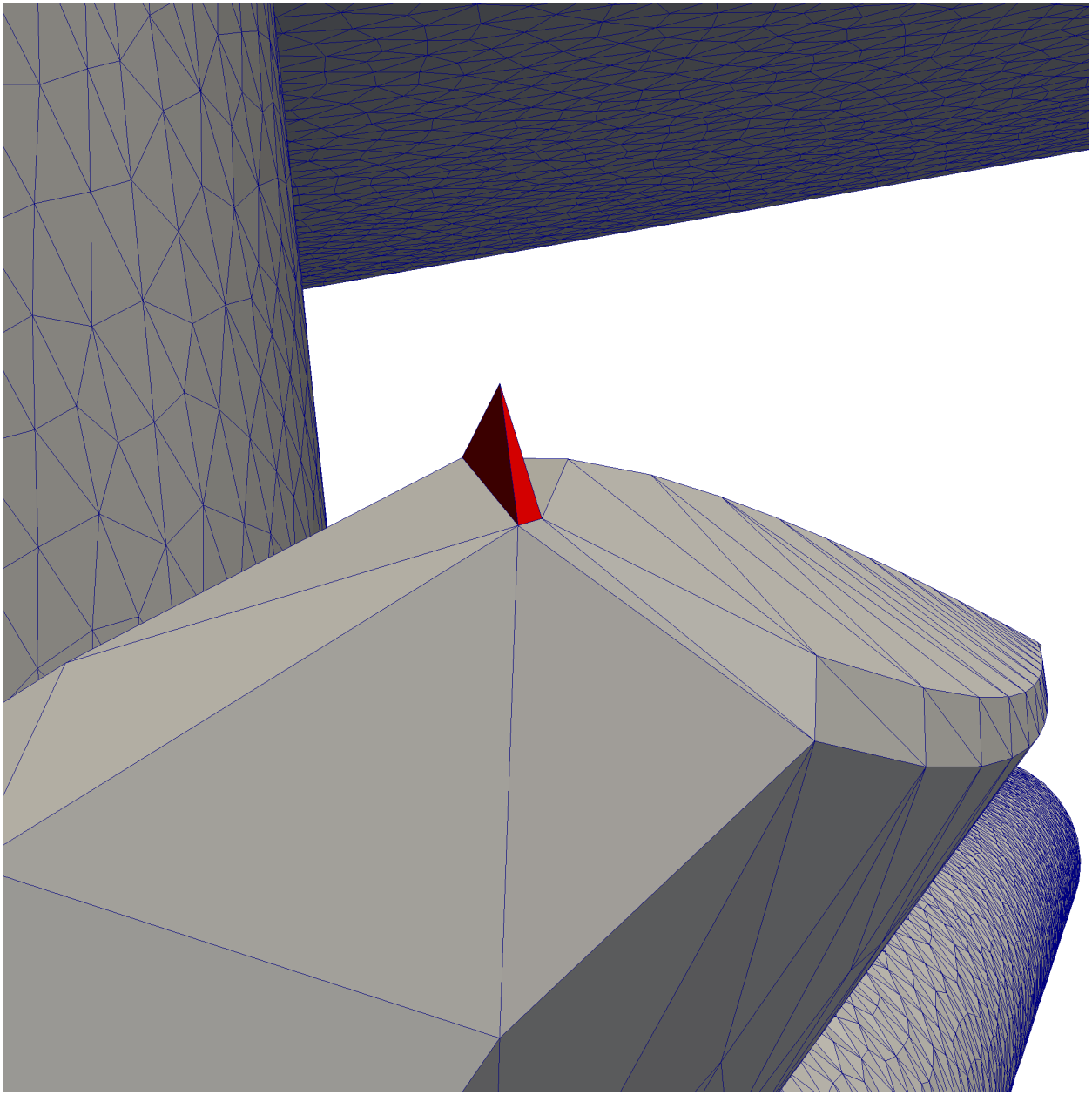
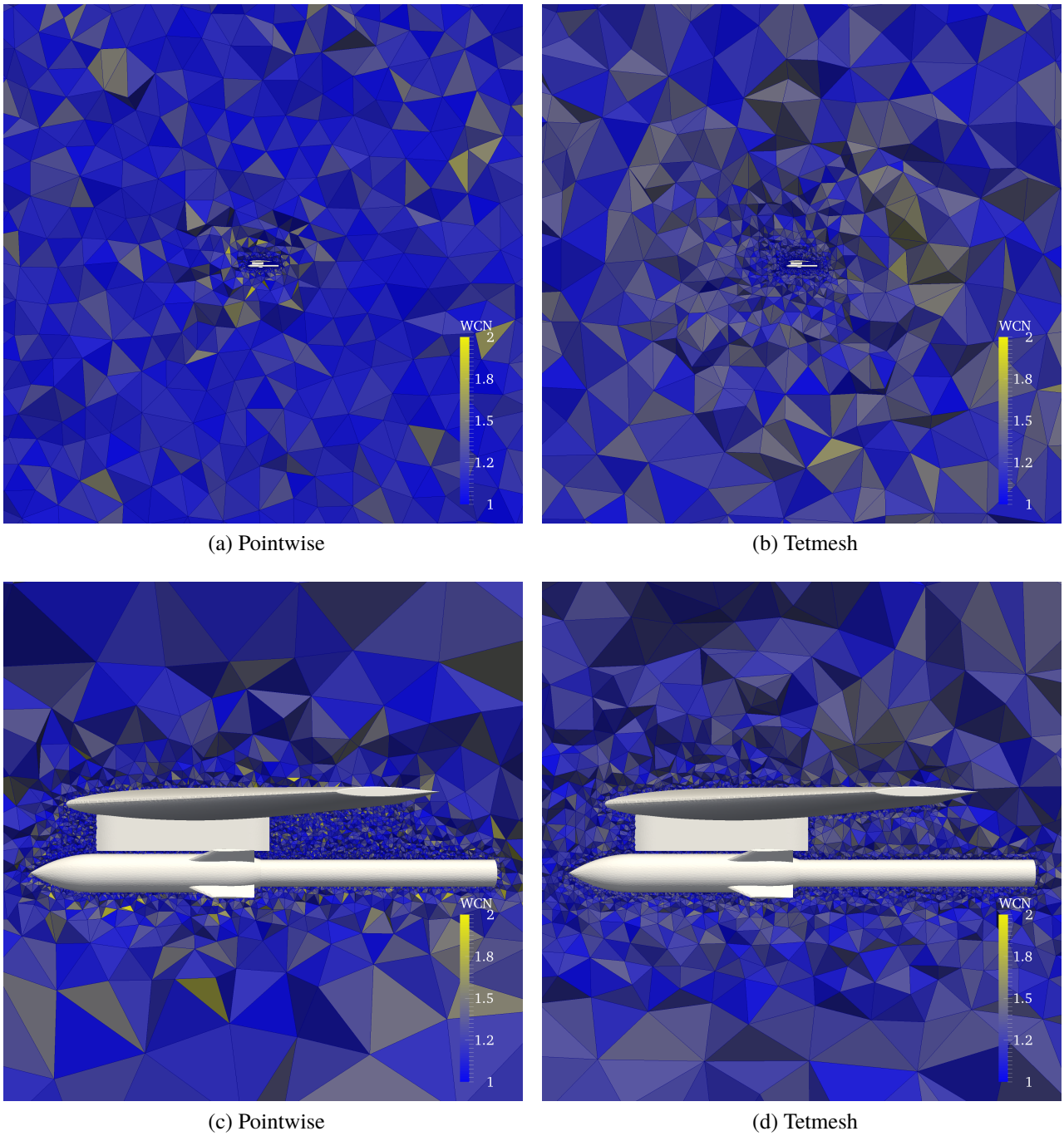


Figure 3.31 Wing Pylon Store Poor Quality Tetrahedron on Surface



(a) Pointwise

(b) Tetmesh

(c) Pointwise

(d) Tetmesh

Figure 3.32 Wing Pylon Store Visual Comparison of Pointwise and Tetmesh Grids

3.4.6 Stanford Bunny

To demonstrate the potential of changing boundary topology (i.e. boundary flips) like those described in Section 2.2.2, a case with an obviously poor surface mesh was chosen: the Stanford Bunny. The Stanford Bunny is an early example of three-dimensional scanning research conducted at Stanford University. The surface triangulation is poor because of the immature nature of the technology used to create it. Also, in fairness, this triangulation is not intended for computational meshing but rather for three-dimensional modeling and printing. The repository from which the model was taken as well as other information can be found in [87].

