Isosurface Stuffing Improved

Acute Lattices and Feature Matching

by

Crawford Doran

B. Software Engineering, University of Waterloo, 2011

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

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

October 2013

Abstract

We present two improvements to Labelle and Shewchuk's isosurface stuffing algorithm for tetrahedral mesh generation. First, we use an acute tetrahedral lattice known as the A15 lattice in place of the original BCC lattice. In the uniform case, the higher-quality tetrahedra of A15 significantly improve the quality of the resulting mesh; BCC remains the best choice for adaptive meshes, as the A15-based adaptive lattices we have designed are not able to outperform it. Second, we extend the method to match features such as corners and sharp creases. Feature lines and points are matched by snapping nearby mesh vertices onto them. A final vertex smoothing pass mitigates the loss of mesh quality due to feature-matching. Our experiments show that for input surfaces with reasonable crease angles, this is generally sufficient to restore mesh quality.

Preface

This thesis consists primarily of independent work by the author, Crawford Doran. Initial ideas and implementation of the material in Chapter 3 came from Robert Bridson. Athena Chang provided additional help in creating and visualizing the results found in Chapter 6.

A version of the material in Chapter 3, Chapter 5, and Chapter 6 was presented as a Technical Talk at SIGGRAPH 2013, but none of the original material contained herein has seen official publication.

Table of Contents

Abstract . . . . . **ii**

Preface . . . . . **iii**

Table of Contents . . . . . **iv**

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

Glossary . . . . . **vii**

Acknowledgments . . . . . **viii**

Dedication . . . . . **ix**

1 Introduction **1**

2 Overview of Mesh Generation **4**

2.1 Background and Motivation 4

2.2 Mesh Quality 5
2.3 Related Work 7
2.3.1 Delaunay Refinement 7
2.3.2 Advancing Front 9
2.3.3 Octree Methods 9
2.3.4 Lattice Methods 10

3 A15 Isosurface Stuffing 13
3.1 Improving on the BCC Lattice 13
3.2 TCP and Edge Valence Analysis 14
3.3 A15 Lattice 16
3.4 Algorithm 17

4 Adaptivity 21
4.1 Background 21
4.2 Tetrahedral Subdivision 23
4.3 A15 Point Lattice 26
4.4 Adaptive Red-Green A15 29

5 Feature Matching 34
5.1 Algorithm 34
5.2 Feature Endpoints 36
5.3 Feature Paths 38
5.4 Mesh Smoothing 41

6 Results 44

6.1 Uniform A15 Isosurface Stuffing 44

6.2 Feature Matching 48

6.3 Adaptive A15 Lattice 53

7 Conclusion 57

Bibliography 60

Appendix A A15 Tile 63

List of Tables

Table 6.1 Mesh quality for smooth inputs 45
Table 6.2 Smooth inputs with optimization 48
Table 6.3 Nonsmooth inputs with feature matching, after vertex snapping 49
Table 6.4 Nonsmooth inputs with feature matching, after smoothing . . . 49
Table 6.5 Quality of lattices produced with the A15 Point Lattice algorithm 54
Table 6.6 Quality of lattices produced with the Red-Green A15 algorithm 54

List of Figures

Figure 2.1 Comparison of equilateral and sliver tetrahedra. 6

Figure 3.1 Illustration of the BCC lattice. 14

Figure 3.2 A single tile of the A15 lattice viewed from multiple angles. . 17

Figure 3.3 A15 Isosurface stuffing in action. 20

Figure 4.1 Adaptive meshing using BCC lattice. 22

Figure 4.2 Red-green subdivision stencils. 25

Figure 4.3 A15 point lattice (and the octree from which it was generated). 28

Figure 4.4 Transforming one face of A15 tile to double resolution. 30

Figure 4.5 Subdivided bridge cells in the red-green A15 algorithm. 32

Figure 5.1 Feature points (black) and curves (pink) of the Fandisk model. 35

Figure 5.2 Fandisk meshed with and without feature matching. 37

Figure 5.3 Example of feature curve matching. 38

Figure 6.1 Cutaway view of Sphere input, meshed with $h = 0.2$. 45

Figure 6.2 Dragon input, $h = 0.02$. with dihedral angle histogram 46

Figure 6.3 Bunny input, $h = 1.0$. with dihedral angle histogram 47
Figure 6.4 Cube, $h = 0.1$, after smoothing. Cutaway view at right. 48
Figure 6.5 Joint, $h = 0.03$, after smoothing. Cutaway view at right. 50
Figure 6.6 Block, $h = 0.05$, after smoothing. Cutaway views on bottom. . 51
Figure 6.7 Fandisk, $h = 0.03$, after smoothing. Cutaway view at right. . . 52
Figure 6.8 Adaptive lattice (Bunny) from A15 Point Lattice algorithm. . . 55
Figure 6.9 Adaptive lattice (Block) from Red-Green A15 algorithm. . . . 56

Figure A.1 A15 tile overlaid on cube (thick black lines) 64
Figure A.2 Mapping of A15 tile elements to cube. 65

Glossary

BCC Body-Centred Cubic lattice, a space-filling tetrahedral lattice formed by two offset cartesian grids

CDT Constrained Delaunay Tetrahedralization, an algorithm for finding a volumetric triangulation of a space that is constrained to contains a given set of vertices, edges, and faces

FEM Finite Element Method, a procedure for solving partial differential equations on domains discretized into an unstructured mesh of elements

CAD Computer-Aided Design, commonly used in the creation of industrial machine components

TCP Tetrahedral Close-Packed structures, a family of space-filling tetrahedral lattices composed of near-equilateral elements

SDF Signed Distance Field, a data structure that represents an isosurface by storing the signed distance to the surface at regular grid locations

Acknowledgments

Thank you to Prof. Robert Bridson for providing insight, encouragement, patience, and inspiration in my continuing exploration of the world of physics-based animation.

Thank you to Athena Chang for writing test scripts for my algorithms, and for being a sounding board for my ideas over the course of this project.

Thanks also to Essex Edwards for many enlightening conversations on a variety of topics in math, physics, and computer science.

A big thank you to James Jacobs for giving me the opportunity to be part of an exciting new project while being incredibly accommodating as I wrote this thesis.

Thank you to NSERC for enabling my research through its funding.

It is hard for me to imagine successfully completing this project without the love and support of my wife Gwen. Thank you for putting up with the long nights and for reminding me to eat, sleep, and enjoy the sunshine every so often.

And lastly, a big thank you to my family, whose support has meant the world to me, even from the other side of the continent.

In loving memory of my father, David Doran, to whom I owe my inspiration to pursue higher education.

Chapter 1

Introduction

Tetrahedral meshes have become an important tool in physics-based animation, particularly as discretizations of volumetric domains for fluid and elastic solid simulations. The task of creating a good-quality tetrahedral mesh describing an arbitrary domain is prohibitively difficult and time-consuming by hand, hence the motivation to create automatic mesh generators. However, high-quality tetrahedral mesh generation is very much a non-trivial task, and many different techniques have been tried over the years to solve this difficult problem.

The problem of high-quality tetrahedral mesh generation is essentially a conflict between two opposing forces: the need for a mesh to have tetrahedral elements of good quality, and the need for a mesh to adequately describe a volumetric domain, particularly the surface describing its boundary. The ideal, best-quality tetrahedron in most applications is the equilateral tetrahedron. Unfortunately, it is effectively impossible to mesh even relatively simple volumetric domains using

only equilateral elements, and as such it becomes necessary to compromise on quality in order to achieve domain conformance. In some applications, such as modeling of smooth biological tissues or boundary-embedded simulation methods, faithful recreation of the domain boundary is less important, allowing for additional quality gains. In others, such as Computer-Aided Design (CAD) or high-accuracy offline Finite Element Method (FEM) simulations, the domain must be modeled precisely, and element quality usually suffers as a result.

One of the most notable recent results in tetrahedral mesh generation is the isosurface stuffing algorithm of Labelle and Shewchuk [13]. Isosurface stuffing begins with a tetrahedral lattice covering the domain (the Body-Centred Cubic (BCC) lattice is used), then deforms or cuts elements to place vertices on the domain boundary. The algorithm is conceptually quite simple, and relatively easy to implement compared to many other meshing algorithms. Remarkably, it is nevertheless one of the few published tetrahedral mesh generators to give guaranteed bounds on element quality. Although the bounds are impressive, there remains room for improvement in the quality of elements produced by the algorithm.

The attractive results of isosurface stuffing come with a caveat; the input surface to isosurface stuffing is assumed to be smooth and of adequately thick shape (e.g. with minimum radius of curvature greater than or comparable to the spacing of the tetrahedral lattice). If this assumption does not hold, isosurface stuffing "rounds off" non-smooth features such as corners and ridges. For those applications where non-smooth features are required to be represented in the tetrahedral mesh, isosurface stuffing is unsuitable.

In this document, we present the work that we have undertaken to improve the isosurface stuffing algorithm. These improvements fall into two categories: improvement of the quality of elements produced by the algorithm, and extension of the algorithm to include non-smooth features in the resulting mesh.

In order to improve the quality of elements produced by isosurface stuffing, we replace the commonly-used BCC lattice with an acute tetrahedral tiling known as the A15 lattice. We describe the A15 lattice and our use of it in Chapter 3. In the case of generating a uniform tetrahedral mesh, the A15 lattice allows for large gains in the overall quality of the result. However, one of the strengths of the original isosurface stuffing algorithm using the BCC lattice is its adaptive meshing scheme. Chapter 4 describes our attempts to produce an adaptive scheme based on the A15 lattice that improves on the BCC-based adaptive scheme.

To resolve non-smooth features in input surfaces, we choose vertices in the mesh and snap them onto the features wherever possible. Care must be taken not to invert tetrahedra or degrade quality below acceptable levels. Our approach does not guarantee that all features will be resolved completely, but our experiments show that this procedure maintains very reasonable levels of quality while producing much better representations of non-smooth input surfaces. Our feature-matching algorithm is presented in Chapter 5, and our experimental results can be found in Chapter 6.

We begin in Chapter 2 with a discussion of previously published mesh generation techniques, focusing specifically on quality of results as a basis for comparison.

Chapter 2

Overview of Mesh Generation

2.1 Background and Motivation

The problem of tetrahedral mesh generation has been studied for several decades. Much of the preliminary work comes from the engineering literature, motivated by the need to automatically create automatic volumetric discretizations for use with the finite element method (FEM). More recently, with increased interest in simulation methods for physics-based animation, the topic has also been treated in the computer graphics literature. Some of the more common uses for tetrahedral meshes in computer graphics are simulations of fluids and deformable solid objects.

In order to assess and differentiate between the diversity of mesh generation methods, we look primarily at the quality of elements generated. Ease of implementation and flexibility are also important secondary considerations. We present

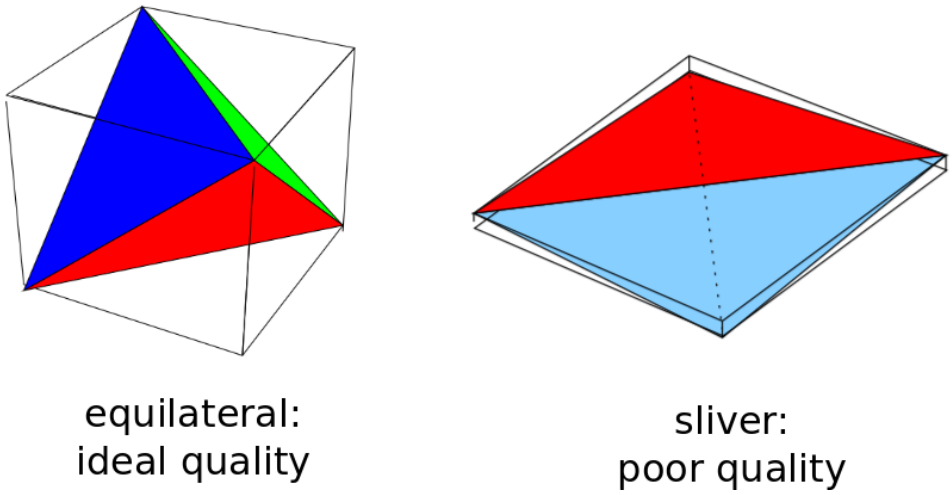
here a more detailed discussion of how we measure tetrahedral mesh quality, and then use this definition to analyze several the mesh generation methods most relevant to our own work.

2.2 Mesh Quality

It has been shown that the equilateral tetrahedron is the ideal, highest-quality tetrahedral element in the typical isotropic case. A thorough treatment of this subject, focusing on FEM, is given in Shewchuk [19]; we give a brief summary of relevant points here.

An equilateral tetrahedron, by its nature, has all edges of equal length, all altitudes equal (distance between a vertex and its opposite face), and all dihedral angles equal (angle between two faces sharing an edge). In fact, any of the above properties are by themselves sufficient conditions for the equilateral property, and imply the others to be true. Dihedral angles are a very popular and useful indicator of tetrahedron quality. All dihedral angles of an equilateral tetrahedron are equal to $\arccos(1/3) \approx 70.53^\circ$. As the maximum and minimum dihedral angles diverge further from this value, the tetrahedron becomes further from equilateral, and thus of lower quality. Figure 2.1 illustrates the difference between a high-quality and a low-quality tetrahedron.

Furthermore, small dihedral angles cause poor conditioning in finite element methods, slowing convergence to a solution of the partial differential equation in question. Even worse, large dihedral angles reduce the accuracy of the solutions produced. These are concrete examples of why tetrahedral quality is important,



Source: Si [21]

Figure 2.1: Comparison of equilateral and sliver tetrahedra.

and also motivate many to use dihedral angles to as their primary measure of element quality. Unfortunately, this makes it easy to adopt too narrow a focus in measuring quality, for example only maximizing the minimum dihedral angle of a mesh (e.g., Burkhart et al. [6]).

It is a much better approach to adopt a quality measure that indicates in a single scalar value how close to equilateral a tetrahedron is. One such measure is the aspect ratio of a tetrahedron, defined as the longest edge of a tetrahedron divided by its smallest altitude. As advocated by Shewchuk [19], we compute the aspect ratio using the signed volume of the tetrahedron, then invert it and normalize it with respect to the aspect ratio of an equilateral tetrahedron ($(\sqrt{2})^{-1}$), This gives a quality metric with a maximum of one for an equilateral tetrahedron, a value

of zero for degenerate tetrahedra with no volume, and a value less than zero for an inverted tetrahedron (with orientation opposite the expected orientation). This is a quality function that can be used as the basis for a maximization problem; when the quality measure is maximized over a mesh, we know that its elements are as close to equilateral as possible. Shewchuk [19] gives many different quality measures that can be used in this way, including aspect ratio. Our experiments use aspect ratio as the basis for quality measurement.

