Four-Dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations

Philip Claude Caplan

S.M. Massachusetts Institute of Technology (2014)
B.Eng. McGill University (2012)

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Aeronautics and Astronautics at the Massachusetts Institute of Technology

June 2019

Author .....
Department of Aeronautics and Astronautics
May 23rd, 2019

Certified by .....
David L. Darmofal
Professor of Aeronautics and Astronautics, MIT
Thesis Committee Chair

Certified by .....
Robert Haimes
Principal Research Engineer, MIT
Thesis Committee Member

Certified by .....
Jaume Peraire
H.N. Slater Professor of Aeronautics and Astronautics, MIT
Thesis Committee Member

Accepted by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics, MIT
Chairman, Graduate Program Committee

Four-Dimensional Anisotropic Mesh Adaptation for Spacetime Numerical Simulations

Philip Claude Caplan

Submitted to the Department of Aeronautics and Astronautics
on May 23rd, 2019, in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Engineers and scientists are increasingly relying on high-fidelity numerical simulations. Within these simulations, mesh adaptation is useful for obtaining accurate predictions of an output of interest subject to a computational cost constraint. In the quest for accurately predicting outputs in problems with time-dependent solution features, a fully unstructured coupled spacetime approach has been shown to be useful in reducing the cost of the overall simulation. However, for the simulation of unsteady three-dimensional partial differential equations (PDEs), a four-dimensional mesh adaptation tool is needed.

This work develops the first anisotropic metric-conforming four-dimensional mesh adaptation tool for performing adaptive numerical simulations of unsteady PDEs in three dimensions. The theory and implementation details behind our algorithm are first developed alongside an algorithm for constructing four-dimensional geometry representations. We then demonstrate our algorithm on three-dimensional benchmark cases and it appears to outperform existing implementations, both in metric-conformity and expected tetrahedra counts. We study the utility of the mesh adaptation components to justify the design of our algorithm. We then develop four-dimensional benchmark cases and demonstrate that metric-conformity and expected pentatope counts are also achieved. This is the first time anisotropic four-dimensional meshes have been presented in the literature.

Next, the entire mesh adaptation framework, Mesh Optimization via Error Sampling and Synthesis (MOESS), is extended to the context of finding the optimal mesh to represent a function of four variables. The mesh size and aspect ratio distributions of the optimized meshes match the analytic ones, thus verifying our framework. Finally, we apply MOESS in conjunction with the mesh adaptation tool to perform the first four-dimensional anisotropic mesh adaptation for the solution of the advection-diffusion equation. The optimized meshes effectively refine the solution features corresponding to both a boundary layer solution as well as an expanding spherical wave.

Thesis Supervisor: David L. Darmofal
Title: Professor of Aeronautics and Astronautics, MIT

Thesis Supervisor: Robert Haimes
Title: Principal Research Engineer, MIT

Thesis Supervisor: Jaume Peraire
Title: H.N. Slater Professor of Aeronautics and Astronautics, MIT

Acknowledgements

No way could this work
Have seen the light of day,
Were it not for the many
Who helped along the way.

My first week in the lab
in September 2012,
"From where do you join us?"[1]
So welcomed I felt.

A grasshopper I was,
In front of a class.
But I've had a great coach[2],
And now I'm up to the task.

I learned the importance,
Though some still don't see -
Of – a thing-ama-bob?
No! Of the *geometry*![3]

Now is it sans or is it mit?[4]
Or is it with – or without?
I'm so confused, please help!
Ah a unit test – there's no doubt.

It's a good thing I write code,
Because some[5] are aware
Of my nightmarish practices
With the machines downstairs.

Without eyes[6] on this text
You'd wonder: how can it be?
He spoke English for thirty years?
He'd be last in a spelling bee!

To those[7] who were patient
With the customs I imposed.
Make coffee! Mind the lights!
In the recycling, that goes!

[1] Jaume Peraire

[2] David Darmofal

[3] Bob

[4] Arthur, Ben, Carmen, Cory, Hugh, Marshall, Savithru, Shun, Steve Allmaras

[5] Max, Sarthak, Rich, Dave Robertson, Todd Billings, Unified class of 2017

[6] Mike Park

[7] Anyone sitting in 37-312

Some of you kept me
Healthy and sane.
With a coffee or a beer[8]
Or the lifestyle in Spain[9].

I'd be lost without those[10]
Who never let me forget:
A deadline! Your reimbursement!
Did you get your insulin[11] yet?

Far off in the woods,
Is a home away from home,
Where the birds are well-fed,
A loving bunch[12] makes me welcome.

But my true home always is
With the denmates[13] in my heart,
And ever-supportive bears[14],
This way, we're never apart.

Now, never would I deliver
This work - even a sliver -
Without a giver so sillier,
My heart was soon tillered,
And filler'd with a river
Of love so much bigger,
I owe many thanks
To my purple-haired pillar[15].

Last, but not least,
I must credit my funding source.
Though Canadian I am,
I thank the US Air Force[16].