The original triangulation had several holes which for meshing purposes needed to be closed. So this triangulation was read into Pointwise and the holes were "patched." The final surface mesh has 8,294 nodes and 16,584 triangles and is shown in Figure 3.33. Relative frequency plots of the Pointwise optimized mesh and the grid created by Tetmesh are given in Figures 3.34 and 3.35. The listing of grid statistics are to be found in Tables 3.11 and 3.12. Note that the relative frequency plots and statistics tables include entries related to boundary flips.

As with the previous two cases, the relative frequency plots do not highlight the poor quality tetrahedra. Unsurprisingly, all these tetrahedra occur at the boundaries. Indeed, no small part of the surface triangulation is completely unacceptable for computational purposes. Figure 3.36 shows the worst tetrahedron in both the Pointwise and Tetmesh grids before any boundary flips are performed. A nearly "folded over" triangle is forcing an extremely flat tetrahedron into the final meshes. This configuration is unavoidable with the given surface mesh. However, boundary flips involving this tetrahedron substantially improve its quality when utilized both in the pure optimization process

and in Tetmesh grid creation. Note the changes to the surface highlighted in Figure 3.36d. This image shows the original mesh in blue and the mesh after boundary flips in red.

Overall, allowing boundary flips was able to substantially improve the quality of the mesh. However, there are still a few poor quality elements. Figure 3.37 shows a section to the Tetmesh generated grid. As can be seen, there are still areas of the mesh where the boundary has subtended some poorer quality elements. Alas, only so much can be done with a surface mesh of such low quality.

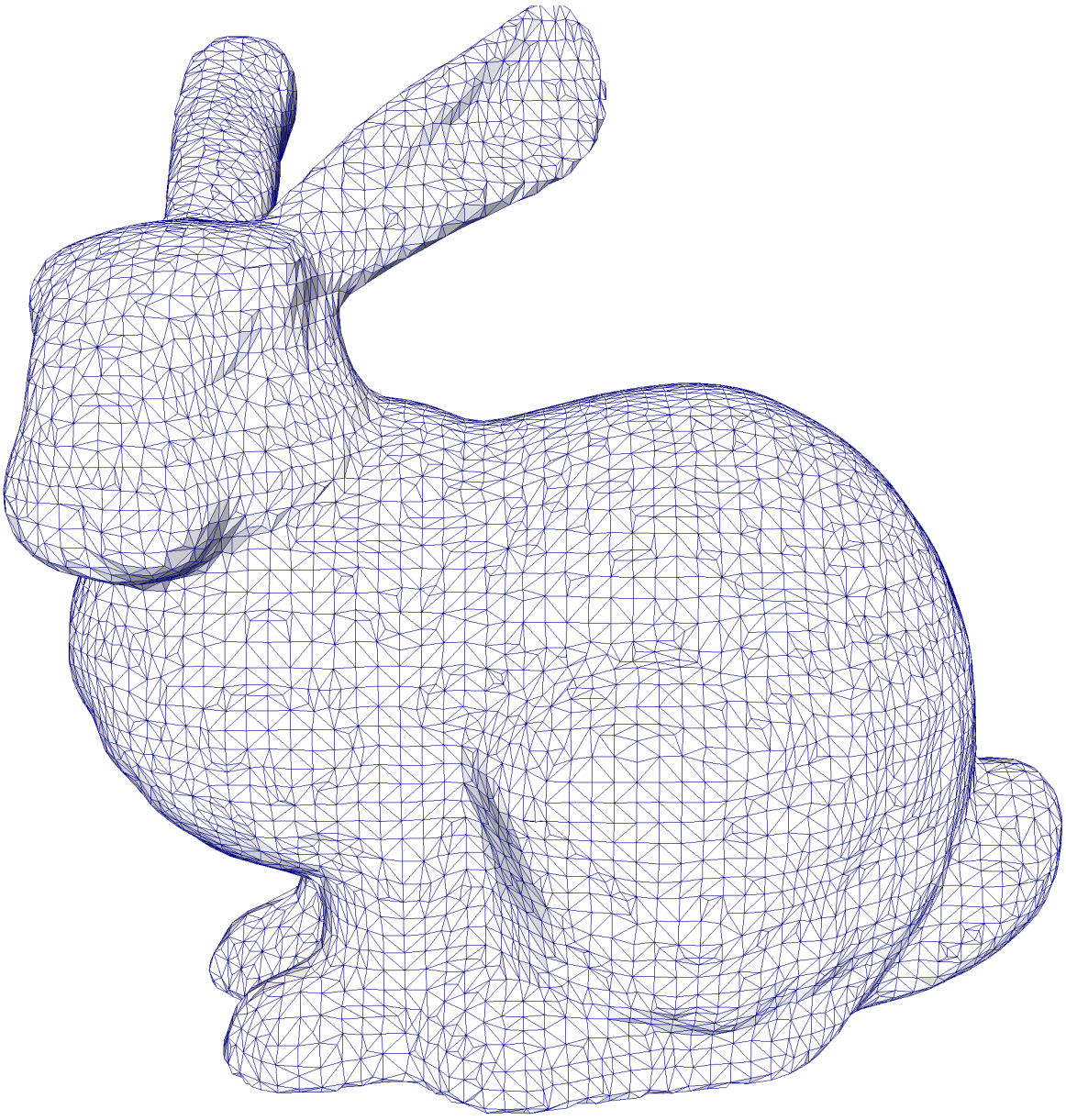


Figure 3.33 Stanford Bunny Geometry and Surface Grid

Table 3.11 Stanford Bunny Optimization Statistics

	quality measure percentiles						
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Original Pointwise Grid: 137,527 tetrahedra, 27,098 nodes			
Weighted Condition Number	1.001123	1.098790	1.159626	1.194365	1.234985	1.359171	58.741372
Aspect Ratio	1.001290	1.119371	1.194297	1.237182	1.287697	1.442231	1316.046388
				After Flips Only: 137,464 tetrahedra, 27,098 nodes			
Weighted Condition Number	1.001123	1.098918	1.159861	1.194567	1.235224	1.359602	58.741372
Aspect Ratio	1.001290	1.119551	1.194510	1.237498	1.288092	1.442695	1316.046388
				After Flips and Point Perturbation: 136,761 tetrahedra, 27,098 nodes			
Weighted Condition Number	1.000326	1.139619	1.215376	1.251120	1.284664	1.359991	58.741372
Aspect Ratio	1.000366	1.168482	1.258725	1.300135	1.341175	1.445675	1316.046388
				After Boundary Flips and Point Perturbation: 137,041 tetrahedra, 27,098 nodes			
Weighted Condition Number	1.001109	1.133389	1.202933	1.235435	1.267336	1.330956	23.106222
Aspect Ratio	1.001348	1.160619	1.243168	1.280606	1.317414	1.402322	299.301891

Table 3.12 Stanford Bunny Tetmesh Grid Statistics

					quality measure percentiles		
	min	*20th*	*40th*	*median*	*60th*	*80th*	*max*
				Tetmesh Grid, no boundary flips: 92,331 tetrahedra, 20,209 nodes			
Weighted Condition Number	1.002344	1.172427	1.262107	1.307073	1.356167	1.473983	58.741372
Aspect Ratio	1.002561	1.209466	1.323232	1.386378	1.454321	1.640034	1316.046388
				Tetmesh Grid, with boundary flips: 92,977 tetrahedra, 20,209 nodes			
Weighted Condition Number	1.001245	1.165223	1.248545	1.289205	1.330623	1.427518	23.106222
Aspect Ratio	1.001534	1.199754	1.305167	1.358384	1.415801	1.562850	299.301891