Note that, unless otherwise mentioned, the overall quality of a tetrahedral mesh is computed as the quality of its worst single element. A mesh can also be measured in terms of the average quality of its elements; however, the worst-element metric is generally preferred because in practice it correlates better with the accuracy and robustness of FEM simulations performed on the mesh. Mesh optimization is therefore treated as an exercise in improving the quality of the worst tetrahedron in the mesh.

2.3 Related Work

2.3.1 Delaunay Refinement

The well-known two-dimensional Delaunay criterion states that, for a given set of points on a plane, there exists a unique triangulation such that the circumcircle of every triangle contains no other points. In two dimensions, this has been shown to produce optimal triangulations. It is therefore natural to expect that its three-dimensional analogue, which uses the circumsphere of a tetrahedron, would be

similarly useful for producing three-dimensional meshes.

Arguably the most popular tetrahedral mesh generation software available today, TetGen, uses Delaunay refinement techniques [21]. Specifically, it uses a technique called Constrained Delaunay Tetrahedralization (CDT) to produce a tetrahedral mesh that conforms to a given input polygonal surface. The algorithm is also augmented with Steiner point insertions that place additional vertices into the mesh in order to improve quality. The overall algorithm is anything but trivial to implement, and we will not go into detail here; the interested reader should consult Si [21], Shewchuk [20], and Alliez et al. [1].

Perhaps the greatest strength of Delaunay refinement is its ability to faithfully reproduce an input surface in the output mesh. This ability comes from the fact that Delaunay refinement preserves the vertices and faces from the input (although sharp "needle-like" features can require special handling). It is also relatively easy to control the density of tetrahedra in different regions of the domain; more vertices are inserted in areas of higher density, while low density areas are left unrefined.

However, mesh quality is the area where Delaunay refinement falters. The key issue is that Delaunay refinement does not in fact optimize tetrahedra toward the ideal equilateral property. Instead, Delaunay methods minimize the circumradius-to-shortest-edge ratio of a tetrahedron, which admits a specific class of poor-quality tetrahedra known as slivers [20]. This has led to investigation of techniques for removing slivers from tetrahedral meshes, for example Cheng et al. [7] and Edelsbrunner and Guoy [9]. Furthermore, the fact that Delaunay

refinement preserves input vertices and faces means that the quality of the result is dependent on the quality of the initial vertex placement and face triangulations. As a result, Delaunay refinement offers no guarantees on element quality, and in practice cannot consistently be relied upon to produce high-quality tetrahedral meshes.

2.3.2 Advancing Front

Advancing front methods begin at the boundary of a given domain and incrementally insert vertices and tetrahedra further and further inside the surface until the domain has been completely meshed. For an example from the graphics literature, see Möller and Hansbo [16]. Like Delaunay refinement, this technique explicitly preserves the input surface in the result. The difficult step involved in this algorithm is the merging of "fronts" as they inevitably intersect inside the domain. It is this step that prevents advancing front methods from offering any guarantees of output quality. Advancing front is not particularly relevant to our work, so we will not discuss it further.

2.3.3 Octree Methods

Looking back into the engineering literature reveals a long history of meshing algorithms which work with a Cartesian grid structure, particularly quadtrees or octrees. Yerry and Shephard [28] first applied this technique in two-dimensions, encoding the input curve into a quadtree structure and then triangulating each quadtree cell. The approach was then applied to three dimensions by Yerry and

Shephard [29] and further refined by Shephard and Georges [18].

Algorithms of this class begin by encoding the input surface into an adaptive grid structure, i.e., an octree. This effectively converts the surface into a volumetric representation made up of cubical octree cells, plus information about the input surface in the octree cells that contain it. To achieve a tetrahedral mesh, each octree cell is meshed locally in such a way that adjacent cells conform to each other. Octree cells containing the input surface are treated specially, and meshed in such a way that the surface is represented in the output. The details of how each cell is meshed are what distinguish the different octree algorithms.

One of the more popular octree methods is the Finite Octree method, which takes great care to reproduce all features of the input surface, including non-smooth features [18]. Its ability to accurately reproduce CAD-style non-smooth models with relatively high quality elements has led to its adoption for engineering applications. Other notable highlights include provably optimal (up to potentially large constant factors) meshers by Bern et al. [2] in two dimensions and Mitchell and Vavasis [14] in three dimensions. Although octree methods offer great promise for element quality, in practice there is still significant room for improvement, both in terms of quality and in terms of algorithmic complexity.

2.3.4 Lattice Methods

A later class of methods that we will refer to as lattice methods can be thought of as spiritual successors to octree methods. Instead of starting with a Cartesian structure for the initial volumetric representation of the domain, a space-filling

tetrahedral lattice is used instead. This is a major breakthrough because it eliminates the need to do local meshing of octree cells; the domain is already filled with a tetrahedral lattice, and no additional work needs to be done on the interior of the domain. Only the tetrahedral elements intersecting the input surface need to be treated in order to resolve the boundary. In addition, better quality elements are possible because there is no longer any need to conform to the cubical cells of an octree.

Most of the literature exploring these methods deals primarily with smooth inputs, and are aimed at applications where it is either unnecessary to match every detail of an input surface, or it is assumed that the input surface has no sharp features. Velho et al. [25] began with the Freudenthal subdivision of a grid, and deformed it to match an implicit surface boundary. Molino et al. [15] introduced the BCC lattice, with red-green refinement used to achieve adaptivity. These methods accomplish the task of matching the input surface through deformation of the tetrahedra in the initial lattice.

The isosurface stuffing algorithm of Labelle and Shewchuk [13] makes the crucial observation that tetrahedra can also be cut in order to resolve the input surface. If lattice vertices lie close enough to the surface then they are warped onto the surface in the same way as previous methods. However, if an intersecting tetrahedron has no vertices close to the surface, new vertices are inserted that lie on the surface and the tetrahedron is cut according to a small number of context-specific stencils. This results in better quality tetrahedra at the boundary than can be achieved through vertex warping alone. In fact, isosurface stuffing gives prov-

able bounds on the quality of the resulting tetrahedra, making it one of the only tetrahedral meshing algorithms to guarantee high-quality elements, and likely the simplest conceptually to do so.

Isosurface stuffing forms the basis of our work on tetrahedral mesh generation, and as such we go into more detail on how we have implemented the method in Chapter 3. Our work focuses on starting from the already-impressive level of quality achieved by the algorithm and seeking to improve it even further. We also look to imbue isosurface stuffing with the ability to resolve non-smooth input features, a capability found in methods such as Delaunay refinement, but so far absent from any lattice methods.

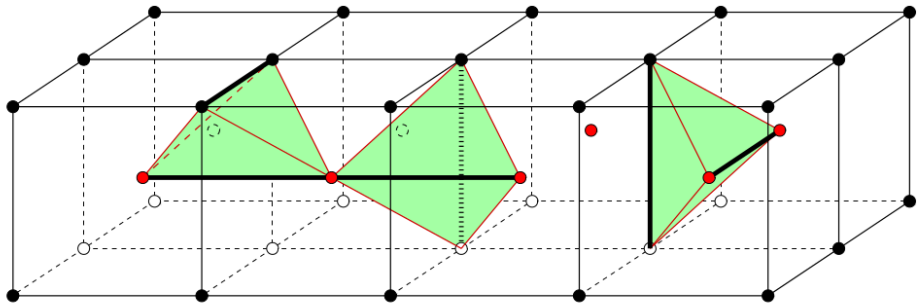
Chapter 3

A15 Isosurface Stuffing

3.1 Improving on the BCC Lattice

Probably the most popular tetrahedral lattice used in mesh generation today is the Body-Centred Cubic (BCC) lattice. The vertices of the BCC lattice can be thought of as two regular grids, one offset from the other by half the grid resolution [15]. This makes the BCC lattice relatively intuitive to visualize and work with. This same cubical, grid-like property that makes BCC nice to work with is also the source of its primary limitation in terms of element quality; all tetrahedra in the lattice have dihedral angles of 90°.

The nature of the isosurface stuffing algorithm is such that tetrahedra not near the isosurface are not deformed from their initial shape in the starting lattice. It therefore follows that choosing a lattice with higher-quality tetrahedra will result in generally better elements in the output mesh, particularly in the interior of the



Source: Labelle and Shewchuk [13]

Figure 3.1: Illustration of the BCC lattice.

volume. Moreover, starting with better-quality elements is expected to limit the degradation in quality caused by the deformation and cutting of tetrahedra near the boundary. Stated another way, a higher-quality initial lattice allows for more deformation before quality becomes too poor, a fact that is particularly important for our sharp feature-matching algorithm presented in Chapter 5. It is with this motivation that we look to replace the BCC lattice with a lattice composed of higher-quality tetrahedra.

3.2 TCP and Edge Valence Analysis

There are many space-filling tetrahedral tilings, of which BCC is only one choice [24]. The topic of space-filling geometric tiles has received much study in the literatures of both mathematics and chemistry in the form of geometric foams and crystalline molecular structures, respectively [22]. Of particular interest is the family of Tetrahedral Close-Packed (TCP) structures, since their structure is

composed of nearly-equilateral tetrahedra. By definition, the Voronoi cells of a TCP lattice have faces that are either pentagonal or hexagonal. As a consequence, all edges in a TCP lattice have valence five or six (in this context, the valence of an edge is the number of tetrahedra that share that edge).

It is worth noting at this point that it is not possible to tile three- dimensional space using only equilateral tetrahedra. As noted before, the dihedral angles of an equilateral tetrahedron are all equal to $\arccos(1/3) \approx 70.53^\circ$. If we consider a single edge on the interior of a tetrahedral lattice, it becomes apparent that the sum of the dihedral angles at that edge must be equal to 360°. However, because $360^\circ / \arccos(1/3) \approx 5.1$ is not an integer, it is not possible for an edge to be shared by all equilateral tetrahedra; at least one of the dihedral angles at the edge must be larger or small than the ideal $\arccos(1/3)$ in order for the sum to be 360°. The best we can do is either an edge of valence five, with an optimal maximum dihedral angle of $360^\circ/5 = 72^\circ$, or an edge of valence six, with an optimal minimum dihedral angle of $360^\circ/6 = 60^\circ$. To have edges of valence four or less is an immediate indication of poorer quality, since it is impossible to have a maximum angle less than 90° incident to a valence-four edge. Valences greater than six should also be avoided where possible due to the small dihedral angles they induce.

Edge valence analysis of the BCC lattice reveals that its 90° dihedral angles are the result of valence-four edges; even more strongly, it allows us to state that it is impossible to improve the quality of the BCC lattice further without changing its topology, since 90° is the optimal angle at a valence-four edge. Edge valence analysis can also be applied to other mesh generation techniques; the majority

of techniques do not take edge valences into account during execution, and as such are prone to producing edges with valences that are either too small or too large. This is an indication of poor-quality tetrahedra, and also of limits on quality improvement that can be achieved without significant topological modifications.

3.3 A15 Lattice

With the insight of edge valence analysis available to us, we can clearly classify the TCP lattices as superior alternatives to BCC in terms of structural quality; all edges in a TCP lattice have valence five or six, meaning that for a space-filling tile its tetrahedra will be as close to equilateral as we can reasonably expect. There are three basic TCP structures, differentiated by the specific arrangement and types of their Voronoi cells, known as A15, Z, and C15 [22]. All three have been observed in nature as molecular structures.

We have chosen the A15 lattice to use in our mesh generation algorithm. This choice comes from the fact that the vertices of A15 can be defined in terms of integral coordinates on a grid, making it easier to work with in practice. By choosing A15 we are also able to leverage the insights of Williams [27], who used the A15 tile for a "marching tets" isosurfacing algorithm.

The A15 tile has all dihedral angles between 53° and 79°, A rendering of the tile can be seen in Figure 3.2. making it a significant improvement over BCC. It is also locally Delaunay, which means that performing Delaunay refinement on the vertices of the A15 tile will reproduce the A15 tile exactly. It is not possible to improve the tile locally by means of vertex smoothing; the vertices are already at

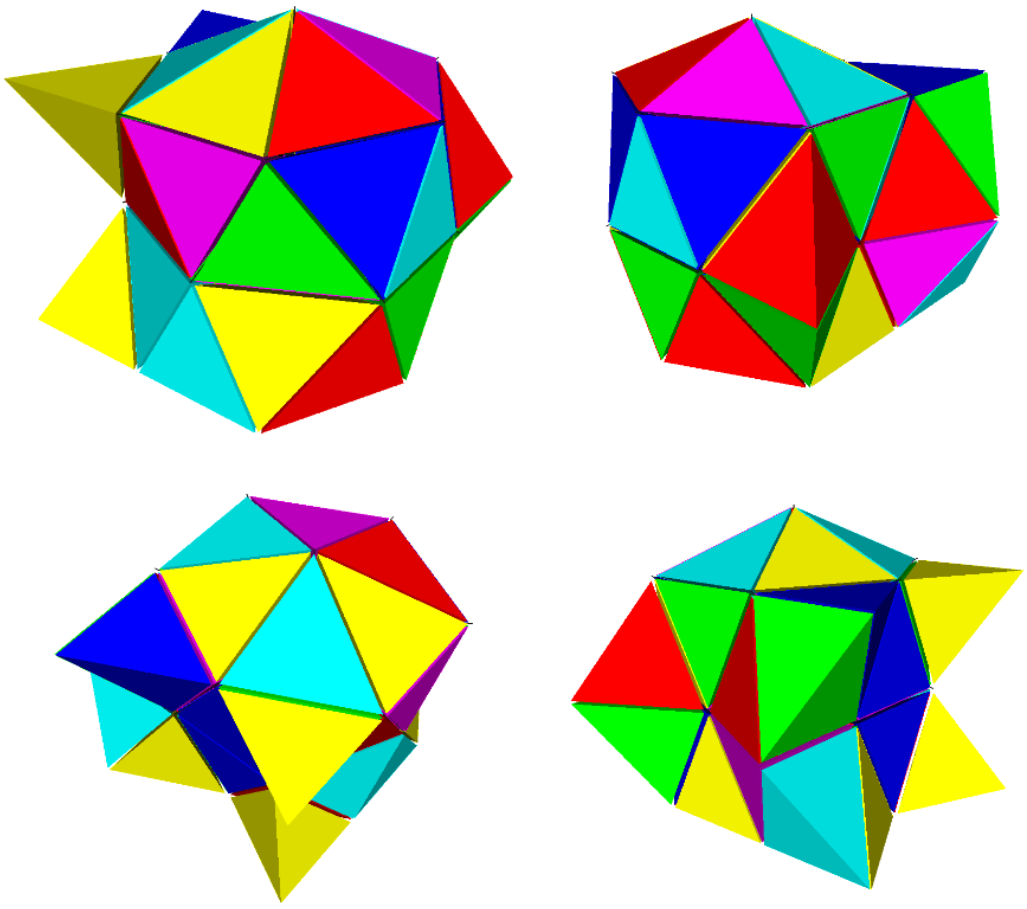


Figure 3.2: A single tile of the A15 lattice viewed from multiple angles.

local maximums for quality of their adjacent tetrahedra. A full description of the geometry of the A15 tile can be found in A.