[8] Fan, Irina, Nisha, Patrick, Victor

[9] Abel, Eloi, Ferran, Guillem, Julia, Xevi

[10] Beth, Jean, Robin, Pam

[11] Jessica

[12] Caroline, Dot, Ellie, Mike, Ollie

[13] Ryan, Alex, Ursa

[14] Maman, Dad,
Grandmaman, Grandpapa, Nani

[15] Catherine Miller

[16] This work was funded by the CAPS project: AFRL Contract FA8050-14-C-2472: *CAPS: Computational Aircraft Prototype Syntheses* with Dean Bryson as technical monitor.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

It's not about aptitude.
It's about attitude.
— *Masayuki Yano*

1.1 Motivation

Hi, I'm a Voronoi diagram and I'll be here to clarify concepts along the way.

The increase in computational power in the last several decades has given rise to efficient algorithms for predicting engineering quantities of interest with numerical simulations. In particular, high-order finite element methods combined with mesh adaptation techniques for numerically solving partial differential equations have demonstrated their potential for *accurately* predicting these quantities. Yano and Darmofal[1] remark upon the importance of mesh adaptation when using high-order discretizations for estimating the drag and lift with the Reynolds-Averaged Navier-Stokes equations. This finding is further supported by the CFD Vision 2030 study[2] in which the authors emphasize the development of (1) mesh adaptation and (2) high-order discretizations to achieve an autonomous and reliable CFD simulation.

A typical adaptive numerical simulation is shown in Figure 1.1. The simulation begins with a description of the domain geometry, a system of partial differential equations (PDEs) along with appropriate boundary conditions, the engineering output of interest and a measure of the computational cost available to expend on the simulation. Upon entry into the simulation loop, the domain is decomposed into a *mesh*, a discrete representation of the domain through a collection of non-overlapping *elements*. The partial differential equation (PDE) is solved on this initial mesh and error estimation techniques are used to assess

1. Yano *et al.*, *The Importance of Mesh Adaptation for Higher-Order Discretizations of Aerodynamic Flows*. 2011

2. Slotnick *et al.*, *CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences*. 2014

the numerical error in the output resulting from the discretization of the PDE. If the error is within tolerance, the adaptive simulation is finished. However, when the error is too large and the computational cost is still below the limit, the mesh is *adapted* to reduce this error and the loop returns to the solver block.

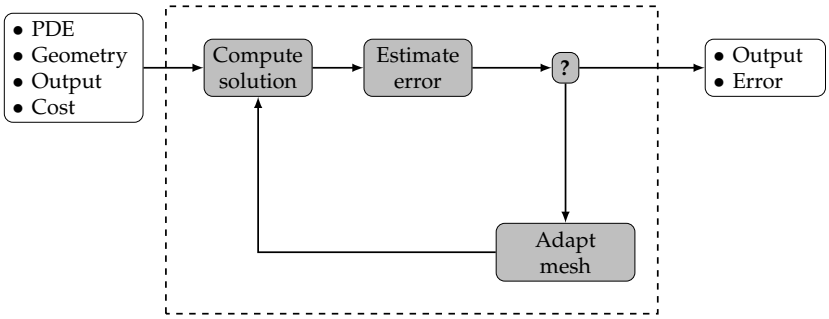


Figure 1.1: Schematic of an adaptive numerical simulation.

For the calculation of time-varying outputs, Yano demonstrated that a significant computational savings can be achieved by adapting in a spatiotemporal domain[3]. For example, consider a one-dimensional problem with a propagating feature of characteristic size δ (moving from left to right in Figure 1.2(a)). A purely spatial mesh would require elements of width $\Delta x < \delta$ near the feature to accurately resolve its position. Now, to accurately track the position in time, a traditional time-marching approach (Figure 1.2(a)) would require a small time step (Δt) to accurately resolve the feature. A fully unstructured, coupled spacetime approach (Figure 1.2(b)) would accurately resolve the feature if the mesh elements are stretched to directly align with the anisotropic feature in a now two-dimensional domain. The latter has the advantage of dramatically reducing the number of elements required to achieve a similar level of error with a time-marching approach. Previous efforts have successfully applied this technique to capture convecting vortices[3] and for spacetime oil reservoir simulations[4,5] in $2d + t$.

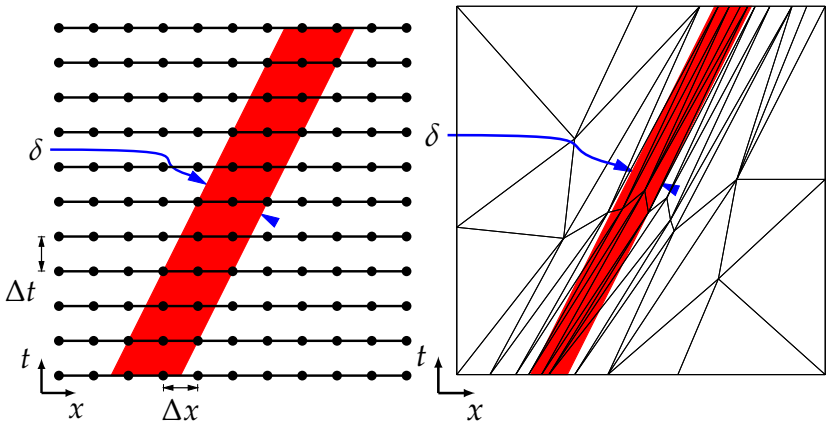
The ultimate goal of this work is to develop a meshing tool to be used in conjunction with an adaptive numerical simulator for the solution of PDEs in $3d + t$ to enable the accurate solution of unsteady problems. To do so, we will need a four-dimensional meshing capability.