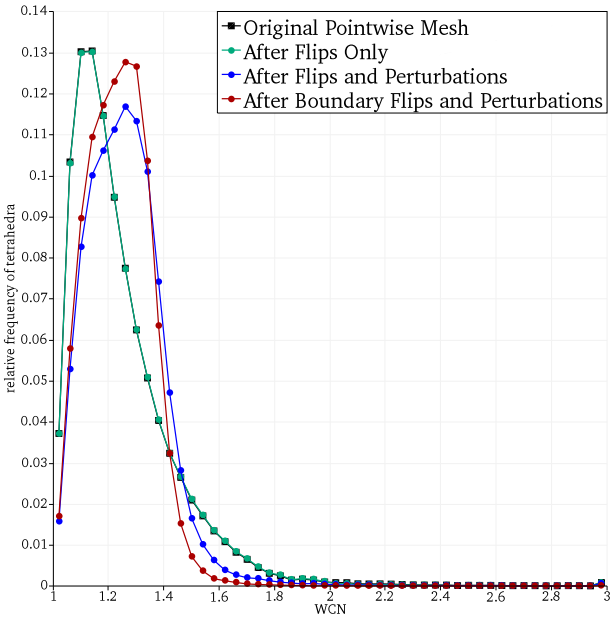


Figure 3.34 Stanford Bunny Optimization of Pointwise Mesh Relative Frequency Plots

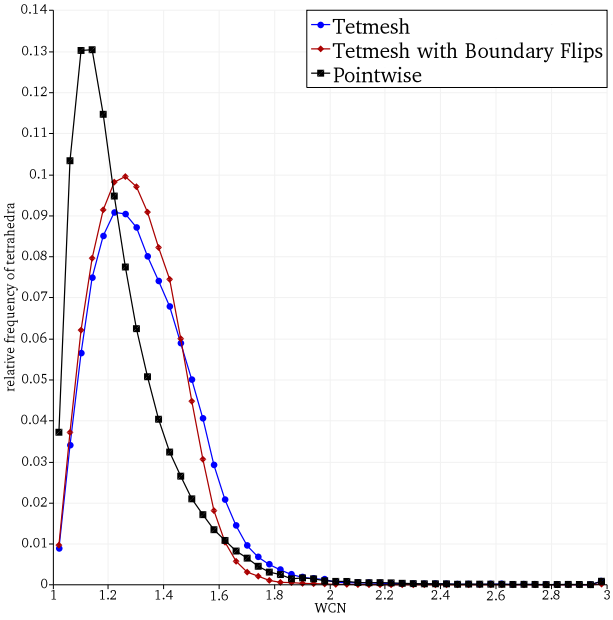
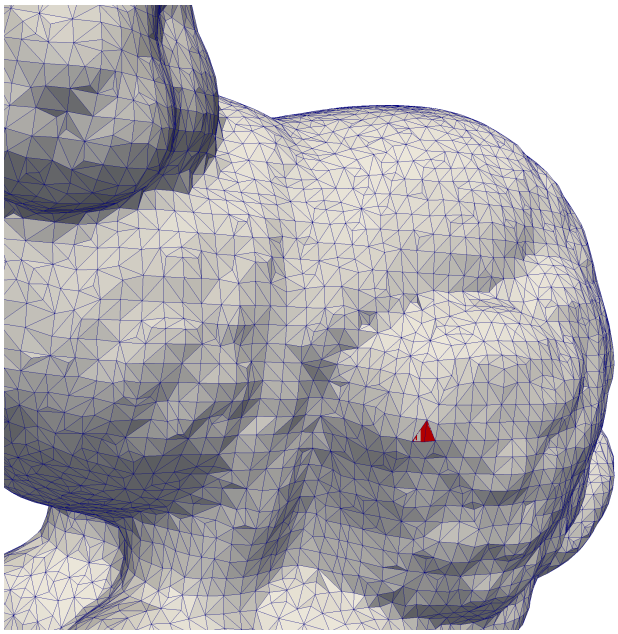
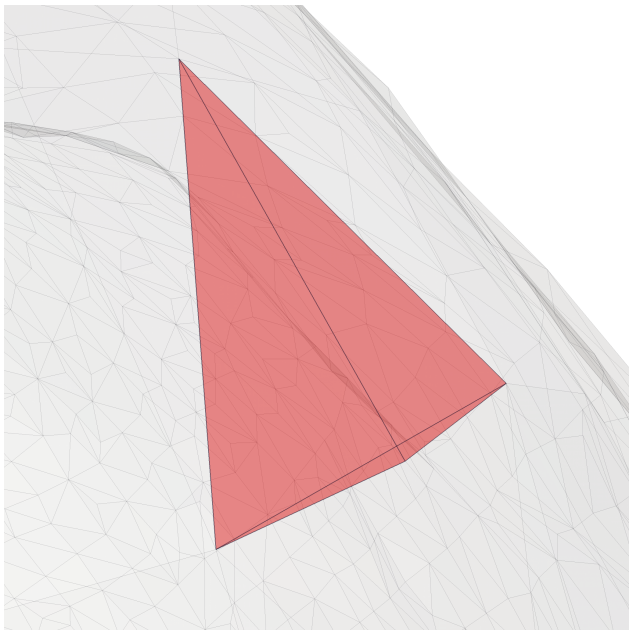


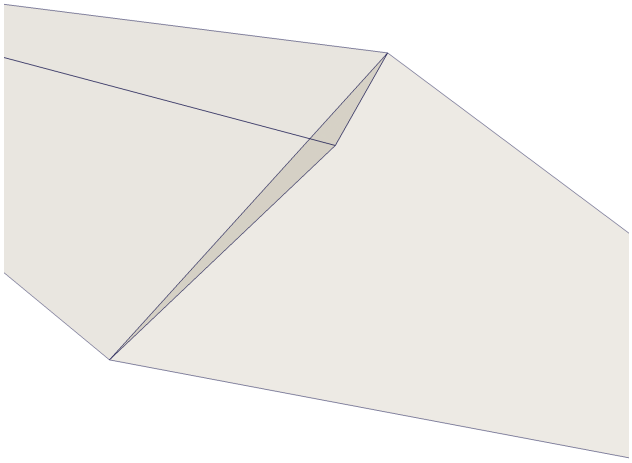
Figure 3.35 Stanford Bunny Tetmesh versus Pointwise Relative Frequency Plots