3.4 Algorithm

Other than replacing the BCC lattice with the A15 lattice in the initial domain-filling step, our mesh generation algorithm is very similar to the original isosurface

stuffing algorithm of Labelle and Shewchuk [13]. We give a brief overview of our version of the algorithm here.

1. Convert the input to a Signed Distance Field (SDF) representing the domain boundary as a zero-isosurface (if it is not already given in this format). Most commonly, the input surface is given in the form of a triangle mesh; in this case we use the fast sweeping method described in Bridson [4] and Zhao [30] to convert the triangles to an SDF. We use the convention that negative distances indicate that a point is inside the domain, and positive distances are outside. The resolution of the SDF is given by a desired resolution parameter, h. It is understood that features of the input of finer resolution than h will be lost and not represented in the output.

2. Cover the volumetric domain with the uniform A15 lattice; this is accomplished by filling the domain's bounding box with the tetrahedral lattice, then removing all tetrahedra that lie completely outside the domain (i.e., have all four vertices with positive values in the SDF). The size of the tetrahedra corresponds to the resolution of SDF, h, so that the isosurface information encoded in the SDF can be sampled accurately with no aliasing artifacts.

3. All edges of the lattice that intersect the isosurface are found and added to the set of "violated" edges. The point along the edge where the intersection occurs is denoted as the "cut point" of the violated edge. We measure the distance of the cut point to the endpoints of the violated edge as a ratio of

the total edge length. These distance ratios determine how the violated edge is handled.

4. If the cut point of a violated edge occurs within a certain distance ratio threshold α of one of the edge's endpoints, then the violated edge is resolved by warping that endpoint to the cut point. The warped vertex now lies on the isosurface, and any other violated edges that contain that vertex are marked as resolved as well. All vertex warping operations are performed before moving to the next step.

5. All remaining violated edges now have cut points that are not close to either endpoint of their violated edge. These violated edges are resolved by performing a cutting operation on the tetrahedra that contain them, adding the cut points to the mesh in the process. Tetrahedra are cut according to a stencil determined by the number and relative placement of cut points on their edges. The stencils ensure that the resulting tetrahedra are conforming and create a well-formed mesh. Unlike the BCC-based isosurface stuffing algorithm, we do not have a concept of "red" and "black" edges in the A15 lattice; we consider all edges equally, so fewer stencils are needed.

At the completion of the final step, no more violated edges remain, and the tetrahedral mesh has been warped and cut so that its boundary conforms to the isosurface of the domain boundary. Figure 3.3 illustrates the effect of the process. As noted previously, non-smooth features and features under-resolved by the SDF are not reproduced.

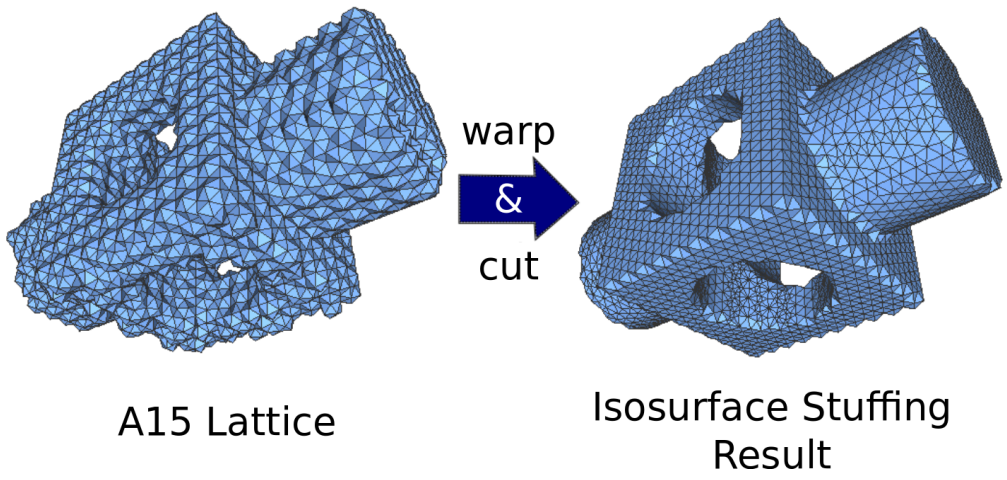


Figure 3.3: A15 Isosurface stuffing in action.

One of the biggest successes of BCC-based isosurface stuffing paper was the derivation of worst-case quality bounds for the resulting mesh. The bounds vary depending on the metric being optimized and other algorithm parameters, but most guarantee that all dihedral angles will fall within the range of approximately 9° to 160° [13]. These bounds are obtained by a computer-assisted proof. We leave as future work the task of deriving similar bounds for the A15-based version of isosurface stuffing. For the time being, we rely on our earlier assertion that a higher-quality initial lattice will improve the final resulting mesh. At the very least, we expect that the average element quality will improve significantly, because the high-quality A15 tetrahedra on the interior of the domain are left unchanged by the algorithm. We provide representative experimental results in Section 6.1.

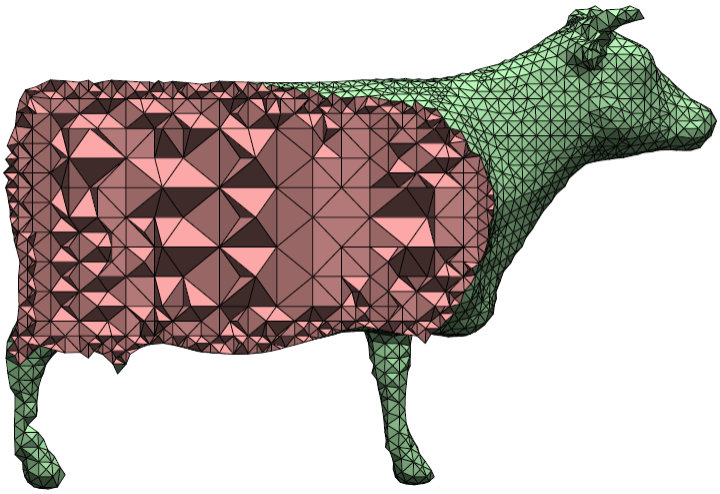
Chapter 4

Adaptivity

4.1 Background

It is generally desirable in mesh generation to have a finer resolution of elements in areas of interest of the domain, including the domain boundary. This adaptive meshing approach uses fewer elements in areas where detail is not required, thus saving both space and time when operating on the mesh. The original isosurface stuffing algorithm includes an adaptive meshing scheme based on the BCC lattice. In Chapter 3, we presented an improved uniform isosurface stuffing algorithm using the A15 lattice; a natural next step, therefore, is to produce an analogous A15-based adaptive meshing scheme.

Adaptivity in isosurface stuffing is achieved by substituting an adaptive lattice for the uniform lattice in the first step. The original algorithm of Labelle and Shewchuk [13] uses a customized octree structure to construct a graded lattice



Source: Labelle and Shewchuk [13]

Figure 4.1: Adaptive meshing using BCC lattice.

based on BCC. The lattice uses BCC tiles in areas of uniform resolution; only in the gaps between these regions are modifications made in order to "bridge" the gap between tiles of different resolutions. These bridge tetrahedra are generated algorithmically based on the topology of the octree at interface between cells of different sizes. The result gives an adaptive lattice with all dihedral angles between 45° and 120°. Crucially, the lattice is generated such that the input surface only intersects BCC tetrahedra. This means that the same proofs of quality bounds used with the uniform lattice are applicable here, because the warping and cutting operations will operate on the same tetrahedra in both cases. Figure 4.1 shows an example of the BCC adaptive lattice used for isosurfacing.

We have already demonstrated that the A15 lattice outperforms the BCC lattice

in terms of quality in the uniform case. Unfortunately, we have been unable to find any mention of adaptive variants of TCP structures in the literature, and the creation of an adaptive lattice reflecting the quality of A15 appears to be an open problem. In the absence of useful mathematical theory or examples from nature, adaptive meshes are generally achieved by methods such as octree algorithms [18, 29], subdivision operations [15, 23], heuristic remeshing operations [26], or some combination of all these.

The remainder of this chapter documents our efforts to produce an adaptive A15-based tetrahedral lattice using these and other varied methods. As described in Section 3.2, we use edge valences as our primary measure of whether a particular lattice emulates the structural quality of A15. The ideal lattice has only edges with valence five or six. Edges with valence four are considered undesirable but acceptable due to how commonly they occur in tetrahedral subdivision schemes. Valences greater than six are similarly undesirable but acceptable, but valences of three or less are considered unacceptable due to the large dihedral angles they induce.

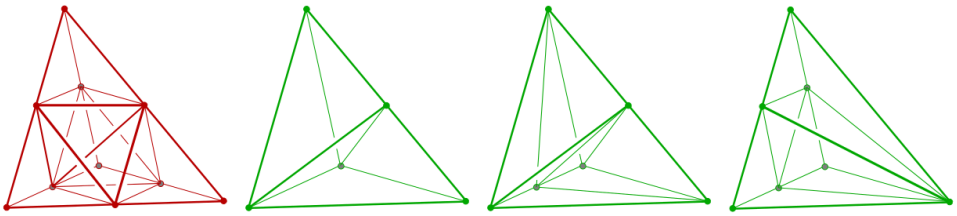
4.2 Tetrahedral Subdivision

Subdivision schemes are a classic method for increasing and decreasing resolution in geometric applications. In volumetric subdivision an initial cell (or small set of cells) is recursively split into smaller cells until a certain resolution threshold is met. In the case of adaptivity, the desired resolution threshold varies spatially. Adaptive subdivision is much more difficult for meshes than for octrees, how-

ever, because the regions between cells of different resolutions must be handled specially to ensure conformance of the elements of the mesh. For this reason, uniform subdivision schemes are much easier to devise than adaptive ones.

Probably the most popular tetrahedral subdivision scheme is red-green refinement [3, 8]. The algorithm is so-named because it consists of regular "red" subdivisions, performed to increase resolution, and irregular "green" subdivisions, performed to preserve conformity of the mesh. Red subdivisions divide one tetrahedron into eight by inserting a new vertex on each edge, splitting the tetrahedron into four smaller tetrahedra (one at each corner), and a central octahedron. There are three choices of diagonal to split the octahedron into four tetrahedra, and the choice can affect the quality of the resulting tetrahedra. Regardless of the choice of diagonal, red subdivision can be performed uniformly on adjacent tetrahedra and give a conforming result. However, unsubdivided neighbours of red tetrahedra cause three-dimensional T–junctions that need to be resolved with green refinement. Figure 4.2 shows the different tetrahedral subdivisions used in the red-green scheme.

Molino et al. [15] use red-green refinement on an initial uniform BCC lattice to generate an adaptive BCC-based lattice. It seems simple enough to take the same approach but substitute in an A15 lattice. Unfortunately, red-green refinement produces valence-four edges, which severely undermines the structural quality of A15 (this is not a problem with the BCC lattice, which already has valence-four edges). As a result, successive refinements yield fine-resolution tetrahedra that do not retain the high-quality properties of the initial A15 lattice. For this reason,



Source: Molino et al. [15]

Figure 4.2: Red-green subdivision stencils.

we did not pursue red-green refinement of an initial A15 lattice as an adaptive meshing scheme.

There are other, lesser-known tetrahedral subdivision schemes in the literature. One such scheme comes from Burkhart et al. [6], which seeks to produce a volumetric analogue to the well-known $\sqrt{3}$-subdivision scheme of Kobbelt [12]. However, analysis of this subdivision scheme shows that it produces valence-three edges, which we consider unacceptable. We note that the quality of the published results of this method are measured in terms of minimum dihedral angle; the inevitably large maximum dihedral angles that result from such low-valence edges are not mentioned. Because this method seems not to have been designed with the same definition of tetrahedral quality that we use, we do not pursue it further.

After exhausting the literature searching for a tetrahedral subdivision scheme that would preserve or emulate the quality of the A15 lattice, we attempted to design our own. We analyzed numerous prototypes based on combinations of edge subdivisions, face subdivisions, and point insertions on the interior of the

tetrahedron. The following criteria were used for acceptance of a scheme:

1. The scheme must be able to tile uniformly (i.e., the "red" case).

2. The scheme must have finite propagation in the adaptive case (i.e., the "green" case).

3. The scheme must produce no edges with valence less than five.

The first two requirements are necessary of any adaptive tetrahedral subdivision scheme. The final requirement is specific to our goal of achieving high-quality tetrahedra, and is the property by which a scheme may be judged superior to standard red-green refinement.

Despite several prototypes that could satisfy up to two of the criteria listed above, we were unable to come up with a subdivision scheme that satisfied all three. We do not claim to have performed a full search of the space of possible subdivision schemes; we leave this, as well as exploration of the possibility that no such scheme might exist, as future work. For the time being, we conclude that standard red-green refinement is the best tetrahedral subdivision scheme in terms of tetrahedral quality, and turn our focus to other methods of creating an adaptive lattice.

4.3 A15 Point Lattice

As mentioned in Section 3.3, the A15 lattice is locally Delaunay. Thus, given a set of points located at the vertices of the A15 tile, performing Delaunay tetrahedralization will reproduce the A15 lattice. This insight forms the basis of our next

adaptive meshing algorithm, in which we generate an adaptive point lattice based on A15, then find its Delaunay tetrahedralization.

Before describing the algorithm, we note that it depends on the ability to overlay an A15 tile onto an octree cell. The version of the A15 tile we use (the listing of which is in A) has vertices at the corners of the cube with side-length four. To overlay the tile, we scale and translate it such that those corner vertices align with the corners of the octree cell. It is also important to note that we establish a mapping between the edges and faces of the octant to the corresponding edges and faces of the A15 tile. This allows us to, for example, determine which vertices are shared by two A15 tiles overlaid on adjacent octree cells. An illustration of this mapping can also be found in A.