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

4. Jayasinghe, *An Adaptive Space-Time Discontinuous Galerkin Method for Reservoir Flows*. 2018

5. Jayasinghe *et al.*, *A Space-Time Adaptive Method for Reservoir Flows: Formulation and One-Dimensional Application*. 2018



(a) Time-marching approach. **(b)** Unstructured approach.

Figure 1.2: Time-marching versus unstructured spacetime approaches for the solution of unsteady problems.

1.2 Background

Mesh adaptation for the numerical solution of unsteady partial differential equations

Unsteady partial differential equations can be solved by discretizing the spatial domain with, say, a finite element method, thus constructing an ordinary differential equation that can be discretized in time. This fixed (in time) spatial mesh can be adapted, however, the computational cost remains high because this single mesh would need to capture all spatial and temporal solution features.

Alternatively, the finite element method can be used to discretize the entire spatiotemporal domain – see the pioneering work of Oden[6], Argyris and Scharpf[7] and Fried[8]. This coupling method enables the use of a wider range of adaptation techniques, ranging from timeslab to fully unstructured approaches.

First, a timeslab approach constructs spacetime elements as the tensor-product of a (possibly unstructured) spatial element with a time interval. These are simpler than fully unstructured spatiotemporal approaches because they essentially require spatially-adapted meshes. The coupled spatiotemporal mesh can be constructed in several ways. Behr[9] extrudes an initial spatial mesh to form prismatic elements which are then subdivided to form the required simplices. Temporal refinement is achieved by inserting vertices along edges formed during this extrusion process. The Tent Pitcher algorithm of Ungor[10] is essentially an advancing front method, starting from an initial spatial mesh and inserts points in the temporal direction to satisfy a *cone constraint* imposed by the characteristics of the governing PDE. This method was extended to the $3d + t$ case by Mont[11]. Thite[12] improved the Tent

6. Oden, *A General Theory of Finite Elements II. Applications*. 1969

7. Argyris *et al.*, *Finite Elements in Time and Space*. 1969

8. Fried, *Finite-Element Analysis of Time-Dependent Phenomena*. 1969

9. Behr, *Simplex Space-Time Meshes in Finite Element Simulations*. 2008

10. Üngör *et al.*, *Tent-Pitcher: A Meshing Algorithm for Space-Time Discontinuous Galerkin Methods*. 2000

11. Mont, *Adaptive Unstructured Spacetime Meshing for Four-Dimensional Spacetime Discontinuous Galerkin Finite Element Methods*. 2011

12. Thite, *Adaptive Spacetime Meshing for Discontinuous Galerkin Methods*. 2007

Pitcher algorithm to also allow for coarsening of $2d + t$ spatiotemporal meshes.

Fidkowski also employs a tensor-product approach for the solution of the compressible Navier-Stokes equations[13,14] in which the spatial mesh is fixed and the temporal discretization is refined by bisecting the time intervals. In other words, each spatial element at a particular time takes the same time step.

Bangerth and Rannacher[15] employ a hierarchical refinement approach to subdivide quadrilateral spatiotemporal elements in a tensor-product manner (Figure 1.3(b)). Their adaptive method is driven by the dual-weighted residual error estimator. They demonstrate the approach on linear second-order hyperbolic problems in $1d + t$ and later to $2d + t$[16]. They conclude that their refinement approach is superior to uniform refinement approaches (Figure 1.3(a)) in achieving a lower output error at a lower DOF count. The advantage of this tensor-product approach was further demonstrated for Burger's equation by Hartmann[17].

Yano demonstrates that these tensor-product approaches are effectively isotropic and, whereas they offer a substantial DOF savings over uniform refinement, they are dramatically outperformed by a fully unstructured anisotropic approach. In the context of the one-dimensional problem of Figure 1.2 with a propagating feature of characteristic size δ, Yano experimentally observes that uniform (Figure 1.3(a)), tensor-product (Figure 1.3(b)) and anisotropic refinement approaches (Figure 1.2(b)) respectively require $\mathcal{O}(\delta^{-2})$, $\mathcal{O}(\delta^{-1})$ and $\mathcal{O}(1)$ DOF.

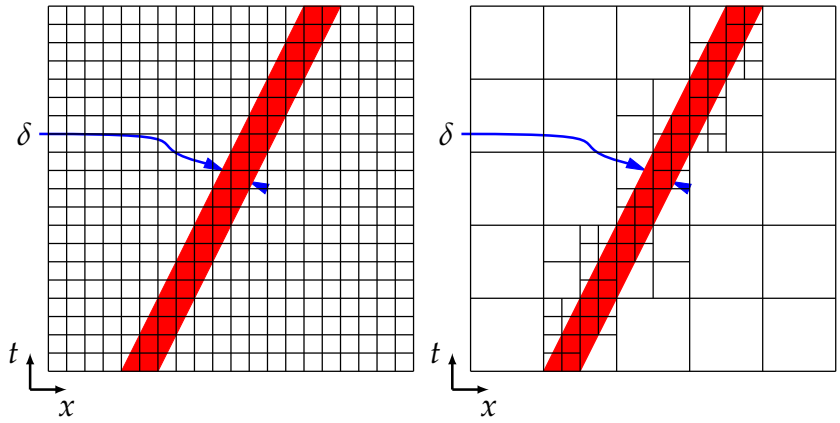
13. Fidkowski *et al.*, *Output-Based Space-Time Mesh Adaptation for the Compressible Navier-Stokes Equations*. 2011