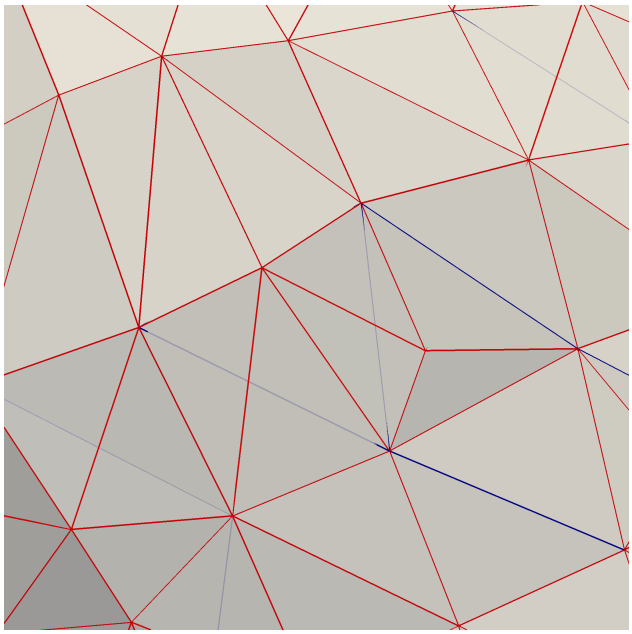
(a) locating view



(b) close up view



(c) poor quality surface mesh



(d) changes to surface that improve quality

Figure 3.36 Stanford Bunny Poor Quality Tetrahedra on Surface

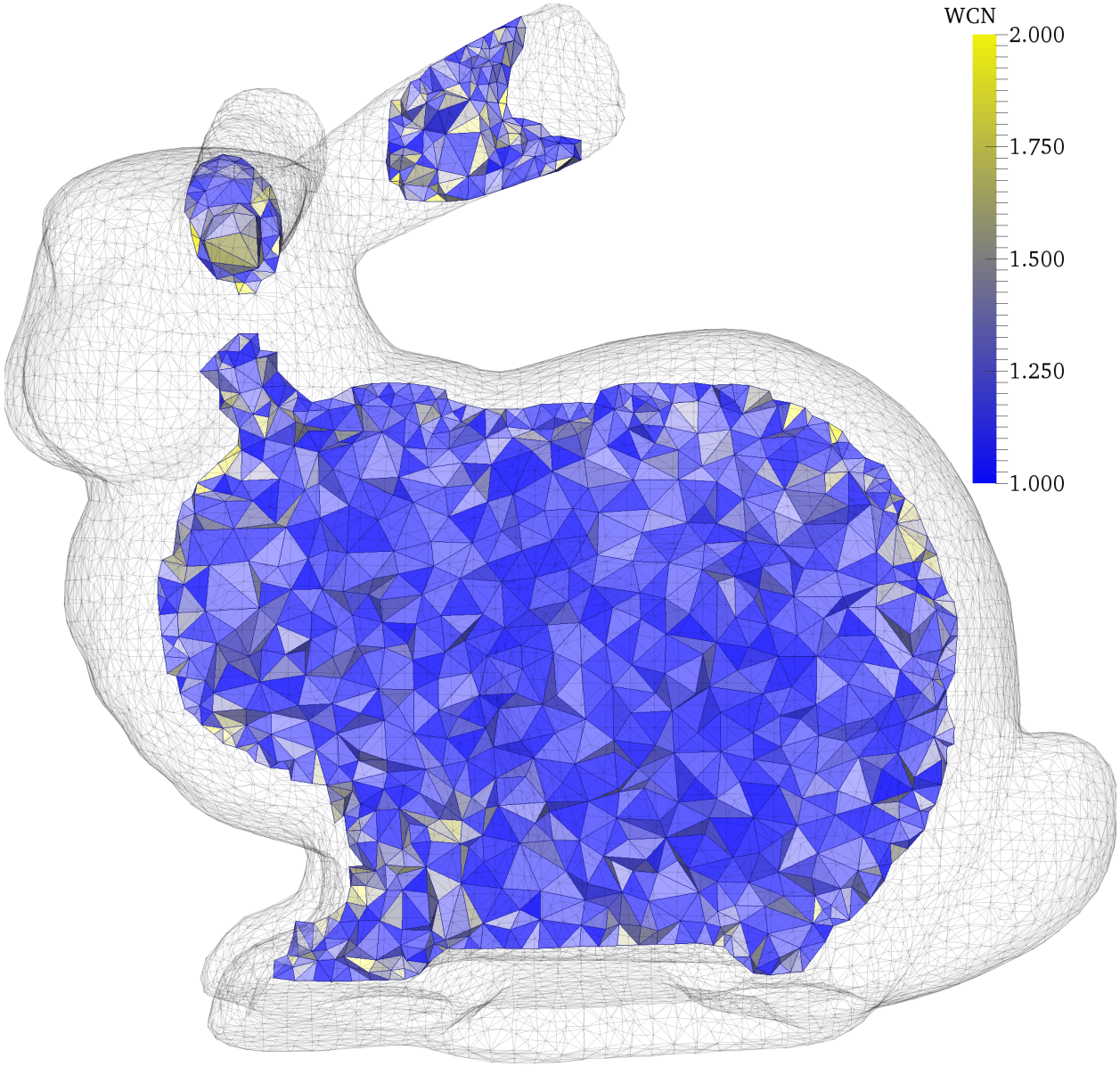


Figure 3.37 Stanford Bunny Slice of Tetmesh Grid

3.5 Timing Results

Presented here is a very brief discussion of the timings involved in mesh optimization and creation. All times listed below are for cases run on a Linux desktop with a 2.4GHz CPU. Overall timings are given in Table 3.13.

As speed was never a goal of this research, there is little guilt in admitting that the code as implemented is slow. For example, the mesh on the ONERA M6 Wing took Tetmesh over 17 hours to make. In contrast, Pointwise takes less than 20 seconds. The notional spaceplane took Tetmesh over 9 days! Pure optimization times are not nearly as daunting but still excessive. Obviously, these times are unacceptable in a production environment.

Despite the apparently abysmal performance, the times for these cases do compare well with those of Klinger [77, 78]. For example, optimization of mesh an anthropomorphic marshmallow with 102,393 elements took at least 5,376 seconds, or about 19 elements per second, depending upon the choice of optimization functions. The Three Boxes Test Case optimization, that has 140,266 elements took 5,792 seconds or about 24 elements per second . There are similar comparisons for other meshes. Klinger's timings are from a 2.66 GHz Mac Pro.

Experience shows that the slowest part of the process is topological optimization. The enormous amount of cost checking and connectivity updates involved seem to take a good bit of computational time. It is very much believed that this bottleneck, as well as other slow parts of the codebase, could be ameliorated by some more intelligent application of computer science techniques than have thus far been employed. Any such improvement would bring the pure optimization cases in line with reasonable production times. Also, at this juncture, the algorithms have only been implemented in serial.

Table 3.13 Run Times

Case	*Time (s)*
Pure Optimization Cases	
Simple Sphere, Pointwise	80
Simple Sphere, TetGen	71
Three Boxes Test Case	5,792
ONERA M6 Wing, Pointwise	1,006
ONERA M6 Wing, TetGen	1,275
Notional Spaceplane	15,310
Eglin Wing Pylon Store	1,662
Stanford Bunny	833
Creation in Tetmesh	
Simple Sphere	471
Three Boxes Test Case	8,868
ONERA M6 Wing	62,325
Notional Spaceplane	793,551
Eglin Wing Pylon Store	66,021
Stanford Bunny	13,800

CHAPTER 4

DISCUSSION

The general tetrahedral mesh optimization scheme presented here has been shown to be successful. Topological changes combined with weighted condition number gradient driven smoothing is clearly a viable and useful scheme. What follows is a discussion of the several aspects of the schemes presented with an emphasis on the lessons learned and possible future research.

The Lawson Style Optimization Insertion algorithm was itself shown to be very successful. Even in the random insertion cases in Section 3.2, inserting and optimizing at the same time produced reasonable results. The ultimate advantage of such a methodology lies in the ability to insert points anywhere in the mesh and expect reasonable results. For example, one may be able to identify undesirable tetrahedra and refine them without fear of producing poor quality elements. A purely Delaunay based scheme can provide no such guarantee.

LSOI does have two apparent drawbacks. First, it can tend to create the simply partitioned tetrahedra as discussed in Section 2.5.4. While these elements do tend to be of lower quality, in practice this tendency is trivial as those nodes are quite easy to remove. The other main disadvantage is time. The implementation of LSOI used in Tetmesh was much slower than the Bowyer-Watson style insertion. This is owed to the fact that the mesh convergence cycle itself was slow. As stated in Section 3.5, it is believed this is mostly due to the slowness in the topological optimization routine.

Weighted condition number has shown itself to be a fine quality metric producing high quality meshes. The mathematical analysis in Section 2.1 shows tetrahedra with a particular range of WCN to be high quality elements. In a certain sense, this work begs the question of the efficacy of WCN by assuming it is a good quality metric. However, what is really being assumed is that equiangularity is a good quality metric and WCN has been shown to be a fine measure of that property.

Another interesting result is how well the meshes created by Pointwise and TetGen via a Bowyer-Watson insertion scheme lent themselves to optimization via topological changes and vertex soothing. In all the cases presented in Chapter 3 the commercial methods showed good to dramatic improvement when the mesh convergence cycle was applied to them. Indeed, in Section 3.2, the initial randomly created Bowyer-Watson style mesh was of terrible quality but optimization was readily able to improve it to a usable state. These results suggest there is something about Bowyer-Watson style insertion that well prepares a mesh for flips and node smoothing. Indeed, Frietag had similar results with her schemes based on meshes created with Bowyer-Watson insertion. (See Section 2.3.)

Experience would seem to indicate that the reason Bowyer-Watson meshes are easy to optimize is the amount of new connectivity that algorithm tends to generate. Only in rare cases would the circumsphere test indicate that only a single tetrahedron should be deleted to form the convex hull into which to insert the new node. Thus, the new vertex will likely be connected to at least five and probably many more vertices. LSOI, on the other hand, will only delete the single tetrahedron in which the new node is located unless the node happens to be on an edge or face. Even

in such a case, LSOI will not make more connectivity than Bowyer-Watson. This fact is evidenced by the existence of the simply partitioned tetrahedra.