We begin by generating an octree representation of the input surface. The octree is then balanced so that adjacent cells only differ in depth by at most one level. This is done to limit the complexity of the problem of bridging the gap between regions of different resolutions. We then iterate over all cells in the octree from smallest to largest. Each octree cell is filled with the vertices of an A15 tile; however, the vertices corresponding to any octree corners, edges, or faces that have already been visited are omitted to prevent duplicate and conflicting points. The result of this procedure is an adaptive A15 point lattice, an example of which can be seen in Figure 4.3. Finally, we use TetGen [21] to compute the Delaunay tetrahedralization of the point lattice to get the adaptive tetrahedral lattice.

This algorithm succeeds at creating an adaptive conforming tetrahedral mesh, and it also succeeds for the most part at replicating the structure of the A15 lat-

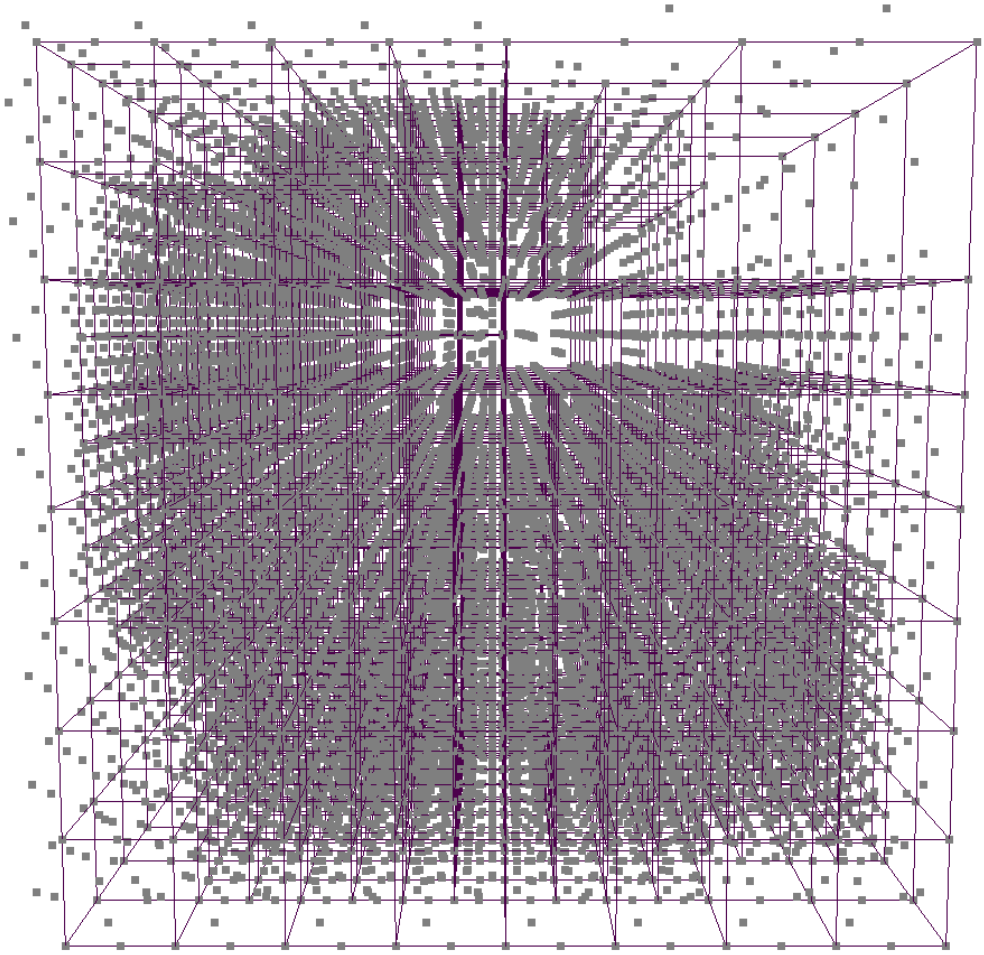


Figure 4.3: A15 point lattice (and the octree from which it was generated).

tice in regions of uniform resolution. Unfortunately, in our experiments with the method the "bridge" regions connecting regions of different resolutions exhibit very poor quality. Our results are presented in Section 6.3. The limitations of Delaunay refinement methods arise here, as the method does not optimize for equilateral tetrahedra, nor are any guarantees made about edge valences. Indeed, we observed frequent occurrences of valence-three edges in the meshes produced

by this method, as well as many poor-quality sliver tetrahedra. There is room for further work in this direction, such as more sophisticated point lattice generating techniques and post-processing operations for improving the quality of a Delaunay mesh. We chose to focus instead on a hybrid strategy for adaptive lattice generation, detailed below.

4.4 Adaptive Red-Green A15

The final adaptive lattice generation algorithm we tried is a hybrid of octree and subdivision methods. Observing that the poor quality in the A15 point lattice method described above occurs in the "bridge" regions of the adaptive lattice, we sought to design an algorithm that would give more control over the quality of the triangulation of those regions. We begin as before with an octree, and label all cells with subdivided neighbours as "bridge" cells. All non-bridge cells are filled with the standard A15 tile. The bridge cells must then be treated specially in order to conform to the A15 tiles of different resolutions surrounding them.

We achieve this by noting that the faces of one side of the A15 tile can be transformed to the faces of a finer-resolution set of four A15 tiles by the following two operations:

1. Subdivide each face 1-4.

2. Translate vertices to match the finer-resolution tiles.

3. Flip the long edges that result in the middle of the tile.

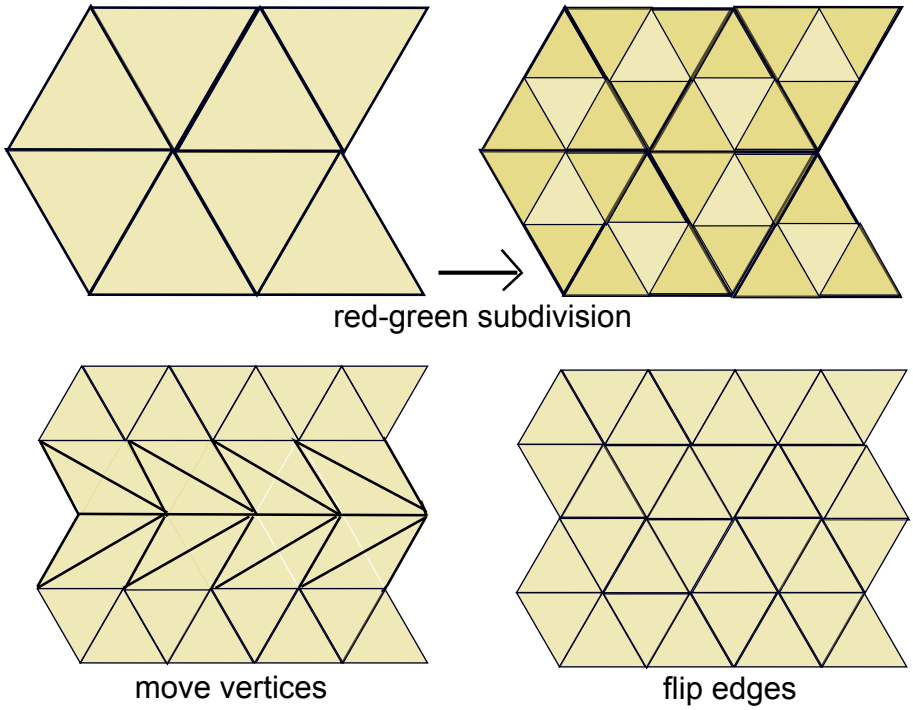


Figure 4.4: Transforming one face of A15 tile to double resolution.

By performing three-dimensional analogues of these operations, an A15 tile can be made to conform to the smaller A15 tiles overlaid on adjacent subdivided octree cells. See Figure 4.4 for an illustration of this process on a face of an octree cell. Therefore, we overlay A15 tiles over all bridge cells, merge all bridge cells of the same size together into a single tetrahedral mesh, and then perform the above steps on each bridge mesh to make the boundaries conform to their neighbouring A15 tiles.

The subdivision step is performed using red-green subdivision as described above. In this case, the valence-four edges caused by this scheme are considered acceptable because they will only occur in the bridge regions; the uniformly high-

resolution regions in this algorithm are meshed directly with the A15 tile. All faces that require subdivision are marked as being adjacent to virtual red-refined tetrahedra. The subdivision scheme is then propagated through the mesh as in Bey [3] (most tetrahedra whose faces require subdivision will be marked green, though some may require a full red refinement).

The edge flipping operation depends on the valence of the edge to be flipped. If the edge is of valence two, the 2-2 face swapping operation found in Freitag and Ollivier-Gooch [10] is used (recall that all edges to be flipped are boundary edges). If the edge is of valence three, we first insert a tetrahedron consisting of the existing edge and the "flipped" edge, effectively inserting the desired edge into the mesh. We then do a 4-4 edge flip as described in Klingner and Shewchuk [11] around the original edge, choosing the new diagonal so as to maximize quality of the adjacent four tetrahedra.

Once all the required edges have been flipped, vertices are translated to match smaller adjacent A15 tiles. At this point the bridge mesh should be able to conform with all adjacent tiles. Figure 4.5 shows the a mesh at this stage of the algorithm. Once all bridge meshes have been processed, all tiles can be merged together into the final mesh.

By itself, this procedure produces a conformal adaptive mesh with A15 tiles in the regions of uniform resolution. It does not, however, produce good quality elements in bridge regions. This is largely due to the final vertex translation step, which can cause the tetrahedra created through subdivision to deform substantially. We have no choice but to move the boundary vertices in this way to achieve

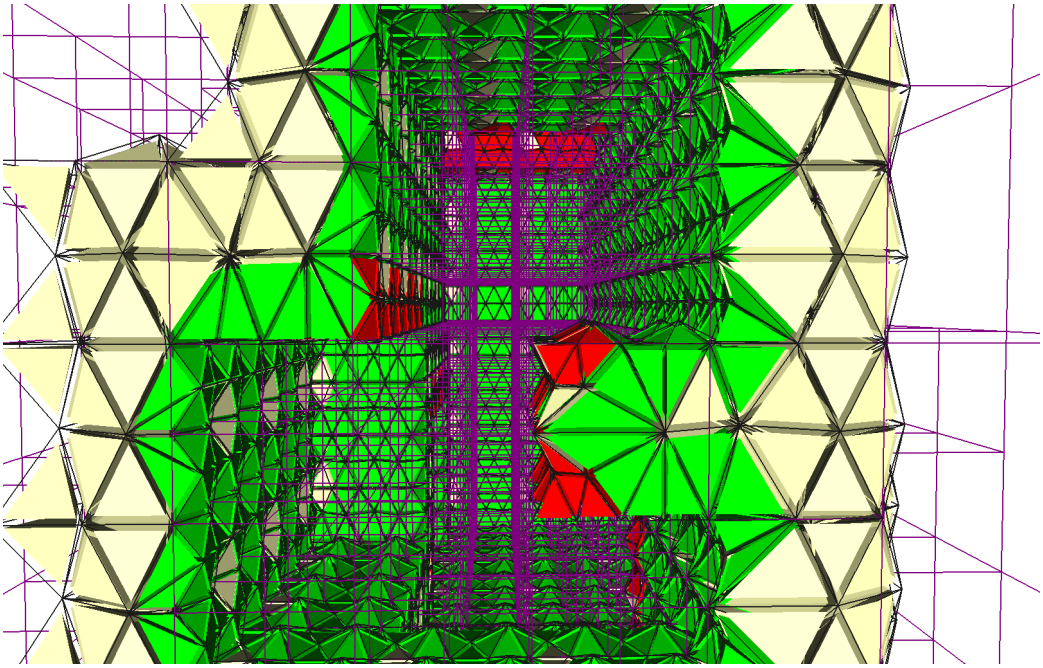


Figure 4.5: Subdivided bridge cells in the red-green A15 algorithm.

conformance with adjacent tiles; however, interior vertices of the bridge mesh are not constrained in this way. As such, we introduce a vertex optimization step to improve the quality of the result. We use a simplified version of the technique described in Freitag and Ollivier-Gooch [10], doing a steepest-descent search to find a new location for each unconstrained vertex. The search terminates whenever the worst adjacent tetrahedron is improved to be the same quality as at least one other adjacent tetrahedron. We use the inverted aspect ratio quality measure introduced in Section 2.2 and compute the gradient using the second-order central difference method.

Even with the introduction of vertex optimization, our results consistently contained poor-quality tetrahedra, leading us to introduce additional mesh improve-

ment techniques. We examine all valence-four edges of the mesh, performing a 4-4 edge flip wherever this would improve quality. We also insert a Steiner point in place of a valence-four edge when doing so would improve quality, a step motivated by the fact that situations can arise where all possible choices of diagonal in a 4-4 edge flip yield degenerate or inverted tetrahedra.

These steps combined with vertex optimization are able to produce adaptive A15 meshes of reasonable quality for relatively simple input surfaces. Unfortunately, however, our experiments show that more complex inputs can cause our algorithm to create degenerate or very poor-quality tetrahedra. Even in the simple cases, the quality of the adaptive mesh does not exceed the quality of the adaptive BCC lattice of Labelle and Shewchuk [13]. Our results are presented in Section 6.3. We postulate that the formulation of our algorithm overconstrains the vertices and topology of the mesh, leading to the lack of quality despite numerous optimization steps. Future work would see exploration of additional alternative strategies based on conversion of octrees into A15-based meshes. For now, we conclude that the highest-quality adaptive lattice available is the BCC-based adaptive lattice.

Chapter 5

Feature Matching

5.1 Algorithm

Our second avenue of improvement to isosurface stuffing is the inclusion of feature matching operations that attempt to include sharp corners and creases of the input surface in the output mesh. Standard isosurface stuffing guarantees accurate matching of the smooth regions of the input surface, while rounding off sharp features. Our approach is to start with the smoothed-off mesh, and then resolve the sharp features that have been missed by moving vertices onto them. We will refer to this process as "snapping," and vertices that have been moved onto a feature are referred to as "snapped vertices."