14. Fidkowski, *Output-Based Space-Time Mesh Optimization for Unsteady Flows Using Continuous-in-Time Adjoints*. 2017

15. Bangerth *et al.*, *Finite Element Approximation of the Acoustic Wave Equation: Error Control and Mesh Adaptation*. 1999

16. Bangerth *et al.*, *Adaptive Galerkin Finite Element Methods for the Wave Equation*. 2010

17. Hartmann, *Adaptive FE Methods for Conservation Equations*. 2001



(a) Uniform. **(b)** Tensor-product.

Figure 1.3: Uniform and tensor-product refinement approaches to capture a spatiotemporal solution feature (in red).

Yano further demonstrates his approach for nonlinear $2d + t$ problems which is later extended to oil reservoir simulations by Jayasinghe[4]. In fact, Jayasinghe[5] is the first to demonstrate that a spatiotemporal approach also scales very well with the number of parallel pro-

4. Jayasinghe, *An Adaptive Space-Time Discontinuous Galerkin Method for Reservoir Flows*. 2018

5. Jayasinghe *et al.*, *A Space-Time Adaptive Method for Reservoir Flows: Formulation and One-Dimensional Application*. 2018

cessors when compared to time-marching approaches. The use of a fully unstructured spatiotemporal approach to solve $3d + t$ problems is well motivated but has yet to be demonstrated.

Mesh generation versus adaptation

At this point, it is worth highlighting the distinction we make between mesh *generation* and mesh *adaptation*.

- **Mesh generation:** consists of the complete construction of the topology of a mesh. An input set of points may be provided or may be determined during the mesh generation procedure. For example, the Delaunay triangulation constructs the set of triangles satisfying the Delaunay *property* from a fixed input set of points. Mesh generation might, but is not required to, start with an initial mesh.
- **Mesh adaptation:** consists of the modification of an initial mesh to match some desired properties. Both the topology and vertices (points) of the mesh may be modified by the adaptation procedure. An input mesh must be provided.

Whenever we employ the term *meshing*, it should be understood as *either* mesh generation or adaptation in an application where either approach might be suitable. That is, *meshing* will refer to the construction of a new mesh, whether it be entirely new (generation) or as a modification of an initial mesh (adaptation).

In the next sections, we describe common methods for mesh generation and adaptation, primarily for two- and three-dimensional domains. Some algorithms are inherently dimension-independent, though lack practical demonstrations in higher-dimensions. The goal is to select a method which we can use to generate anisotropic four-dimensional meshes.

Anisotropic meshing by advancing fronts

The advancing front method is successful at generating meshes about complex geometries by beginning with a discretization of the domain boundaries. It then proceeds by inserting points until the domain volume is filled with elements[18–20]. Fronts may be advanced into a void domain in which expensive intersection calculations need to be employed to detect front collisions, or can be advanced into an existing mesh[21] so as to robustly determine when front collisions occur.

The method readily extends to the anisotropic setting, in which the placement of each point is guided by requested sizing and stretching fields[22,23]. Alternatively, layers can be advanced[24,25] to achieve the orthogonal high aspect-ratio elements often desired to capture boundary layers when solving the Navier-Stokes equations.

18. Peraire *et al.*, *Adaptive Remeshing for Compressible Flow Computations.* 1987

19. Löhner *et al.*, *Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method.* 1988

20. Peraire *et al.*, *Finite Element Euler Computations in Three Dimensions.* 1988

21. Marcum *et al.*, *Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection.* 1995

22. Löhner, *Adaptive Remeshing for Transient Problems.* 1989

23. Peraire *et al.*, *Adaptive Remeshing for Three-Dimensional Compressible Flow Computations.* 1992

24. Pirzadeh, *Three-Dimensional Unstructured Viscous Grids by the Advancing-Layers Method.* 1996

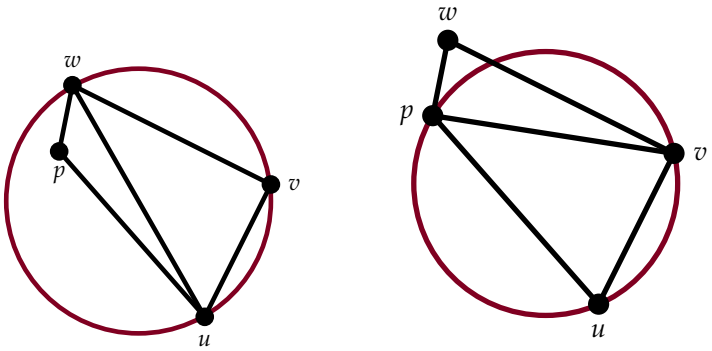
25. Alauzet *et al.*, *A Closed Advancing-Layer Method with Connectivity Optimization-based Mesh Movement for Viscous Mesh Generation.* 2015

Anisotropic Delaunay mesh generation

The Delaunay kernel is based on the simple premise: *no vertex shall be contained within the circumscribing ball of a simplex*. To satisfy this property, various algorithms have been proposed to construct Delaunay triangulations (triangulations, here, referring to a mesh of any dimensional simplices)[26] and implemented in softwares such as `Qhull`[27]. Edge swaps are commonly used to satisfy the Delaunay property, as shown in Figure 1.4.