It may be possible to avert this problem by simply forcing LSOI to artificially delete more tetrahedra in the vicinity of the node being inserted thus mimicking the behavior of Bowyer-Watson. This approach may, however, lead to the same numerical problems present in Bowyer-Watson; so care will have to be taken to avoid these issues. On the other hand, since LSOI does not depend on the Delaunay property, there will be little harm in leaving a tetrahedron that could otherwise have been successfully deleted.

One of the more interesting aspects of the mesh convergence cycle is its tendency to produce elements around WCN $\approx$ 1.3. Such behavior was apparent in all the cases presented in Chapter 3. This phenomenon is in contrast to the results of the Delaunay-based methods in Pointwise and TetGen. Much of this behavior can be attributed to the design philosophy of the algorithms i.e. the sacrificing of high quality tetrahedra to improve poor quality elements. However, precisely why the spikes happen at this particular value is still a matter up for debate.

Perhaps the weakest part of the Tetmesh creation scheme comes from the octree based pseudo-tiling. Unfortunately, as it is currently implemented, the octree is not created to ensure each octant, and therefore the points chosen for insertion, is a cube. This behavior could be seen in Onera M6 Wing case in Section 3.4.3. Also, as several of the cases in Chapter 3 showed, the gradation in points given by the octree is not sufficient. The Pointwise meshes clearly packed more points close to the geometry.

Fortunately both of these issues are easy to correct. Ensuring that the root octant of the tree is a cube is trivial. This change would simply require the bounding box created around the

triangulation itself be a cube. All of the octants would then, of course, be cubic. Improving gradation of points given by this method is not much more of a challenge. One must simply ensure that the octree is refined such that the leaf octants do not increase in size too rapidly. This refinement could be accomplished by enforcing leaf spacing based upon their neighbors.

In general, though, the sometimes poor quality tetrahedra created at the boundaries lead one to think that perhaps the best method of point choice near the boundary should be based upon the boundary triangulation itself. A choosing of points in some way similar to Marcum's AFLR (See Section 1.4.2) seems very desirable. In that same vein of thinking, there was some thought and research given to finding the optimal point subtended by each surface triangle to create the highest quality tetrahedron possible. Finding such optimal point placement near the boundary seems like a area ripe for exploration. Indeed, optimal point placement is a subject not only for refining the grid near the boundary but for all parts of the mesh.

Generally speaking, the work presented here has been quite fruitful. The results show the methodologies have utility in the field of tetrahedral meshing. It is hoped the optimization algorithms can be successfully and efficiently integrated into other research and production codes.

APPENDIX A

DUAL NUMBERS

The gradients referenced in Section 2.1 were calculated numerically using dual numbers. What follows is a simple explication of dual numbers and their implementation. This presentation is merely a reprisal of the unpublished work of Hyams [88].

Define some infinitesimal number, ϵ, to have the special property that $\epsilon^2 = 0$. Suppose this numbers interacts with real numbers in all the ways one would expect. Specifically assume additive commutativity, $a + \epsilon = \epsilon + a$, and multiplicative commutativity, $a\epsilon = \epsilon a$, hold for all real numbers a.

Consider some real valued function $f(x)$ and it's Taylor series expansion

$$f(x + \Delta x) = f(x) + \Delta x \left.\frac{df}{dx}\right|_x + \frac{1}{2!}\Delta x^2 \left.\frac{d^2 f}{dx^2}\right|_x + \frac{1}{3!}\Delta x^3 \left.\frac{d^3 f}{dx^3}\right|_x + \ldots \tag{A.1}$$

about x. Substituting ϵ for Δx gives

$$f(x + \epsilon) = f(x) + \epsilon \left.\frac{df}{dx}\right|_x + \frac{1}{2!}\epsilon^2 \left.\frac{d^2 f}{dx^2}\right|_x + \frac{1}{3!}\epsilon^3 \left.\frac{d^3 f}{dx^3}\right|_x + \ldots \tag{A.2}$$

However, since $\epsilon^2 = \epsilon^3 = \ldots = 0$, we have

$$f(x + \epsilon) = f(x) + \epsilon \left.\frac{df}{dx}\right|_x \tag{A.3}$$

What this means is that, if we evaluate the function f with $x + \epsilon$, the result will be given in two parts: the real part, $f(x)$, and the coefficient on ϵ, $\frac{df}{dx}$ evaluated at x.

This idea extends easily to multivariable functions. For some function $f(x_1, x_2, \ldots)$,

$$f(x_1 + \epsilon_1, x_2 + \epsilon_2, \ldots) = f(x_1, x_2, \ldots) + \epsilon_1 \left.\frac{df}{dx_1}\right|_{x_1, x_2, \ldots} + \epsilon_2 \left.\frac{df}{dx_2}\right|_{x_1, x_2, \ldots} + \ldots \tag{A.4}$$

This result gives us a wonderful way to calculate the derivatives of any function. As an example, consider the simple polynomial

$$f(x) = 3x^2 - 9x + 6 \tag{A.5}$$

So, of course,

$$f'(x) = 6x - 9 \tag{A.6}$$

Note that $f(1.6) = -0.72$ and $f'(1.6) = 0.6$. Now, using the method described above, if we evaluate $f(1.6 + \epsilon)$ we get

$$\begin{aligned} f(1.6 + \epsilon) &= 3(1.6 + \epsilon)^2 - 9(1.6 + \epsilon) + 6 \\ &= 3\left(2.56 + 3.2\epsilon + \epsilon^2\right) - 9(1.6 + \epsilon) + 6 \\ &= 3(2.56 + 3.2\epsilon) - 9(1.6 + \epsilon) + 6 \\ &= 7.68 + 9.6\epsilon - 14.4 - 9\epsilon + 6 \\ &= -0.72 + 0.6\epsilon \end{aligned} \tag{A.7}$$

with the real part giving the value of the function an the coefficient on ϵ giving the value of the derivative.

Multivariable functions are just as simple to calculate. Consider

$$f\left(x, y, z\right) = 3xy + \sin\left(xz\right) \tag{A.8}$$

and, so,

$$\frac{\partial f}{\partial x} = 3y + z\cos\left(xz\right), \; \frac{\partial f}{\partial y} = 3x, \text{ and } \frac{\partial f}{\partial z} = x\cos\left(xz\right) \tag{A.9}$$

Note that $f\left(1.1, 2.2, 0\right) = 7.26$, $f_x\left(1.1, 2.2, 0\right) = 3.3$, $f_y\left(1.1, 2.2, 0\right) = 6.6$, and $f_z\left(1.1, 2.2, 0\right) = 1.1$. To find these derivatives with a dual number calculation, we do confront the issue of finding the sine of a dual number. However, invoking Equation A.3 gives

$$\begin{aligned} \sin\left(x + k\epsilon\right) &= \sin\left(x\right) + k\epsilon \left.\frac{d\left[\sin\left(x\right)\right]}{dx}\right|_x \\ &= \sin\left(x\right) + k\epsilon\cos\left(x\right) \end{aligned} \tag{A.10}$$

So, if we evaluate $f\left(x + \epsilon_1, y + \epsilon_2, z + \epsilon_3\right)$ we get

$$\begin{aligned} f\left(1.1 + \epsilon_1, 2.2 + \epsilon_2, 0 + \epsilon_3\right) &= 3\left(1.1 + \epsilon_1\right)\left(2.2 + \epsilon_2\right) + \sin\left(\left(1.1 + \epsilon_1\right)\left(0 + \epsilon_3\right)\right) \\ &= 7.26 + 6.6\epsilon_1 + 3.3\epsilon_2 + \epsilon_1\epsilon_2 + \sin\left(1.1\epsilon_3 + \epsilon_1\epsilon_3\right) \\ &= 7.26 + 6.6\epsilon_1 + 3.3\epsilon_2 + \sin\left(1.1\epsilon_3\right) \end{aligned} \tag{A.11}$$

Note, by the above calculation, $\sin(1.1\epsilon_3) = \sin(0 + 1.1\epsilon_3) = \sin(0) + 1.1\epsilon_3 \cos(0) = 1.1\epsilon_3$. Thus

$$f(1.1 + \epsilon_1, 2.2 + \epsilon_2, 0 + \epsilon_3) = 7.26 + 6.6\epsilon_1 + 3.3\epsilon_2 + 1.1\epsilon_3 \tag{A.12}$$

where, similar to the polynomial above, we have the real part giving the value of the function and the coefficients on $\epsilon_1, \epsilon_2, \epsilon_3$ giving the values of the derivative.