In addition to the usual signed distance field representing the surface we wish to reproduce, we include as input a list of curves and points indicating sharp features on the surface. Figure 5.1 shows the set of sharp features we used with the

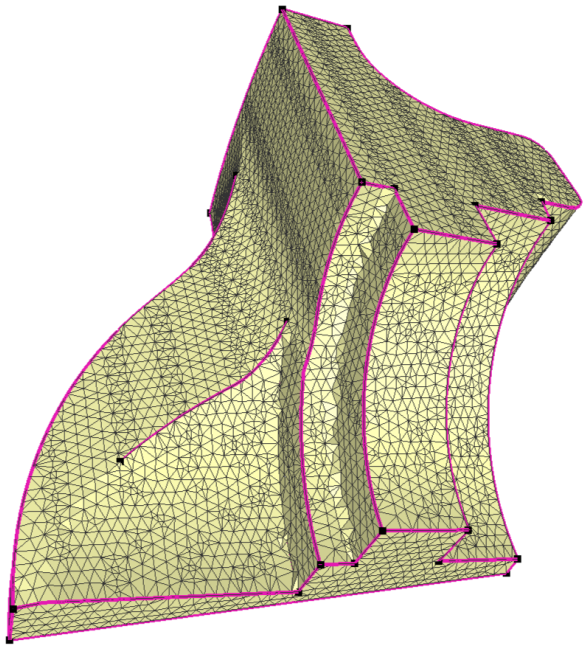


Figure 5.1: Feature points (black) and curves (pink) of the Fandisk model.

Fandisk model. Our feature-matching meshing algorithm can be summarized as follows:

1. Perform isosurface stuffing using the signed distance field to produce an initial tetrahedral mesh that matches the input except at sharp features.

2. For each endpoint of a feature in the input, find the closest vertex on the boundary of the mesh and snap it to the feature endpoint.

3. For each feature curve, find a path through the boundary of the mesh between the vertices now situated at each of the feature's endpoints. Snap each vertex along the path onto the feature.

4. Perform a round of vertex smoothing on the mesh to restore element quality. Vertices moved onto features are constrained to those features, and vertices on the mesh boundary are constrained to lie on the isosurface.

As before, we assume an isotropic domain so that the optimal tetrahedron is equilateral with all dihedral angles equal. We use the normalized, inverted tetrahedral aspect ratio from Shewchuk [19] as our quality measure for the purposes of vertex snapping and smoothing. Signed volume and orientation of tetrahedra are computed using Shewchuk's geometric predicates [17] to ensure accuracy and robustness.

Figure 5.2 shows a before-and-after comparison of the Fandisk model meshed with and without our feature matching algorithm.

5.2 Feature Endpoints

Assuming that the initial tetrahedral lattice used is of a sufficiently high resolution, the closest vertex to each feature endpoint should be unique; in other words, a mesh vertex should not be the closest vertex to more than one feature endpoint. Thus, step (2) of our algorithm is very simple: Find the closest mesh vertex to each feature endpoint and snap it to that feature endpoint.

Practically speaking, if two or more feature endpoints do end up in close enough proximity that they are competing for the same vertex, two options are available. First, the algorithm may choose to fail gracefully, requesting that additional resolution is required to correctly resolve the given features. Alternatively, the offending feature endpoints can be "merged" into a single endpoint, and the

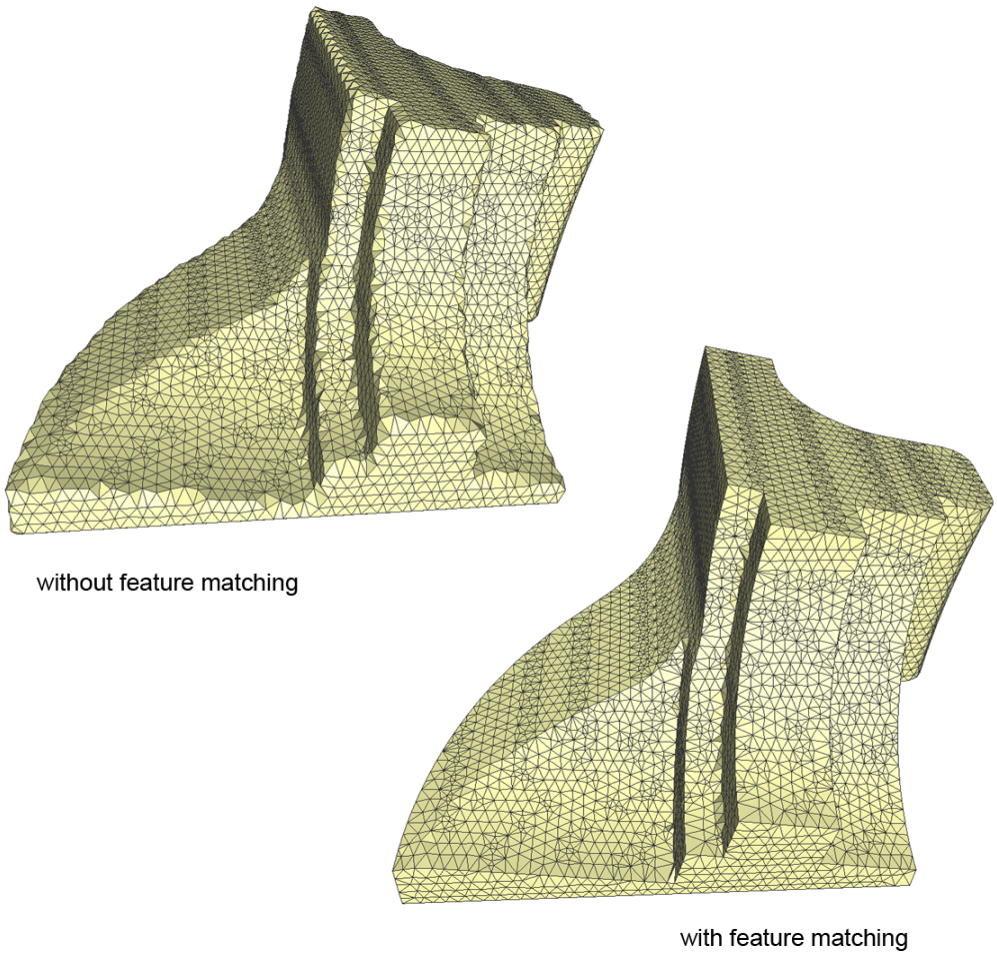


Figure 5.2: Fandisk meshed with and without feature matching.

closest vertex snapped to that merged endpoint. That vertex would then be treated as the de facto endpoint of the feature for the remainder of the algorithm's execution, even though it may no longer be at the same location as the endpoint given as input.

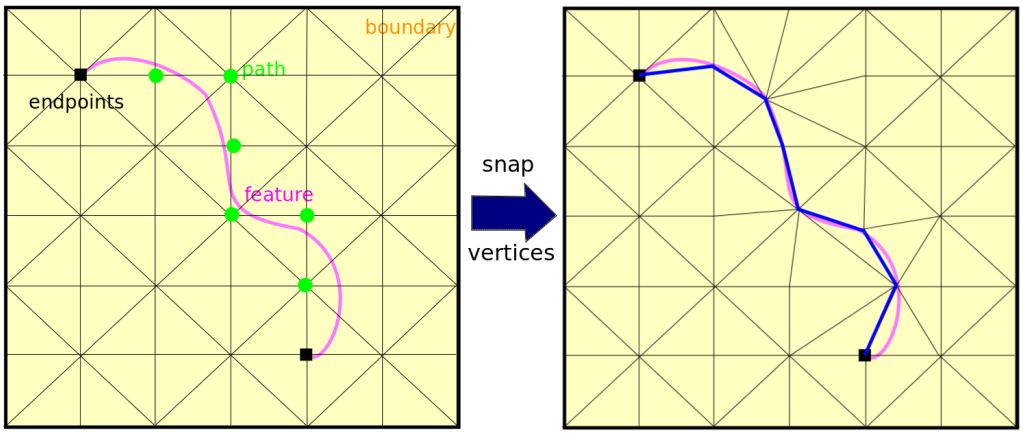


Figure 5.3: Example of feature curve matching.

5.3 Feature Paths

Next, we find paths through the boundary of the mesh between the endpoints of each feature. Each vertex along this path is snapped onto the closest point on the feature to that vertex. The final result is that the vertices and edges of the path follow the feature, thus including it in the output tetrahedral mesh. Optimally, the path chosen to snap to the feature should follow the feature's shape as closely as possible. We can accomplish this by treating the problem as a shortest-path problem. First, set the weight of each vertex to be the distance from the vertex to the feature. The problem is now to find the path that minimizes the sum of vertex weights, which can be solved using Dijkstra's algorithm. This process is illustrated in Figure 5.3 as a two-dimensional view of a feature curve and the nearby mesh boundary.

It is worth noting, however, that finding the optimal path is not as simple if

we wish to avoid creating low-quality tetrahedra. For example, snapping three vertices that are part of the same tetrahedra onto a feature that is a straight line instantly makes that tetrahedron degenerate. Furthermore, from this scenario we can see that the decision to add a candidate vertex to a path in the search space will depend on the previous vertices of that path. A vertex that inverts a tetrahedron when included in one path may be perfectly acceptable when included in a different path. This dependence between previous path and vertex admissibility is not handled in traditional shortest-path graph algorithms, such as Dijkstra, that guarantee the optimal result.

However, in this application we do not believe that optimality of the path is a necessary condition for success; any path that is sufficiently close to optimal and does not reduce element quality too much will serve. As such, an algorithm such as greedy best-first search is probably enough in most cases. Our implementation uses a modified Dijkstra algorithm that does not add vertices to the search front if adding them would deform a tetrahedron excessively. This does not guarantee optimality, but has proven to give acceptable quality in our experiments.

Our algorithm always seeks to find a close-to-optimal feature-following path between endpoints that does not reduce tetrahedron quality below a certain threshold, and also does not include vertices that have already been snapped to other features. This can be a very restrictive set of constraints on the path-finding search algorithm, and will often fail for non-trivial inputs. If it does fail, we try again while relaxing the constraint that the path must not use any vertices that have already been snapped. If this also fails, we try one last time with the tetrahedron

quality constraint removed. If all attempts to find a path fail, the feature is declared unresolvable given our constraints, and we move on to the next feature. If a path was found, we snap each vertex on that path to the feature. If we had to relax our constraints to find the path, however, not every vertex will be admissible to snap; vertices already snapped to other features are not snapped again, and any snapping operation that would degrade tetrahedron quality below a given threshold is not performed. This quality threshold should at minimum prevent snapping operations that would invert tetrahedra; this is the threshold used in our experiments. More conservative thresholds could prevent snapping operations that would increase the aspect ratio of a tetrahedron above, for example, 25.

We note that a major limitation of our algorithm appears at feature endpoints shared by multiple features. The surface valence of the mesh vertex snapped to a particular endpoint is bounded by the nature of the isosurface stuffing process, and has no relationship to the number of feature curves incident to that point. It is easy to come up with example inputs where the number of incident features to a particular endpoint exceeds the valence of the closest vertex, thus making it impossible to resolve all the those features fully without changing the topology of the mesh. Even in cases where the valence of the vertex is greater than the number of incident features, the specifics of a particular scenario and the order in which features are resolved can easily lead to "stranded" features with no unused vertices left to form a path to the endpoint. Features can also be stranded due to the minimum quality threshold imposed on the algorithm. Areas with high density of features are where tetrahedron quality tends to degrade the furthest,

and such areas occur naturally around endpoints shared by multiple features. This has led us to observe many features that are perfectly resolved on their interiors, but missing one or two edges right beside their endpoints. We leave it as future work to explore methods to change mesh topology to fully resolve features at high-incidence feature endpoints.

5.4 Mesh Smoothing

Once features have been resolved as well as possible given our quality and topology constraints, we perform mesh smoothing operations to try and improve the quality of the mesh elements as much as possible. Any schedule of mesh optimization operations will do, but our experiments have shown that a simplified local vertex smoothing operation generally suffices. The most important consideration is that vertices be correctly constrained during the optimization process; vertices snapped to features must remain on those features, and vertices on the boundary should be constrained to remain on the isosurface.

The smoothing algorithm we implemented is not particularly sophisticated. A small cube around the current vertex position is created with side lengths proportional to the tetrahedral mesh resolution. A random sampling of points within this cube are then selected and projected onto the constrained space of the vertex. The constrained space is described in more detail below. For each sample point, the quality of each tetrahedron containing the current vertex is calculated with the current vertex moved to the sample point. The sample point that yields the best local tetrahedra is chosen. Finally, if moving the vertex to the chosen sample point

improves the quality of its incident tetrahedra, then the vertex is moved. This local optimization scheme ensures that no tetrahedron is ever made worse by vertex smoothing, an approach referred to as "smart" smoothing by Freitag and Ollivier-Gooch [10]. Smoothing operations are repeated until either mesh quality ceases to improve or a fixed maximum number of iterations is reached.

For a boundary vertex each sample point is constrained to the isosurface. This is done by first finding the numerical gradient of the signed distance field at the vertex, and using it to project each sample point onto the plane tangent to the isosurface at the vertex position. We then compute the gradient of the signed distance field at each sample point and do a line search along that gradient until a point at or near the isosurface is found. If the numerical gradient is zero at the sample point, it is discarded. A vertex snapped onto a feature is similarly constrained to that feature. Sample points are first projected onto the line tangent to the feature at the vertex position, then projected onto the feature itself. Vertices snapped to feature endpoints are the easiest case; they are fully constrained and should not be moved at all.

Our mesh smoothing algorithm was designed for simplicity of implementation rather than efficiency or effectiveness. We are confident that any number of more sophisticated mesh improvement algorithms, such as those used by Freitag and Ollivier-Gooch [10] and Klingner and Shewchuk [11], could be used with even better results. The key insight from our experiments, however, has been that snapping vertices onto sharp features degrades element quality but preserves the inherently well- structured mesh produced by isosurface stuffing. Therefore, it

only requires very simple mesh smoothing operations to restore element quality back to acceptable levels.

Chapter 6

Results

6.1 Uniform A15 Isosurface Stuffing

We tested our algorithms on a small set of inputs that included both smooth surfaces and surfaces with sharp features. Each surface began as a triangle mesh and was then converted to a signed distance field using the fast sweeping method described by Zhao [30]. Each input was meshed using several different lattice resolutions (h) for the sake of comparison. We have provided representative results of these tests below.