26. Watson, *Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes*. 1981

27. Barber *et al.*, *Qhull: Quickhull Algorithm for Computing the Convex Hull*. 2013



(a) Before swap **(b)** After swap

Figure 1.4: Satisfying the Delaunay property with an edge swap. Before the swap is applied, the vertex p lies within the circumcircle of triangle uvw. Circumcircles are outlined in red.

The extension of the Delaunay kernel to the anisotropic setting has been achieved in $2d$[28–30] and $3d$[31]. However, Boissonnat recently proved that for any Riemannian manifold of dimension greater than two, the anisotropic Delaunay triangulation of this manifold may not yield a valid triangulation. In fact, he provides a minimum point sampling density to guarantee the construction of a valid mesh. Either the manifold needs to be sufficiently sampled, possibly using the algorithm of Rouxel-Labbé[32], or the simplices need to be curved[33]. Increasing the sampling density compromises the ability to match the desired element sizes governed by the Riemannian metric. Furthermore, it is unclear how to curve the mesh to produce valid simplices.

28. Borouchaki *et al.*, *Mailleur Bidimensionnel de Delaunay Gouverné par une Carte de Métriques. Partie I: Algorithmes*. 1995

29. Hecht, *BAMG: Bidimensional Anisotropic Mesh Generator*. 1998

30. Bossen *et al.*, *A Pliant Method for Anisotropic Mesh Generation*. 1996

31. Dobrzynski *et al.*, *Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations*. 2008

32. Rouxel-Labbé *et al.*, *Discretized Riemannian Delaunay Triangulations*. 2016

33. Boissonnat *et al.*, *An Obstruction to Delaunay Triangulations in Riemannian Manifolds*. 2018

Anisotropic mesh generation via isometric embeddings

The concept of anisotropic mesh generation via isometric embeddings was first proposed by Cañas et al.[34] and revived by Lévy and Bonneel[35]. This method is motivated by the difficulties when directly extending isotropic mesh generation algorithm to the anisotropic setting. The technique consists of three main stages:

1. Stretch an input mesh and metric field to a higher-dimensional Euclidean space such that the lengths of the edges of the mesh in the Euclidean space match the lengths prescribed by the metric field. In

34. Cañas *et al.*, *Surface Remeshing in Arbitrary Codimensions*. 2006

35. Lévy *et al.*, *Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration*. 2012

other words, construct an *isometric* embedding of the mesh-metric pair.

2. Generate a uniform mesh of the domain defined by the stretched mesh. This step regenerates the set of vertices and determines the topology of the generated simplices.

3. Map the vertices of the new stretched mesh back to the original domain.

As a result, the anisotropic mesh generation problem has been recast as a uniform one in the embedding (stretched) space. An illustration of this approach is shown in Figure 1.5. Observe that the isotropic surface mesh in $3d$ becomes anisotropic when mapped back to $2d$.

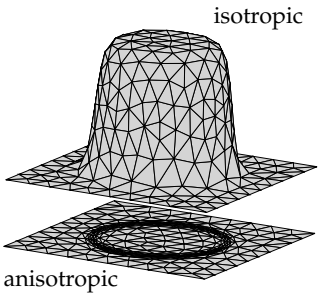


Figure 1.5: Illustration of the embedding technique for anisotropic mesh generation.

A few questions arise: how do we construct this isometric embedding and what is the minimum dimension of the higher-dimensional Euclidean space needed to achieve it? The latter question was addressed in the continuous case by Nash[36,37] who proposed embedding dimensions for the cases of piecewise-constant and smooth metric fields. At the onset of this thesis work, the former question had not yet been addressed for general metric fields. In fact, the works of Cañas and Lévy involved an explicit specification of the embedding coordinates by using the normal vectors of an input surface; this was further used in the work of Nivoliers[38]. The first attempt to generalize the embedding framework for anisotropic mesh generation for PDE-driven adaptation was the work of Dassi[39–42] who used the gradient of the solution to a PDE to construct a five-dimensional embedding. He further generalized the method to $3d$ but the method was driven purely by the solution to the PDE and not by a metric field.