The true value of dual numbers is their simplicity in programmatic implementation, especially in languages where templating is available.

Given below is a sample of the C++ dual number class used in this research. Ellipses indicate omitted material.

```
#ifndef MB_Dual_12_h
#define MB_Dual_12_h

#define NEQN 12

class MB_Dual_12{
  public:
  int index;
  double value;
  double deriv[NEQN];

  //member functions
```

```
//constructor
inline MB_Dual_12(){
  value = 0.0;
  for(int i=0; i<NEQN; i++) deriv[i] = 0.0;
}

. . .

/*************************************************************
OVERLOADED OPERATORS AND FUNCTIONS
*************************************************************/
//Addition
inline MB_Dual_12 operator + (const double &num){
  MB_Dual_12 result;
  result.value = value + num;
  for(int i=0; i<NEQN; i++) result.deriv[i] = deriv[i];
  return result;
}
inline friend MB_Dual_12 operator + (const double d,
      const MB_Dual_12 &num){//double + dual
  MB_Dual_12 result;
```

```
  result.value = d + num.value;
  for(int i=0; i<NEQN; i++) result.deriv[i] = num.deriv[i];
  return result;
}
inline MB_Dual_12 operator + (const MB_Dual_12 &num){
  MB_Dual_12 result;
  result.value = value + num.value;
  for(int i=0; i<NEQN; i++) result.deriv[i]
        = deriv[i] + num.deriv[i];
  return result;
}
//Subtraction
inline MB_Dual_12 operator - (const double &num){
  MB_Dual_12 result;
  result.value = value - num;
  for(int i=0; i<NEQN; i++) result.deriv[i] = deriv[i];
  return result;
}
. . .
inline friend MB_Dual_12 sqrt(const MB_Dual_12 &num){
  MB_Dual_12 result;
```

```
    result.value = std::sqrt(num.value);
    for(int i=0; i<NEQN; i++)
      result.deriv[i] = 0.5*num.deriv[i]/std::sqrt(num.value);
    return result;
  }

. . .

  inline bool operator >= (const MB_Dual_12 d){
    if(value >= d.value)  return true;
    else                  return false;
  }
  inline bool operator == (const MB_Dual_12 d){
    if(value == d.value)  return true;
    else                  return false;
  }
  inline bool operator !=  (const MB_Dual_12 d){
    if(value != d.value)  return true;
    else                  return false;
  }
```

```
. . .


};
#endif
```

Some further experimentation by Dr. Steve Karman has indicated that manually "unrolling" the loops in this class can significantly improve the computational speed. The code is presented here in this form since it is the state in which this work has been presented.

APPENDIX B

INTEGER PAIRING

In the course of creating the code for this research, it was often useful to be able to store a pair of integers. For example, all vertices in the mesh were identified with a node number so a convenient way to mark an edge is with the pair of integers that define it. A similar marking was used for the face shared by two tetrahedra. However, no edge or face lookups were otherwise maintained in the code. So, an elegant mathematical solution in the form of pairing functions was used.

A pairing function is any isomorphism (i.e. one-to-one and onto function) from pairs of members of a set S to members of that set. That is, f is a pairing function if, for every $a, b, c \in S$,

$$f([a,b]) = c \iff f^{-1}(c) = [a,b] \tag{B.1}$$

where $[a, b]$ is a pair of element in which the order is relevant. Importantly here, there are several pairing functions from the set of natural number pairs to the set of natural numbers (i.e. from pairs to positive integers to positive integers.) Such functions are used in theoretical computer science as well as to show that the cardinality of the set of integers and the set of rational numbers is the same.

The most famous of such functions is Cantor's pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ [89, 90] defined as

$$\pi([n_1, n_2]) = \frac{1}{2}(n_1 + n_2)(n_1 + n_2 + 1) + n_2 \tag{B.2}$$

Without going into the details, it can be shown that this function is invertible and, therefore an isomorphism.

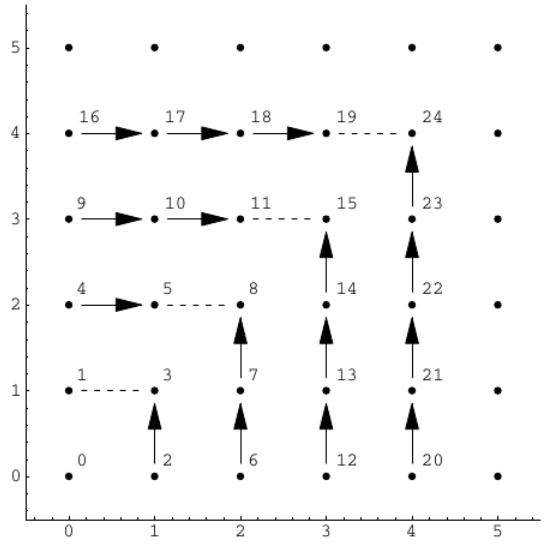


Figure B.1 The Elegant Pairing Function

However, the particular pairing function used in this work, was taken from Szudzik [91]. This function, called the elegant pairing function, $E : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined as

$$E\left([n_1, n_2]\right) = \begin{cases} n_2^2 + n_1 & n_1 \neq \max\{n_1, n_2\} \\ n_1^2 + n_1 + n_2 & n_1 = \max\{n_1, n_2\} \end{cases} \tag{B.3}$$

E orders the pairs of positives integers by assigning consecutive numbers to points along the sides of a square. See Figure B.1, also taken from [91]. If $E\left([n_1, n_2]\right) = z$, then the pair associated with z is given by the inverse function

$$E^{-1}(z) = \begin{cases} \left[z - \left\lfloor\sqrt{z}\right\rfloor^2, \left\lfloor\sqrt{z}\right\rfloor\right] & z - \left\lfloor\sqrt{z}\right\rfloor^2 < \left\lfloor\sqrt{z}\right\rfloor \\ \left[\left\lfloor\sqrt{z}\right\rfloor, z - \left\lfloor\sqrt{z}\right\rfloor^2 - \left\lfloor\sqrt{z}\right\rfloor\right] & z - \left\lfloor\sqrt{z}\right\rfloor^2 \geq \left\lfloor\sqrt{z}\right\rfloor \end{cases} \tag{B.4}$$

where $\lfloor x \rfloor$ is the greatest integer less than, or floor of, x. For example, $E([3,4]) = 4^2 + 3 = 19$ since $3 < 4$ and, as expected $E^{-1}(19) = \left[19 - \left\lfloor\sqrt{19}\right\rfloor^2, \left\lfloor\sqrt{19}\right\rfloor\right] = [3,4]$ since $19 - \left\lfloor\sqrt{19}\right\rfloor^2 < \left\lfloor\sqrt{19}\right\rfloor$ or $3 < 4$.

Note that the inverse function is somewhat complicated and involves a square root. Fortunately, this does not matter for edge and face marking as the inverse is never needed. When marking an edge or face, the pair of integers associated with the element are ordered so that $n_1 < n_2$. Thus, $E([n_1, n_2]) = n_2^2 + n_1 = z$. This result is stored and, so, to later check if that edge or face is marked, one need only check if z has been stored. This procedure was easily implemented in the code for this research with a C++ std::set container. Future work may use this procedure to efficiently identify edges and faces throughout the entire meshing process.

REFERENCES

[1] Babuska, I. and Aziz, A. K., "On the Angle Condition in the Finite Element Method," *SIAM Journal on Numerical Analysis*, Vol. 13, No. 2, 1976, pp. 214–226.

[2] Shewchuk, J. R., "What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures," *Proceedings of the 11th International Meshing Roundtable*, 2002, pp. 115–126.

[3] Bern, M. and Eppstein, D., "Mesh generation and optimal triangulation," *Computing in Euclidean geometry*, Vol. 1, 1992, pp. 23–90.

[4] George, P. L., "Tet meshing: construction, optimization and adaptation," *Proceedings of the 8th International Meshing Roundtable*, 1999.

[5] Danilov, A. A., "Unstructured Tetrahedral Mesh Generation Technology," *Computational Mathematics and Mathematical Physics*, Vol. 50, No. 1, Feb. 2010, pp. 139–156.