We first tested our version of isosurface stuffing augmented with the uniform A15 tetrahedral lattice. Table 6.1 shows mesh quality results for smooth inputs that require no feature matching. We provide minimum and maximum dihedral angles, as well as the maximum aspect ratio. In each case our results show acceptable angle bounds for all of our inputs, with no dihedral angles smaller than

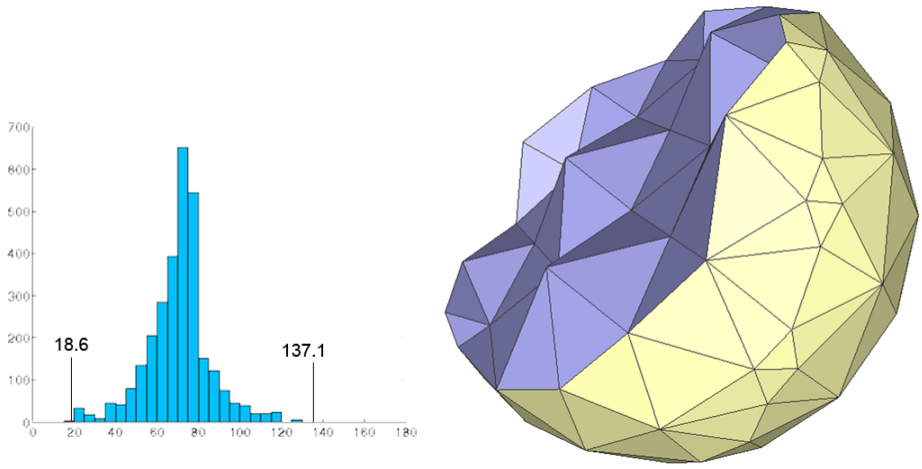


Figure 6.1: Cutaway view of Sphere input, meshed with $h = 0.2$.

Table 6.1: Mesh quality for smooth inputs

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time
Sphere	0.2	19.90°	126.12°	4.78	0.001s
Sphere	0.1	18.59°	137.11°	6.09	0.01s
Bunny	3.0	13.47°	151.42°	9.42	0.03s
Bunny	1.0	14.34°	147.33°	8.73	0.67s
Dragon	0.02	13.94°	148.46°	9.70	0.04s
Dragon	0.01	13.35°	150.29°	10.51	0.33s

13° and none larger than 152°. It is interesting to note that higher resolutions do not necessarily result in better-quality tetrahedra.

The Dragon model is also used by Labelle and Shewchuk [13] in their experiments, allowing for a direct comparison between our methods. The Dragon model meshed with a uniform BCC lattice gives a minimum dihedral angle of 14.9° and a maximum dihedral angle of 157.5° (aspect ratio data is not provided). Using the A15 lattice, we produce a mesh with minimum dihedral angle 13.94° and max-

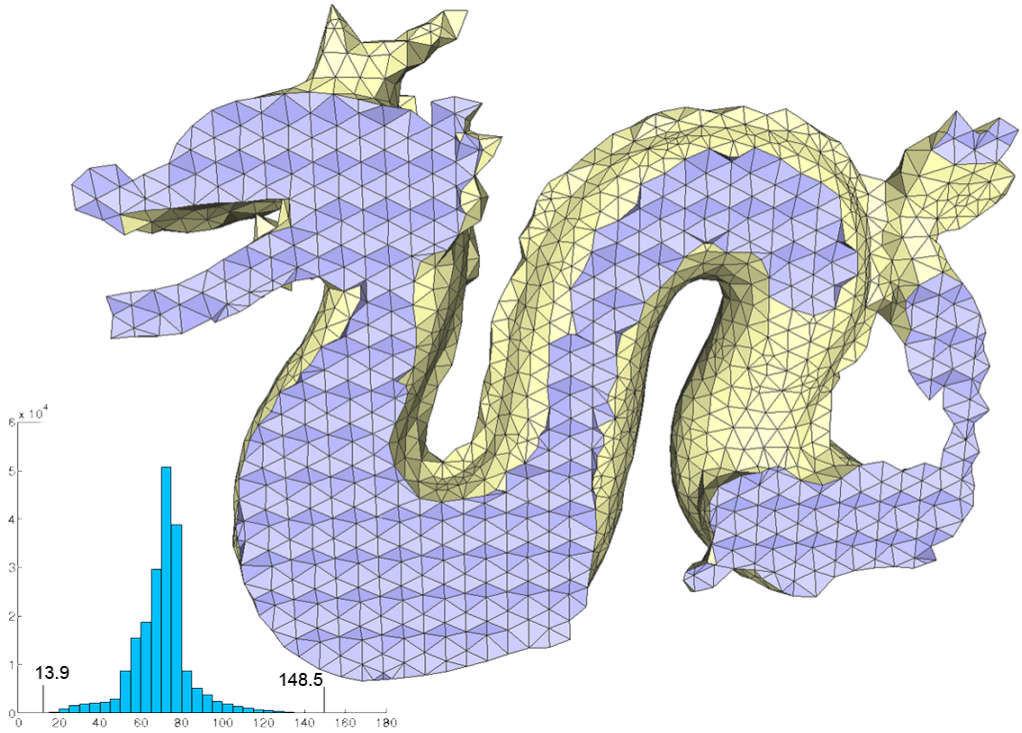


Figure 6.2: Dragon input, $h = 0.02$. with dihedral angle histogram

imum dihedral angle 148.46°, an improvement of ten degrees on the maximum dihedral angle. A more marked improvement is observed in the dihedral angle histograms of the two meshes; the A15 dragon (seen in Figure 6.2) has the vast majority of its angles between 55° and 80°, whereas the BCC dragon has much wider distribution with most angles between 45° and 100°.

We also applied the mesh smoothing algorithm described in Section 5.4 to the meshes in Table 6.1. The results can be seen in Table 6.2, and are quite striking. After vertex smoothing, no dihedral angle is smaller than 20°, and none are larger than 127°. This speaks to the fact that isosurface stuffing, particularly when using

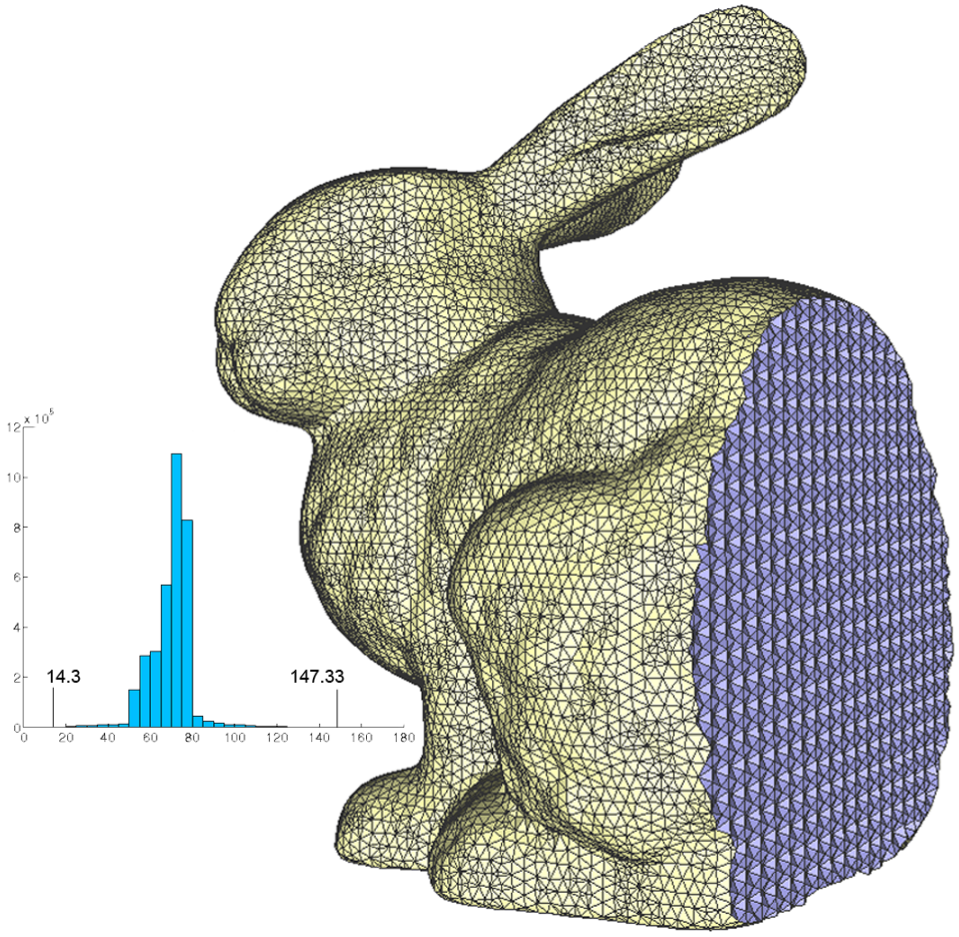


Figure 6.3: Bunny input, $h = 1.0$. with dihedral angle histogram

our acute lattice, produces a mesh of very high quality. Even if some poor dihedral angles exist near the mesh boundary, the structure of the mesh is such that they can usually be improved significantly by simple smoothing operations. This optimization step is not cheap, however, and can take several minutes to execute for more complex meshes (admittedly, our optimization code could likely be made

Table 6.2: Smooth inputs with optimization

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time (Total)
Sphere	0.2	30.18°	110.65°	3.19	0.1s
Sphere	0.1	28.20°	115.70°	3.43	0.86s
Bunny	3.0	25.64°	126.35°	4.05	8.15s
Bunny	1.0	22.51°	125.14°	4.21	414.49s
Dragon	0.02	20.13°	123.04°	3.91	12.56s
Dragon	0.01	20.70°	123.62°	3.85	195.58s

much faster, for example by using gradient descent or by restricting optimization to only take place near the boundary).

6.2 Feature Matching

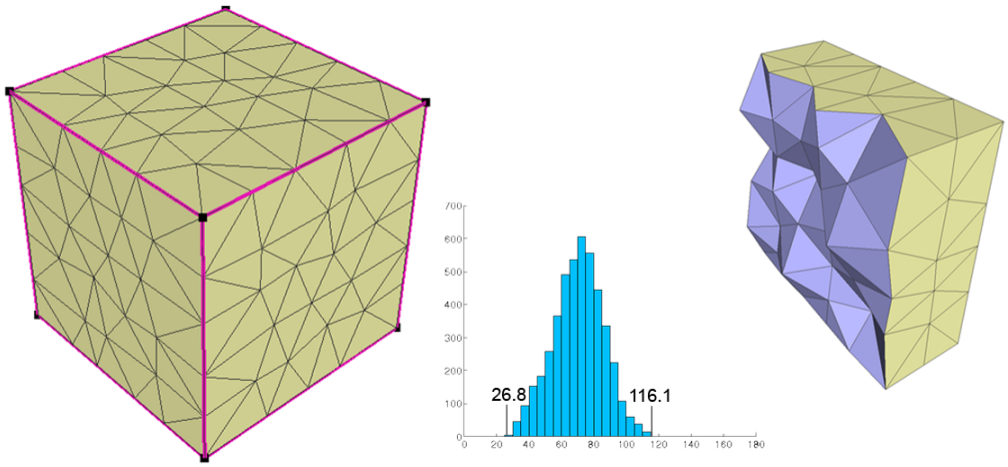


Figure 6.4: Cube, $h = 0.1$, after smoothing. Cutaway view at right.

Next we tested our feature matching algorithm from Chapter 5 on inputs with nonsmooth features. Uniform A15 isosurface stuffing was used as a starting point

Table 6.3: Nonsmooth inputs with feature matching, after vertex snapping

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time
Cube	0.25	26.57°	109.47°	4.47	0.001s
Cube	0.1	26.57°	109.47°	3.67	0.04s
Block	1.0	0°	180°	n/a	0.87s
Block	0.5	6.12°	164.65°	17.53	6.19s
Joint*	0.03	3.20°	168.27°	37.06	1.93s
Joint*	0.02	3.25°	171.42°	46.87	6.09s
Fandisk*	0.1	0.78°	175.38°	132.16	2.55s
Fandisk*	0.05	0.93°	178.26°	185.36	22.7s

*features not all resolved

Table 6.4: Nonsmooth inputs with feature matching, after smoothing

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time (Total)
Cube	0.25	31.08°	117.55°	3.66	0.01s
Cube	0.1	26.79°	116.11°	3.23	0.13s
Block	1.0	13.45°	146.74°	7.29	2.19s
Block	0.5	15.60°	127.08°	4.59	19.5s
Joint*	0.03	16.87°	132.47°	5.01	7.51s
Joint*	0.02	20.51°	126.22°	4.71	25.09s
Fandisk*	0.1	1.61°	173.95°	88.15	4.49s
Fandisk*	0.05	15.85°	145.03°	6.35	48.64s

*features not all resolved

for vertex snapping operations. The feature points and curves to match were provided by an automated process that analyzed the input surface mesh and computed curvature at vertices and face angles of each edge; vertices and edges whose curvature is found to be a particular threshold (tweaked by hand for each input) are marked as features to be matched.

Mesh quality results are given in Table 6.3 and Table 6.4. We have included

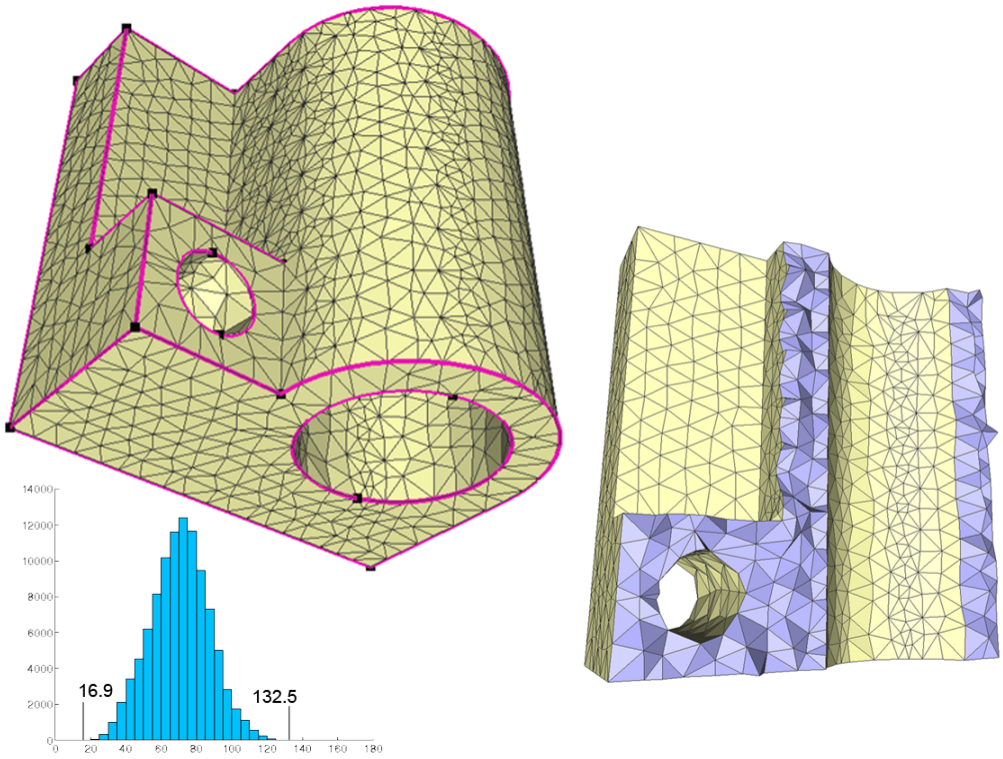


Figure 6.5: Joint, $h = 0.03$, after smoothing. Cutaway view at right.

quality statistics for the meshes before and after the smoothing step. This serves to show that, although snapping vertices onto features produces low quality elements, smoothing is usually sufficient to restore the lost quality.