Aside from the explicit formulations of their embeddings, Lévy and Dassi's approaches also differ in their remeshing methods. Lévy relies on the dual of the restricted Voronoi diagram (computed in the higher-dimensional space) to extract the final triangulation. This method is attractive because the topology of the mesh is simply extracted from

36. Nash, *C^1 Isometric Imbeddings*. 1954

37. Nash, *The Imbedding Problem for Riemannian Manifolds*. 1956

38. Nivoliers *et al.*, *Anisotropic and Feature Sensitive Triangular Remeshing Using Normal Lifting*. 2015

39. Dassi *et al.*, *A Priori Anisotropic Mesh Adaptation Driven by a Higher Dimensional Embedding*. 2017

40. Dassi *et al.*, *Curvature-Adapted Remeshing of CAD Surfaces*. 2014

41. Dassi *et al.*, *Anisotropic Finite Element Mesh Adaptation via Higher Dimensional Embedding*. 2015

42. Dassi *et al.*, *An Anisoptropic Surface Remeshing Strategy Combining Higher Dimensional Embedding with Radial Basis Functions*. 2016

the restricted Voronoi diagram, meaning the isotropic mesh generation problem is reduced to a vertex smoothing problem along the original embedded mesh. Lévy provides clipping algorithms to compute restricted Voronoi polygons and polytopes[43] but it is unclear how four-dimensional polytopes should be computed. Furthermore, the extracted dual restricted Delaunay triangulation will have the same geometry-conformity issues that have plagued the Delaunay method. These issues have been addressed in two and three dimensions when the dimension of the ambient Euclidean space is the same as that of the dimension of the simplices. Notable software implementations to recover a *constrained* Delaunay triangulation for two and three dimensions are `Triangle`[44] and `TetGen`[45].

43. Lévy, *"Geogram: A Programming Library of Geometric Algorithms"*. 2016

Dassi, instead, uses local mesh operators (such as splits, collapses and swaps) in the embedding space but performs validity checks in the original space. The method performs well when compared to existing anisotropic mesh generators[29]. However, the advantage of embedding and remeshing using local mesh operators is unclear in contrast to simply using local mesh operators in the original domain.

44. Shewchuk, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*. 1996

45. Si, *TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator*. 2015

29. Hecht, *BAMG: Bidimensional Anisotropic Mesh Generator*. 1998

As mentioned earlier, the problem of computing an isometric embedding with general metric fields had not been studied at the onset of the current work (circa June 2015). In fact, our original work in 2017 introduced the first algorithm for computing a nearly isometric embedding of meshes equipped with arbitrary Riemannian metric fields in a dimension-independent manner. To compute the embedding, the lengths of the geodesics between all vertices in the mesh were stored in a dense Euclidean distance matrix. The eigendecomposition of this matrix provided the higher-dimensional embedding coordinates, similar to the way multidimensional scaling has been used to infer a lower-dimensional representation of high-dimensional data[46]. A novel concept in our work was to *lift* the mesh coordinates so as to ensure the embedded mesh does not self-intersect. After investigating issues with geometry conformity when employing the restricted Voronoi diagram in the embedding space, we ultimately realized the extracted mesh may not be valid due to a loss of the one-to-one property when creating a new mesh in the higher-dimensional space. In fact, it might still be possible to extract a valid mesh if the extracted simplices are *curved*. Our practical finding is aligned with the theoretical finding of Boissonnat, who suggested simplices should be curved when employing the anisotropic Delaunay kernel[33]. Our work on this topic was published for the embeddings of straight-sided meshes[47] and curvilinear meshes[48].

46. Tenenbaum *et al.*, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*. 2000

33. Boissonnat *et al.*, *An Obstruction to Delaunay Triangulations in Riemannian Manifolds*. 2018

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

48. Caplan *et al.*, *Isometric Embedding of Curvilinear Meshes Defined on Riemannian Metric Spaces*. 2018

More recently, Zhong et al.[49] presented a method similar to our technique. They also lift the mesh to the higher-dimensional Euclidean space but use an optimization-based approach to compute the codi-

49. Zhong *et al.*, *Computing a High-dimensional Euclidean Embedding from an Arbitrary Smooth Riemannian Metric*. 2018

mension coordinates. The authors suggest their algorithm works in the presence of anisotropy and that the provably terminating algorithm of Rouxel-Labbé[32] is implemented to increase the sampling density when inverted elements are detected. However, it should be noted that they study three-dimensional problems with anisotropy ratios of only 25:1. In fact, they mention the point insertion algorithm is never triggered in their problems. For truly anisotropic problems, either this algorithm would be triggered or the extracted restricted Delaunay triangulation would be invalid.

32. Rouxel-Labbé *et al.*, *Discretized Riemannian Delaunay Triangulations*. 2016

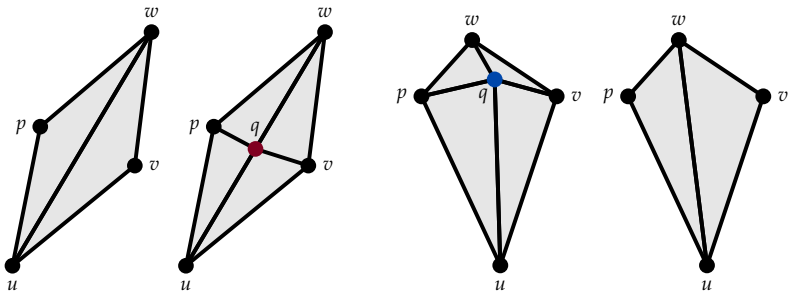
Anisotropic mesh adaptation via local mesh operators

Starting with an initial, geometry-conforming mesh, local mesh operators iteratively modify this mesh until a particular property is satisfied. For example, the mesh simplices may be required to have a certain minimum quality or minimum volume, measured under the Euclidean metric. Operators such as vertex insertion, edge collapse, edge swaps or vertex relocation can be used to achieve this property[50–52]. Examples of edge splits and edge collapses are given in Figure 1.6; edge swaps were discussed in Figure 1.4 when introducing the Delaunay kernel.