[6] Delaunay, B., "Sur la Sphre Vide. A la Memoire de Georges Voronoi," *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, Vol. 7, No. 6, 1934, pp. 793–800.

[7] Voronoi, G., "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites." *Journal für die reine und angewandte Mathematik*, Vol. 133, 1908, pp. 97–178.

[8] Chew, L. P., "Voronoi Diagram / Delaunay Triangulation," http://www.cs.cornell.edu/info/people/chew/delaunay.html.

[9] Sibson, R., "Locally equiangular triangulations," *The Computer Journal*, Vol. 21, 1978, pp. 243–245.

[10] Rajan, V. T., "Optimality of the Delaunay Triangulation in Rˆd," *Discrete & Computational Geometry*, Vol. 12, No. 1, 1994, pp. 189–202.

[11] Lawson, C. L., "Software for C1 surface interpolation," Tech. rep., California Institute of Technology, Pasadena, California, 1977.

[12] Bowyer, A., "Computing Dirichlet tessellations," *The Computer Journal*, Vol. 24, 1981, pp. 162–166.

[13] Watson, D. F., "Computing the n-dimensional Delauny tesselation with application to voronoi polytypes," *Computer Journal*, Vol. 24, 1981, pp. 167–172.

[14] Chew, L. P., "Contrained Delaunay Triangulation," *Algorithmica*, Vol. 4, No. 1-4, 1989, pp. 97–108.

[15] Chew, L. P., "Guaranteed-Quality Triangular Meshes," Cornell University, 1989.

[16] Ruppert, J. M., *Results on Triangulation and High Quality Mesh Generation*, Ph.D. thesis, University of California at Berkeley, 1992.

[17] Ruppert, J., "A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation," Tech. rep., University of California at Berkley, 1992.

[18] Ruppert, J., "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation," *Journal of Algorithms*, Vol. 18, 1994, pp. 548–585.

[19] Bern, M., Eppstein, D., and Gilbert, J., "Provably good mesh generation," *Journal of Computer and System Sciences*, Vol. 48, 1994, pp. 384–409.

[20] Marcum, D., "Generation of unstructured grids for viscous flow applications," *33rd Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, Reston, Virigina, Jan. 1995.

[21] Marcum, D. L. and Weatherill, N. P., "Unstructured Grid Generation using Iterative Point Insertion and Local Reconnection," *AIAA Journal*, Vol. 33, No. 9, 1995, pp. 1619–1625.

[22] Marcum, David, L., "Unstructured Grid Generation Using Automatic Point Insertion and Local Reconnection," *Handbook of Grid Generation*, edited by J. F. Thompson, B. K. Soni, and N. P. Weatherill, 1999, pp. 18–1 – 18–31.

[23] Chen, H. and Bishop, J., "Delaunay Triangulation for Curved Surfaces," *Proceedings of the 6th International Meshing Roundtable*, 1997.

[24] Borouchaki, H. and George, P. L., "Optimal Delaunay Point Insertion," *International Journal for Numerical Methods in Engineering*, Vol. 39, No. 1996, 1996, pp. 3407–3437.

[25] Shewchuk, J. R., *Delaunay refinement mesh generation*, Phd dissertation, Carnegie Mellon University, 1997.

[26] Ruppert, J. and Seidel, R., "On the Difficulty of Tetrahedralizing 3-dimensional Non-Convex Polyhedra," *Proceedings of the fifth annual symposium on Computational geometry*, 1989, pp. 380–392.

[27] George, P. L., Borouchaki, H., and Saltel, E., "'Ultimate' robustness in meshing an arbitrary polyhedron," *International Journal for Numerical Methods in Engineering*, Vol. 58, No. 7, Oct. 2003, pp. 1061–1089.

[28] George, P. L., Hecht, F., and Saltel, E., "Automatic mesh generator with specified boundary," *Computer Methods in Applied Mechanics and Engineering*, Vol. 92, No. 3, Nov. 1991, pp. 269–288.

[29] George, P. L., "Improvements on Delaunay-based three-dimensional automatic mesh generator," *Finite Elements in Analysis and Design*, Vol. 25, No. 3-4, April 1997, pp. 297–317.

[30] Shewchuk, J. R., "Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery." *Proceedings of the 11th International Meshing Roundtable*, 2002.

[31] Shewchuk, J. R., "Constrained Delaunay Tetrahedralization, Bistellar Flips, and Provably Good Boundary Recovery," 2003.

[32] Si, H. and Gärtner, K., "3D boundary recovery by constrained Delaunay tetrahedralization," *International Journal for Numerical Methods in Engineering*, , No. September 2010, 2011, pp. 1341–1364.

[33] Du, Q. and Wang, D., "Boundary recovery for three dimensional conforming Delaunay triangulation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 193, No. 23-26, June 2004, pp. 2547–2563.

[34] Liu, Y., Lo, S. H., Guan, Z. Q., and Zhang, H.-W., "Boundary recovery for 3D Delaunay triangulation," *Finite Elements in Analysis and Design*, Vol. 84, 2014, pp. 32–43.

[35] Ghadyani, H., Sullivan, J., and Wu, Z., "Boundary recovery for Delaunay tetrahedral meshes using local topological transformations." *Finite Elements in Analysis and Design*, Vol. 46, No. 1-2, Jan. 2010, pp. 74–83.

[36] Chew, L. P., "Guaranteed-Quality Delaunay Meshing in 3D (short version)," *Proceedings of the thirteenth annual Symposium on Computational Geometry*, ACM Press, New York, New York, USA, 1997, pp. 391–393.

[37] Shewchuk, J. R., "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," 1997.

[38] Shewchuk, J. R., "Tetrahedral mesh generation by Delaunay refinement," *Proceedings of the fourteenth annual symposium on Computational geometry*, ACM Press, 1998, pp. 86–95.

[39] Shewchuk, J. R., "Sweep Algorithms for Constructing Higher-Dimensional Constrained Delaunay Triangulations," *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, 2000, pp. 350–359.

[40] Si, H., *Three dimensional boundary conforming Delaunay mesh generation*, Ph.D. thesis, Technische Universität Berlin, Fakultät II, 2008.

[41] Si, H., "An Analysis of Shewchuk's Delaunay Refinement Algorithm," *Proceedings of the 18th International Meshing Rountable*, 2009.

[42] Si, H., "TetGen A Quality Tetrahedral Mesh Generator," *ACM Transactions on Mathematical Software*, Vol. 41, No. 2, 2015.

[43] Borouchaki, H., Hecht, F., Saltel, E., and George, P. L., "Reasonably efficient Delaunay based mesh generator in 3 dimensions," *Proceedings of the 4th International Meshing Roundtable*, 1995.

[44] Miller, G. L., Talmor, D., Teng, S.-H., Walkington, N., and Wang, H., "Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening," *Proceeding of the 5th International Meshing Roundtable*, 1996, pp. 47–62.

[45] Gosselin, S. and Ollivier-Gooch, C., "Tetrahedral mesh generation using Delaunay refinement with non-standard quality measures," *International Journal for Numerical Methods in Engineering*, , No. February, 2011, pp. 795–820.

[46] Marcum, D. L., "Efficient Generation of High-Quality Unstructured Surface and Volume Grids," *Engineering With Computers*, Vol. 17, 2001, pp. 211–233.

[47] Weatherill, N. P., Hassan, O., and Marcum, D. L., "Compressible Flowfield Solutions with Unstructured Grids Generated by Delaunay Triangulation," *AIAA Journal*, Vol. 33, No. 7, 1995, pp. 1196–1204.

[48] Weatherill, N. P., "Generation of unstructured grids using Dirichlet tessellations," 1985.

[49] Weatherill, N. P., Hassan, O., and Marcum, D. L., "Calculation of Steady Compressible Flowfields with the Finite Element Method," *Proceedings of the 31st Aerospace Sciences Meeting & Exhibit*, Reno, Nevada, 1993.

[50] George, P. L. and Borouchaki, H., "Back to Edge Flips in 3 Dimensions." *Proceedings of the 12th International Meshing Roundtable*, 2003.

[51] Joe, B., "Three-Dimensional Triangulations from Local Transformations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 10, No. 4, 1989, pp. 718–741.