The inputs we tested on are designed to have features that get progressively more difficult to resolve. The Cube (Figure 6.4) is the simplest input, and the Fandisk (Figure 6.7) is the most complicated, with many ridges and corners spaced relatively close together. The relative difficulty of the inputs is reflected in the results; tetrahedral quality declines as the input features become more and more complicated. The worst case by far is the Fandisk meshed with resolution $h = 0.1$,

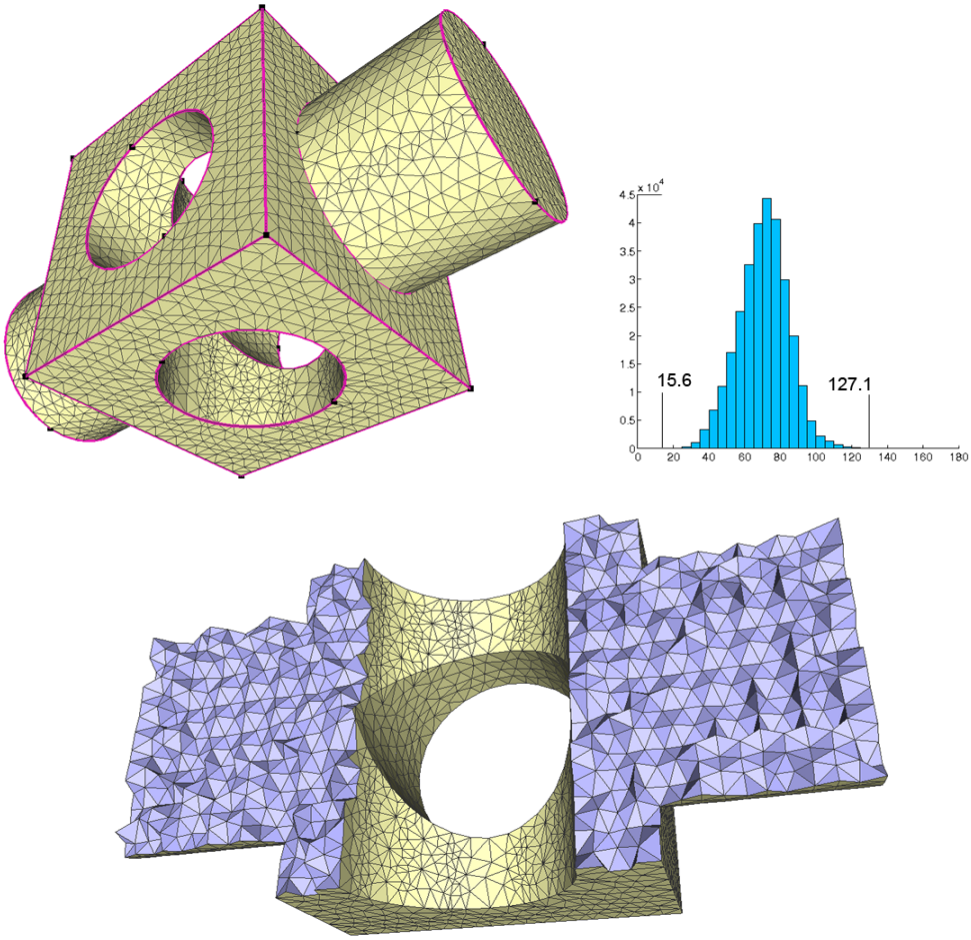


Figure 6.6: Block, $h = 0.05$, after smoothing. Cutaway views on bottom.

which yields dihedral angles less than 2° and greater than 173°. We note, however, that increasing the resolution to $h = 0.05$ gives a much better mesh, suggesting that the uncharacteristically poor result for the $h = 0.1$ mesh is probably due to under-resolving of the input surface.

We denote with asterisks those meshes that did not successfully resolve all

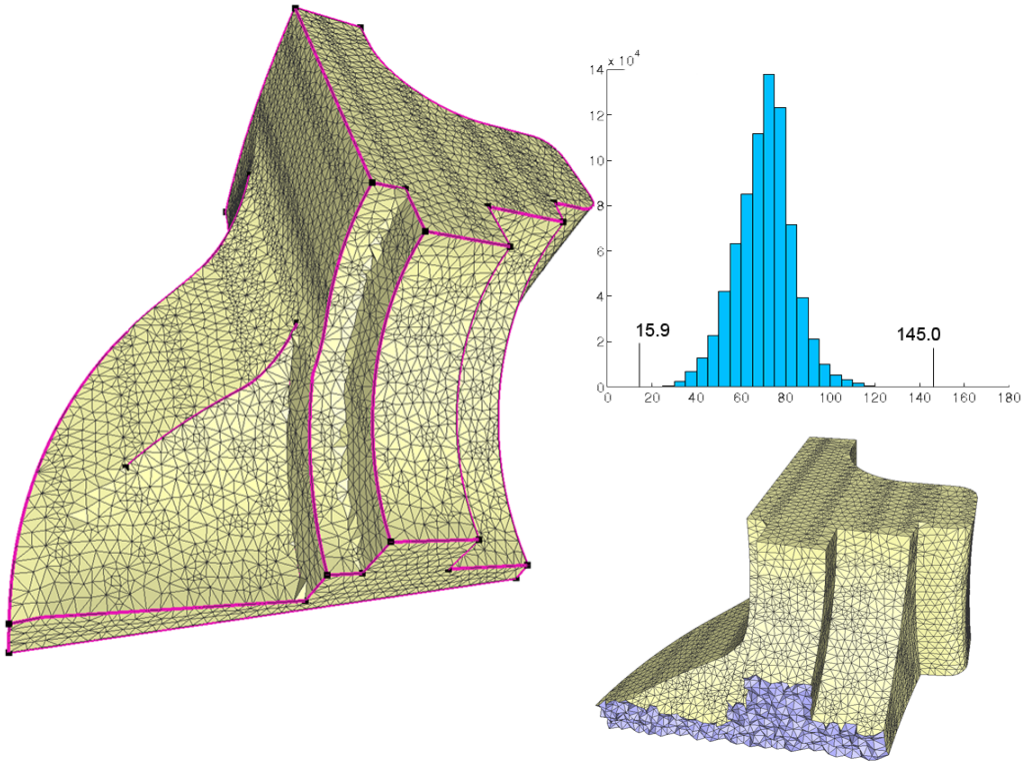


Figure 6.7: Fandisk, $h = 0.03$, after smoothing. Cutaway view at right.

features in the input. As described above, this occurred when no vertex could be snapped onto a feature without creating inverted or degenerate tetrahedra. In our tests, this happened with two or three features on the most difficult inputs. The algorithm still snaps as many vertices as possible, resulting in partially matched features.

Overall, our results show that our algorithm preserves sharp features much more successfully than isosurface stuffing on its own while still giving tetrahedra of reasonble quality. Unfortunately, we cannot guarantee that all features in the input will be resolved, because the algorithm fails gracefully in the cases where

mesh quality would be significantly compromised by our snapping operations. Although this makes our method unsuitable for some engineering applications, we believe this is acceptable in computer animation where it is desirable to trade total accuracy for improved efficiency; in this case the gains in speed and come from using higher-quality tetrahedra.

6.3 Adaptive A15 Lattice

We present here the quality statistics of adaptive lattices generated with the algorithms described in Chapter 4. In our experiments, we generated adaptive lattices based on a subset of the input surfaces used in Section 6.1 and Section 6.2. Because adaptivity permits higher-resolution tetrahedra at the input surface, the h sizes used are smaller than in the uniform cases.

The quality results presented are from the lattices themselves, before isosurface stuffing is performed. Because both algorithms guarantee that only regular A15 tiles intersect the input surfaces, the worst angles generated by warping and cutting operations will be the same as seen above. We give quality statistics of the lattices so as to distinguish the poor-quality tetrahedra that result from the adaptivity algorithm as opposed to the isosurface stuffing process.

Quality results for the A15 point lattice algorithm can be found in Table 6.5 and an example lattice can be seen in Figure 6.8. The results clearly show identical levels of quality for several different meshes. This occurs because of the tiled nature of the adaptive point lattice; particular configurations of tiles occur in each of the test cases, leading to identical Delaunay tetrahedralizations in those regions,

Table 6.5: Quality of lattices produced with the A15 Point Lattice algorithm

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time
Cube	0.05	2.55°	174.87°	47.24	0.51s
Sphere	0.1	0°	180°	n/a	0.47s
Sphere	0.05	0°	180°	n/a	2.57s
Bunny	2.0	2.55°	174.87°	47.24	12.98s
Block	0.5	2.55°	174.87°	47.24	7.22s
Block	0.25	2.55°	174.87°	47.24	36.71s
Joint	0.02	2.55°	174.87°	47.24	3.59s
Joint	0.01	0°	180°	n/a	20.51s
Fandisk	0.05	0°	180°	n/a	11.73s
Fandisk	0.025	0°	180°	n/a	67.74s

Table 6.6: Quality of lattices produced with the Red-Green A15 algorithm

Input	h	Min. Dihed.	Max Dihed.	Max Aspect	Time
Cube	0.05	10.924°	160.42°	12.19	1.82s
Sphere	0.1	9.97°	160.71°	12.35	1.89s
Sphere	0.05	6.38°	164.9°	18	26.3s
Bunny	2.0	0°	180°	n/a	204.33s
Block	0.5	1.03°	178.32°	162.28	120.51s
Block	0.25	0°	180°	n/a	1,882.08s
Joint	0.02	10.41°	161.34°	13.83	19.66s
Joint	0.01	4.01°	171.8°	29.07	388.59s
Fandisk	0.05	0°	180°	n/a	122.44
Fandisk	0.025	0°	180°	n/a	1,841.39s

and thus identical quality measures for the worst-quality tetrahedra in each mesh.

The results for the red-green A15 algorithm are in Table 6.6 and an example lattice is shown in Figure 6.9 (note that here the lattice is shown filling the entire bounding box of the domain, whereas in Figure 6.8 all tetrahedra outside the domain have been trimmed away). Due to the amount of optimization performed in

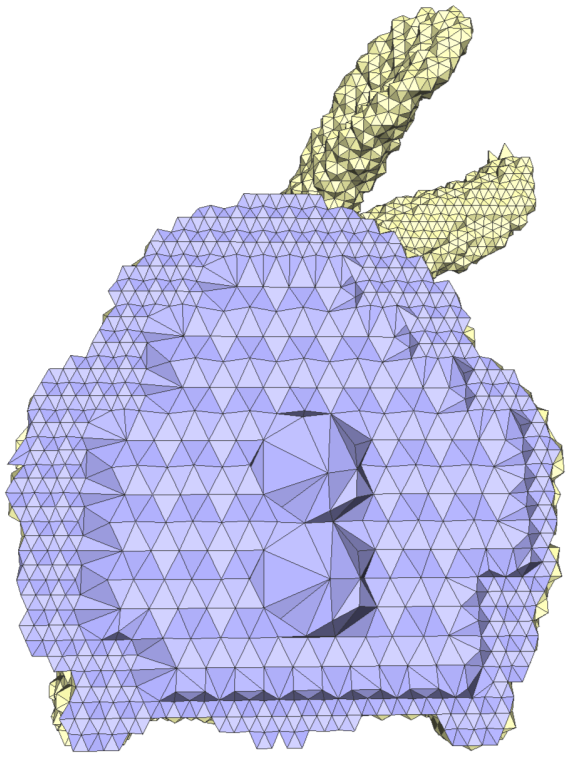


Figure 6.8: Adaptive lattice (Bunny) from A15 Point Lattice algorithm.

this algorithm, each generated mesh is unique in terms of quality, and no pattern emerges like in the point lattice algorithm. This also explains the much longer running times of the algorithm.

As has already been stated, our results show that these algorithms do not reliably produce high-quality tetrahedra. Furthermore, they do not improve on the adaptive BCC lattice algorithm of Labelle and Shewchuk [13], which has maximum dihedral angles of 120°.

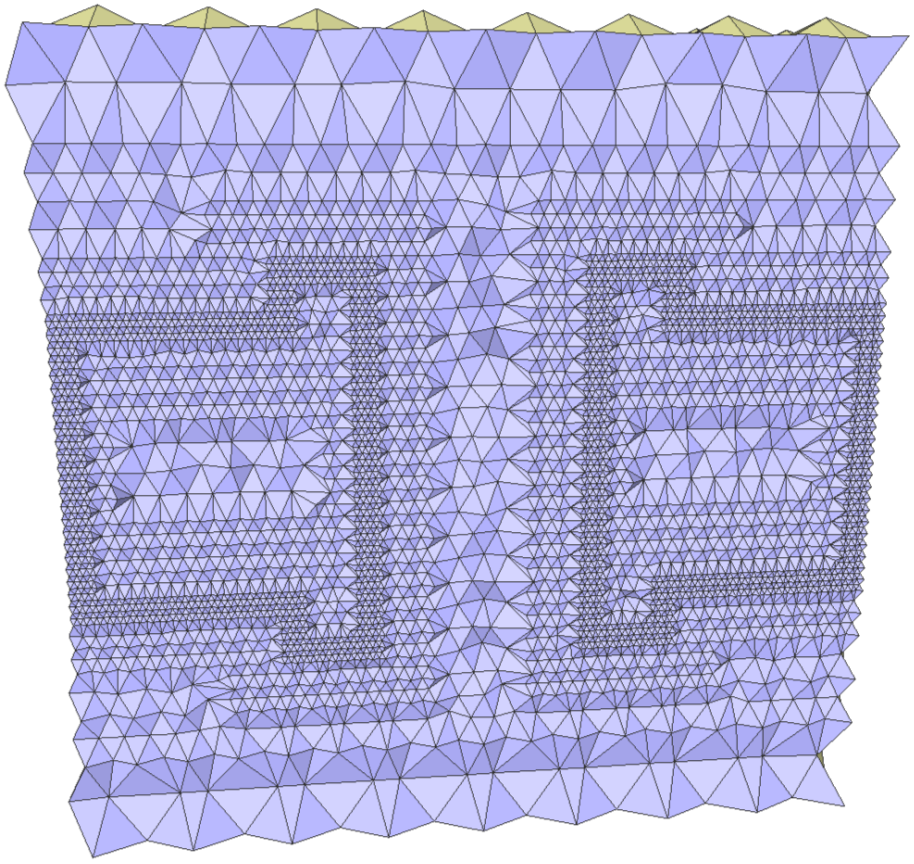


Figure 6.9: Adaptive lattice (Block) from Red-Green A15 algorithm.