50. Michal *et al.*, *Anisotropic Mesh Adaptation through Edge Primitive Operations*. 2012

51. Park *et al.*, *Parallel Anisotropic Tetrahedral Adaptation*. 2008

52. Park, *Anisotropic Output-Based Adaptation with Tetrahedral Cut Cells for Compressible Flows*. 2008



(a) Edge split **(b)** Edge collapse

Figure 1.6: Local edge split and edge collapse operators. (a) Edge uw is too long and gets split, introducing the vertex q. (b) Any of the edges vq, wq or pq might be too short, and a collapse of the vertex q onto either p, q or w would eliminate short edges.

This approach extends well to the anisotropic setting[53,54], provided the geometric quantities such as length and volume are computed in the Riemannian metric space.

53. Alauzet *et al.*, *A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics*. 2016

54. Loseille, *Unstructured Mesh Generation and Adaptation*. 2017

Furthermore, these classical mesh operators can be viewed in a dimension-independent *cavity* framework. The core concept, initially proposed by Coupez[55] replaces the geometric intuition behind the mesh operators with a rigorous mathematical definition. Define a *cavity* as a set of simply-connected simplices, homeomorphic to a ball, to remove from the mesh. All the aforementioned operators can be recovered by appropriately selecting a *re-insertion vertex* and connecting it to the boundary of the removed cavity (forming a *star* in the terminology of Coupez). That is, a local mesh modification by a cavity operator is composed of the following simple steps:

55. Coupez, *Génération de Maillage et Adaptation de Maillage par Optimisation Locale*. 2000

1. Select an appropriate set of elements to remove from the mesh and compute the boundary of these elements,

2. Select a vertex with corresponding coordinates to connect to the boundary of the cavity,

3. Insert the set of elements formed by the connection of the chosen vertex with the boundary of the cavity into the mesh.

Loseille's recent work re-introduces this cavity framework[56]. The methods of Loseille and Coupez differ in how they check for topological validity or enforce the application of an operator. They also schedule their cavity operators differently and it is unclear how to best schedule local operators to satisfy the requirements of an adaptive numerical simulation.

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

Furthermore, Loseille implements his operator for two- and three-dimensional mesh adaptation. Gruau (a student of Coupez) only demonstrated their framework for the special case of a uniform Euclidean metric in four dimensions[57]. This method seems the most promising for generating four-dimensional meshes, however, the ability of the algorithm in a truly anisotropic setting has yet to be demonstrated.

57. Gruau, *Metric Generation for Anisotropic Mesh Adaptation with Numerical Applications to Material Forming Simulation*. 2005

Tremblay also attempted to use local operators to adapt four-dimensional meshes[58]. He used edge splits and collapses but avoids edge swapping by employing a combination of point insertions and collapses to *simulate* an edge swap. His algorithm does not seem capable of producing metric-conforming meshes since the resulting edge lengths are short and the element quality is very poor, even for an analytic metric with a maximum aspect ratio of 10:1. Tremblay further attempts to demonstrate the algorithm for the solution of the unsteady heat equation in $3d + t$ but the resulting mesh is isotropic.

58. Tremblay, *2-D, 3-D and 4-D Anisotropic Mesh Adaptation for the Time-Continuous Space-Time Finite Element Method with Applications to the Incompressible Navier-Stokes Equations*. 2007

Again, we emphasize that a successful anisotropic four-dimensional mesh adaptation capability has yet to be demonstrated.

1.3 *Objectives*

The work in this thesis began with a quest for four-dimensional metric-conforming meshes. The ultimate goal was to provide the meshing tools needed to perform four-dimensional adaptive numerical simulations. As a result, the first objective is to develop the tools for meshing, geometry and visualization of four-dimensional problems. Our presentation of the background material on mesh generation and adaptation suggests the local cavity framework is the most promising for adapting four-dimensional meshes. As such, we strive to extend and demonstrate this method to anisotropic problems in four dimensions. We also aim to fill the gap in the existing mesh adaptation literature in

the discussion of local operator implementation and scheduling. The developed algorithm is compared against existing software implementations for three-dimensional mesh adaptation. Finally we use the developed mesh adaptation algorithm to first perform four-dimensional mesh adaptation on L^2 error control problems. We then demonstrate the first $3d + t$ mesh adaptation driven by the solution to the advection-diffusion equation.

Contributions

The major contribution of this work is the development of *dimension-independent* meshing techniques which are then applied to four-dimensional problems. In fact, with the addition of a few simple predicates (discussed in Appendix C), any dimensional problem can be studied. The primary contributions of this work are summarized as follows:

1. Development of an algorithm and software for four-dimensional metric-conforming mesh adaptation,
2. Validation of the mesh adaptation capability by studying four-dimensional metric-conforming, as well as L^2 error control problems,
3. Demonstration of the first PDE-driven anisotropic unstructured adaptation for unsteady $3d$ problems.