[52] Joe, B., "Construction of k-Dimensional Delaunay Triangulations Using Local Transformations," *SIAM Journal on Scientific Computing*, Vol. 14, No. 6, 1993, pp. 1415–1436.

[53] Joe, B., "Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations," *SIAM Journal on Scientific Computing*, Vol. 16, No. 6, 1995, pp. 1292–1307.

[54] Joe, B., "Delaunay versus Max-Min Solid Angle Triangulations for Three-Dimensional Mesh Generation," *International Journal for Numerical Methods in Engineering*, Vol. 31, 1991, pp. 987–997.

[55] Shewchuk, J. R., "Two discrete optimization algorithms for the topological improvement of tetrahedral meshes," .

[56] Klincsek, G. T., "Minimal Triangulations of Polygonal Domains," *Annals of Discrete Mathematics*, Vol. 9, No. C, 1980, pp. 121–123.

[57] Buell, W. R. and Bush, B. A., "Mesh Generation-A Survey," *Journal of Manufacturing Science and Engineering*, Vol. 95, No. 1, 1973, pp. 332–338.

[58] Field, D. A., "Laplacian Smoothing and Delaunay Triangulations," *Communications of Applied Numerical Methods*, Vol. 4, No. December 1987, 1988, pp. 709–712.

[59] Ollivier-Gooch, C., "An Unstructured Mesh Improvement Toolkit with Application to Mesh Improvement, Generation, and (De-)Refinement," *AIAA Aerospace Sciences Meeting and Exhibit*, Vol. 36, University of British Columbia, American Institute of Aeronautics and Astronautics, 1998.

[60] Dompierre, J., Labbé, P., Guibault, F., and Camarero, R., "Proposal of Benchmarks for 3D Unstructured Tetrahedral Mesh Optimization," *Proceedings of the 7th International Meshing Roundtable*, 1998.

[61] Parthasarathy, V. N. and Kodiyalam, S., "A constrained optimization approach to finite element mesh smoothing," *Finite Elements in Analysis and Design*, Vol. 9, No. 4, Sept. 1991, pp. 309–320.

[62] Canann, S. A., Stephenson, M. B., and Blacker, T., "Optismoothing: An optimization-driven approach to mesh smoothing," *Finite Elements in Analysis and Design*, Vol. 13, June 1993, pp. 185–190.

[63] Canann, S. A., Tristano, J. R., and Staten, M. L., "An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes." *Proceedings of the 7th International Meshing Roundtable*, 1998, pp. 15–17.

[64] Amezua, E., Hormaza, M. V., Hernandez, A., and Ajuria, M. B. G., "A method for the improvement of 3D solid finite-element meshes," *Advances in Engineering Software*, Vol. 22, 1995, pp. 45–53.

[65] Amenta, N., Bern, M., and Eppstein, D., "Optimal Point Placement for Mesh Smoothing," *Journal of Algorithms*, Vol. 30, Sept. 1998, pp. 302–322.

[66] Zhou, T. and Shimada, K., "An Angle-Based Approach to Two-Dimensional Mesh Smoothing," *Proceedings of the 9th International Meshing Roundtable*, 2000.

[67] Xu, H. and Newman, T. S., "An angle-based optimization approach for 2D finite element mesh smoothing," *Finite Elements in Analysis and Design*, Vol. 42, No. 13, Sept. 2006, pp. 1150–1164.

[68] Chen, Z., Tristano, J. R., and Kwok, W., "Construction of an objective function for optimization-based smoothing," *Engineering with Computers*, Vol. 20, No. 3, Aug. 2004, pp. 184–192.

[69] Chen, L., "Mesh Smoothing Schemes Based on Optimal Delaunay Triangulations," *Proceedings of the 13th International Meshing Roundtable*, Williamburg, VA, 2004.

[70] De L'Isle, E. and George, P. L., "Optimization of Tetrahedral Meshes," *IMA Volumes in Mathematics and its Applications*, Vol. 75, 1995, pp. 97–128.

[71] Freitag, L., Jones, M., and Plassmann, P., "An Efficient Parallel Algorithm for Mesh Smoothing," *Proceedings of the 4th International Meshing Roundtable*, 1995.

[72] Freitag, L. A. and Ollivier-Gooch, C., "A Comparison of Tetrahedral Mesh Improvement Techniques," *Proceeding of the 5th International Meshing Roundtable*, Argonne National Laboratory (ANL), Argonne, IL, Dec. 1996.

[73] Freitag, L. A. and Ollivier-Gooch, C., "Tetrahedral mesh improvement using swapping and smoothing," *International Journal for Numerical Methods in Engineering*, Vol. 40, 1997, pp. 3979–4002.

[74] Freitag, L. A., "On combining Laplacian and optimization-based mesh smoothing techniques," *Joint ASME, ASCE, SES symposium on engineering mechanics in manufacturing processes and materials processing*, Evanston, IL, 1997.

[75] Freitag, L. A. and Ollivier-Gooch, C., "A Cost/Benefit Analysis of Simplicial Mesh Improvement Techniques as Measured by Solution Efficiency," *International Journal of Computational Geometry & Applications*, Vol. 10, No. 4, 2000, pp. 361–382.

[76] Freitag, L. A. and Plassmann, P., "Local optimization-based simplical mesh untangling and imporovement," *International Journal for Numerical Methods in Engineering*, Vol. 49, 2000, pp. 109–125.

[77] Klingner, B. M. and Shewchuk, J. R., "Aggressive tetrahedral mesh improvement," *Proceedings of the 16th International Meshing Roundtable*, 2008.

[78] Klingner, B., *Tetrahedral Mesh Improvement*, Ph.D. thesis, University of California at Berkeley, 2009.

[79] Karman, S. L. J., "Hierarchical unstructured mesh generation," *AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, 2004.

[80] Sahasrabudhe, M. S., Karman, S. L. J., and Anderson, W. K., "Grid Control of Viscous Unstructured Meshes Using Optimization," *44th AIAA Aerospace Sciences Meeting and Exhibit*, No. January, 2006.

[81] Karman, S. L., "Adaptive Optimization-Based Smoothing for Tetrahedral Meshes," *53rd AIAA Aerospace Science Meeting*, Kissimmee, Florida, 2015.

[82] Freitag, L. A. and Knupp, P. M., "Tetrahedral mesh improvement via optimization of the element condition number," *International Journal for Numerical Methods in Engineering*, Vol. 53, 2002, pp. 1377–1391.

[83] Pickover, C. A., *The Math Book: from Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics*, Sterling Publishing Company, Inc., 2009.

[84] ElGindy, H., Everett, H., and Toussaint, G., "Slicing an ear using prune-and-search," *Pattern Recognition Letters*, Vol. 14, No. 9, 1993, pp. 719–722.

[85] "ONERA M6 Wing," http://www.grc.nasa.gov/WWW/wind/valid/m6wing/m6wing.html.

[86] Heim, E. R., "CFD Wing/Pylon/Finned Store Mutual Interference Wind Tunnel Experiment," Tech. rep., Arnold Engineering Development Center, 1991.

[87] "The Stanford 3D Scanning Repository," https://graphics.stanford.edu/data/3Dscanrep/.

[88] Hyams, D. G., "Using Dual Numbers for Automatic Derivative Calculations," .

[89] Cantor, G., "Ein Beitrag zur Mannigfaltigkeitslehre." *Journal für die reine und angewandte Mathematik*, Vol. 84, 1878, pp. 242–258.

[90] "Pairing Function," http://en.wikipedia.org/wiki/Pairing_function.

[91] Szudzik, M., "An elegant pairing function," *NKS2006 Conference*, 2006.

VITA

Christopher Bruce Hilbert was born March 14th, 1976 in Chattanooga, Tennessee to Gary and Deborah Hilbert. He attended Boyd-Buchanan High School where he graduated Valedictorian in 1994. Bruce earned a Bachelor of Science with Honors, Summa Cum Laude, with majors in Mathematics and History from Samford University in 1998. In 2001, he earned a Masters of Education from the University of Tennessee at Chattanooga while working as a mathematics instructor at Chattanooga State Community College.

Bruce joined the research staff at UTC's SimCenter, where he was already a student, in January of 2009 and that year earned his Masters of Science in Computational Engineering. He has been the chief grid generation specialist at the SimCenter since that time.