Chapter 7

Conclusion

In summary, we have presented two improvements to the isosurface stuffing method of tetrahedral mesh generation. The first is the use of an acute tetrahedral lattice known as the A15 lattice, which produces better-quality tetrahedra, particularly on the interior of the mesh. The second improvement is a feature-matching algorithm that can restore most, if not all, of an input's sharp creases without sacrificing too much in terms of element quality.

Use of the A15 lattice in place of the BCC lattice is a relatively simple way of achieving large gains in overall quality in the uniform isosurface stuffing algorithm. Because the initial lattice used has higher-quality elements, the tetrahedra on the interior of the final mesh are necessarily of higher quality. Furthermore, the degradation of quality at the domain boundary caused by warping and cutting operations is mitigated by the higher-quality initial elements. We have established that A15 isosurface stuffing gives better-quality results in practice than standard

BCC isosurface stuffing. An important task in future will be to establish the worst-case quality bounds of A15 isosurface stuffing, which will establish whether the algorithm is provably better in all cases.

We have successfully applied the A15 lattice to the generation of uniform tetrahedral meshes; the natural next step is to use it to create high-quality adaptive meshes. We explored numerous techniques for doing so, including tetrahedral subdivision, Delaunay tetrahedralization, and octree techniques. Using these techniques, we designed two algorithms for generating adaptive meshes based on A15. One was based on finding the Delaunay tetrahedralization of an adaptive A15 point lattice; the other uses red-green subdivision and tetrahedral edge flipping operations to create conformal interfaces between A15 tiles of different sizes. Unfortunately, neither of our algorithms are able to consistently produce meshes with quality elements. There remains much potential for future work to discover high-quality adaptive tetrahedral meshing algorithms, either by pursuing additional combinations of the techniques mentioned above, or by novel research into a customized A15-based adaptive scheme. However, for the time being we conclude that the adaptive BCC lattice presented as part of the original isosurface stuffing algorithm is the best adaptive scheme currently available.

Finally, we have introduced an algorithm to restore sharp features of an input surface that are smoothed away by the isosurface stuffing process. First, feature points and endpoints of feature curves are resolved by snapping vertices of the mesh onto them. Next, paths through the boundary edges of the mesh are found between endpoints of each feature curve, with the paths chosen to follow

the curves as closely as possible. Finally, the feature curves are resolved by snapping as many of the vertices on the paths to the curves as possible. Vertices are not snapped if doing so would degrade quality below a given threshold, allowing for control over the trade-off between feature-matching and quality. Our algorithm does not guarantee all sharp features will be resolved in the output mesh. Nonetheless, it greatly expands the set of the inputs that can be effectively meshed using an isosurface stuffing approach to include many with sharp ridges and corners.

Future work in this area would focus on guaranteeing that all features are matched, while still preserving quality as much as possible. Operations that modify the mesh topology should also be explored as a means of improving on our feature-matching results, since our algorithm only modifies vertex positions. It seems likely that an approach making use of the entire initial lattice, not just the already-processed isosurface stuffing mesh, similar to the work done by Bronson et al. [5], could also yield significant improvements.

Bibliography

[1] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *ACM Transactions on Graphics*, 24(3):617–625, July 2005.

[2] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384 – 409, 1994.

[3] J. Bey. Tetrahedral grid refinement. *Computing*, 55(4):355–378, 1995.

[4] R. Bridson. *Fluid Simulation*. AK Peters, 2008.

[5] J. R. Bronson, J. A. Levine, and R. T. Whitaker. Lattice cleaving: Conforming tetrahedral meshes of multimaterial domains with bounded quality. In *Proceedings of the 21st International Meshing Roundtable*, pages 191–209. Springer, 2013.

[6] D. Burkhart, B. Hamann, and G. Umlauf. Adaptive and feature-preserving subdivision for high-quality tetrahedral meshes. In *Computer Graphics Forum*, volume 29, pages 117–127. Wiley Online Library, 2010.

[7] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *Journal of the ACM (JACM)*, 47(5):883–904, 2000.

[8] H. De Cougny and M. S. Shephard. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46(7):1101–1125, 1999.

[9] H. Edelsbrunner and D. Guoy. An experimental study of sliver exudation. *Engineering with computers*, 18(3):229–240, 2002.

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

[11] B. M. Klingner and J. R. Shewchuk. Agressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, Seattle, Washington, Oct. 2007.

[12] L. Kobbelt. 3-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 103–112. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] F. Labelle and J. R. Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics*, 26(3), July 2007.

[14] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in three dimensions. In *Proceedings of the eighth annual symposium on Computational geometry*, SCG '92, pages 212–221, New York, NY, USA, 1992. ACM.

[15] N. Molino, R. Bridson, and R. Fedkiw. Tetrahedral mesh generation for deformable bodies. In *Proc. Symposium on Computer Animation*, 2003.

[16] P. Möller and P. Hansbo. On advancing front mesh generation in three dimensions. *International Journal for Numerical Methods in Engineering*, 38(21):3551–3569, 1995.

[17] J. Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3): 305–363, 1997.

[18] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 32(4):709–749, 1991.

[19] J. Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures. *Eleventh International Meshing Roundtable*, pages 115–126, 2002.

[20] J. R. Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 86–95. ACM, 1998.

[21] H. Si. *TetGen, a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator*, January 2006.

[22] J. M. Sullivan. The geometry of bubbles and foams. In *Foams and emulsions*, pages 379–402. Springer, 1999.

[23] J. Teran, N. Molino, R. Fedkiw, and R. Bridson. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, 21(1):2–18, 2005.

[24] A. Üngör. Tiling 3d euclidean space with acute tetrahedra. In *Proc. 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 169–172, 2001.

[25] L. Velho, J. de Miranda Gomes, and D. Terzopoulos. Implicit manifolds, triangulations and dynamics. *Neural, Parallel & Scientific Computations*, 5 (1-2):103–120, 1997.

[26] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, and J. F. O'Brien. Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics (TOG)*, 29(4):49, 2010.

[27] B. Williams. Fluid surface reconstruction from particles. Master's thesis, The University Of British Columbia, 2008.

[28] M. Yerry and M. Shephard. A modified quadtree approach to finite element mesh generation. *Computer Graphics and Applications, IEEE*, 3(1):39–46, 1983.

[29] M. A. Yerry and M. S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, 20(11):1965–1990, 1984.

[30] H. Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, 74(250):603–627, 2005.

Appendix A

A15 Tile

We provide here a listing of the geometry of the modified A15 tile used in our implementations. It is a cleaned-up version of the tile described in Williams [27], which is in turn derived using the methodology of Üngör [24]. However, we position our tile such that the corners of the cube of side length 4 are all represented in the lattice; this makes it easier to overlay tiles on the cells of an octree.

Our tile contains 27 vertices:

0:	(0, 0, 4)	7:	(2, 0, 0)	14:	(3, 2, 3)	21:	(-1, 4, 2)
1:	(2, 0, 4)	8:	(4, 0, 0)	15:	(-1, 2, 1)	22:	(1, 5, 2)
2:	(4, 0, 4)	9:	(-1, 2, 5)	16:	(1, 2, 0)	23:	(3, 4, 2)
3:	(-1, 0, 2)	10:	(1, 2, 4)	17:	(3, 2, 1)	24:	(0, 4, 0)
4:	(1, 1, 2)	11:	(3, 2, 5)	18:	(0, 4, 4)	25:	(2, 4, 0)
5:	(3, 0, 2)	12:	(-1, 2, 3)	19:	(2, 4, 4)	26:	(4, 4, 0)
6:	(0, 0, 0)	13:	(1, 3, 2)	20:	(4, 4, 4)		

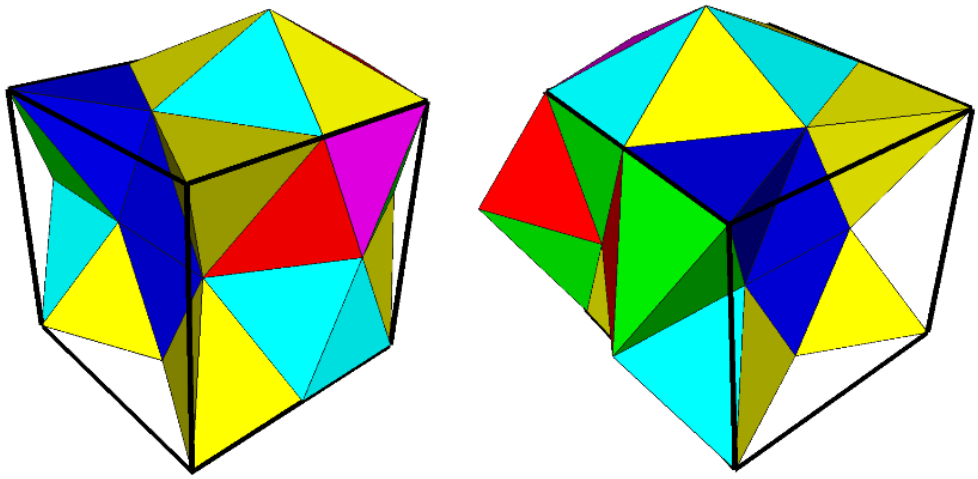


Figure A.1: A15 tile overlaid on cube (thick black lines)

The 46 tetrahedra that connect the vertices are:

0:	(12, 4, 0, 3)	12:	(6, 4,15, 3)	24:	(18,12,10, 9)	36:	(15,24,13,16)
1:	(0,12, 9,10)	13:	(15, 4,12, 3)	25:	(13,19,18,10)	37:	(24,15,13,21)
2:	(4,12, 0,10)	14:	(12,13, 4,15)	26:	(13,12,18,21)	38:	(24,25,22,13)
3:	(1, 4, 0,10)	15:	(16,15, 6, 4)	27:	(12,13,18,10)	39:	(24,22,21,13)
4:	(13,12, 4,10)	16:	(13,15,16, 4)	28:	(18,22,13,21)	40:	(25,24,15,13)
5:	(14, 1, 2,11)	17:	(4,17, 7,16)	29:	(19,13,14,10)	41:	(25,22,13,23)
6:	(1,14, 4,10)	18:	(17, 4, 7, 5)	30:	(19,11,10,14)	42:	(13,17,16,25)
7:	(14, 1, 4,10)	19:	(17,13,16, 4)	31:	(11,19,20,14)	43:	(25,13,17,23)
8:	(14,13, 4,10)	20:	(4,14, 5,17)	32:	(14,19,20,23)	44:	(13,14,17,23)
9:	(1,14, 2, 5)	21:	(8, 5,17, 7)	33:	(19,22,23,13)	45:	(23,26,17,25)
10:	(1,11,14,10)	22:	(13,17,14, 4)	34:	(13,19,14,23)		
11:	(6, 7,16, 4)	23:	(22,19,18,13)	35:	(13,12,21,15)		

When overlaying tiles on an octree, it's important to know which vertices are shared by adjacent tiles. We found the easiest way to accomplish this was to establish a mapping between the A15 tile and a cubical, grid-like mesh with the same topology. This in turn allows a mapping from faces, edges, and corners of

an octree cell to the vertices of the A15 tile. The grid analogue is achieved by repositioning the vertices of the A15 tile as follows:

0:	(0, 0, 4)	7:	(2, 0, 0)	14:	(4, 2, 2)	21:	(0, 4, 2)
1:	(2, 0, 4)	8:	(4, 0, 0)	15:	(0, 2, 0)	22:	(2, 4, 2)
2:	(4, 0, 4)	9:	(0, 2, 4)	16:	(2, 2, 0)	23:	(4, 4, 2)
3:	(0, 0, 2)	10:	(2, 2, 4)	17:	(4, 2, 0)	24:	(0, 4, 0)
4:	(2, 0, 2)	11:	(4, 2, 4)	18:	(0, 4, 4)	25:	(2, 4, 0)
5:	(4, 0, 2)	12:	(0, 2, 2)	19:	(2, 4, 4)	26:	(4, 4, 0)
6:	(0, 0, 0)	13:	(2, 2, 2)	20:	(4, 4, 4)		

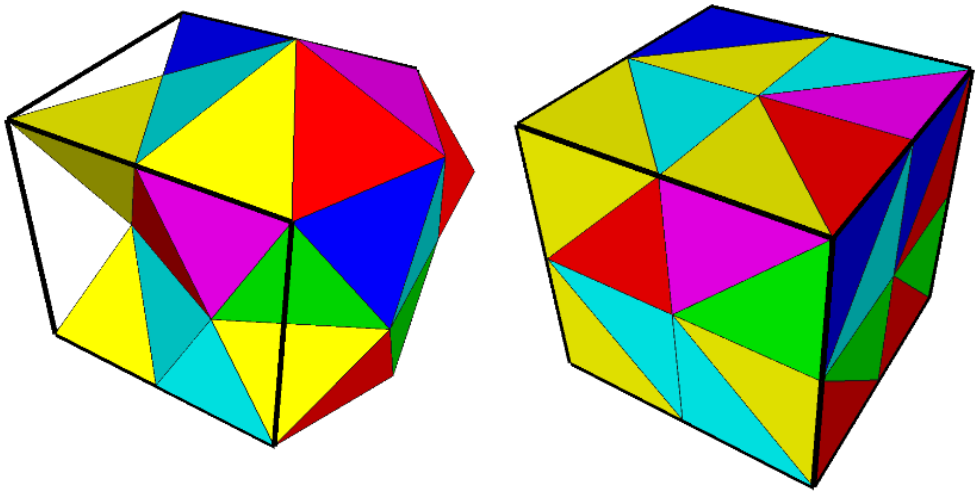


Figure A.2: Mapping of A15 tile elements to cube.