These primary contributions are complemented by other contributions to the literature on mesh generation and adaptation. First, Appendix A discusses our four-dimensional geometry and visualization tools. This geometry description, in fact, is a critical component to our mesh adaptation algorithm. Furthermore, we first investigated the use of isometric embeddings to generate metric-conforming meshes in a dimension-independent manner. The method was, in fact, the first to produce isometric embeddings of arbitrary metric fields[47,48]. The development of the latter technique came along with a dimension-independent algorithm for computing restricted Voronoi diagrams of simplicial meshes (outlined in Appendix B) as well as an algorithm for conforming to an input geometry description with the restricted Voronoi diagram.

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

48. Caplan *et al.*, *Isometric Embedding of Curvilinear Meshes Defined on Riemannian Metric Spaces*. 2018

Outline

Chapter 2 establishes the terminology used in the remainder of this thesis. Chapter 3 develops and demonstrates our four-dimensional mesh adaptation algorithm and compares it against existing implementations for three-dimensional mesh adaptation. Finally, we demonstrate the first four-dimensional mesh adaptation capability on L^2 error

control and advection-diffusion problems in Chapter 4. The latter two chapters employ the geometry description we develop in Appendix A. Conclusions and recommendations for future work are given in Chapter 5.

◁

When reading this text, please keep the opening quote in the back of your mind (in fact, these words were said to me at the onset of my PhD). Though delving into the fourth dimension might seem daunting, maintaining a positive attitude and carefully studying the mathematics behind our algorithms will certainly pay off.

Chapter 2

Preliminaries

Just learn the fundamentals.
Everything else is simple.
— *John H.S. Lee*

This chapter defines the terminology and notation used in the remainder of this thesis. Boldface symbols will be used for vectors and matrices and the scripts i, j, k, l, m will be used for indexing. The scripts d and n are used for dimensions, whether topological or physical.

2.1 Meshes

Before defining a mesh, let us cover the basic building blocks of a mesh, notably, its *elements*. A *n-simplex* is the convex hull of $n+1$ affinely independent points, $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^n$. Any point in the simplex, including the boundary, can be described by

$$\mathbf{x} = \left\{ \sum_{j=0}^{n} \alpha_j \mathbf{p}_j \;\middle|\; \sum_{j=0}^{n} \alpha_j = 1, \text{ and } \alpha_j \geq 0, \forall j \right\} \tag{2.1}$$

where α_j are called the *barycentric coordinates*. The unit-length equilateral ($\boldsymbol{\kappa}_{\Delta_n}$) and unit-length orthogonal-corner ($\boldsymbol{\kappa}_{\perp_n}$) n-simplices – see Figure 2.1 – respectively have volumes of

$$v(\boldsymbol{\kappa}_{\Delta_n}) = \frac{\sqrt{n+1}}{n!\sqrt{2^n}}, \quad v(\boldsymbol{\kappa}_{\perp_n}) = \frac{1}{n!}. \tag{2.2}$$

With the orthogonal corner at the origin, the remaining coordinates of the unit-length orthogonal simplex $\boldsymbol{\kappa}_{\perp_n}$ are $\mathbf{p}_1 = (1, 0, \ldots, 0)^t$, $\mathbf{p}_2 = (0, 1, \ldots, 0)^t$, $\ldots$, $\mathbf{p}_n = (0, 0, \ldots, 1)^t$. The coordinates of the unit-length equilateral simplex can be computed using the properties that

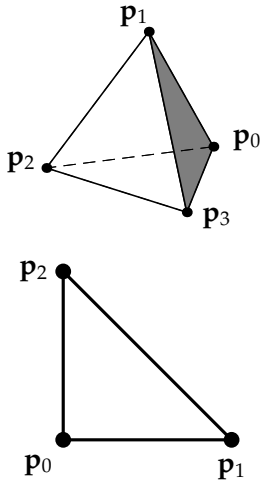


Figure 2.1: Equilateral tetrahedron $\boldsymbol{\kappa}_{\Delta_3}$ (top) and right triangle $\boldsymbol{\kappa}_{\perp_2}$ (bottom). All edges of the equilateral simplex have a length of one. The edges of the orthogonal corner simplex touching the orthogonal corner (at $\mathbf{p}_0$) have a length of one.

◁ **Did you know?**

A 2-simplex is a triangle, a 3-simplex is a tetrahedron and a 4-simplex is a pentatope.

the distance from its centroid $\mathbf{c}$ to any vertex $\mathbf{p}_j$ is equal, and that the dot product between any two vectors $\mathbf{p}_i - \mathbf{c} = -1/n$. These coordinates are listed on the right for the unit equilateral triangle, tetrahedron and pentatope. When the topological dimension of the simplex is clear, we will often drop the subscript for brevity. The unit-length equilateral simplex is used to compute the element metric (Equation 2.19) whereas the orthogonal one is used to describe the reference element on which we define basis functions and quadrature rules for our numerical discretization. The volume of an n-simplex $\boldsymbol{\kappa}$ with vertex coordinates $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^n$ can be computed from

$$v(\boldsymbol{\kappa}) = \frac{1}{n!} \left| \begin{array}{cccc} \mathbf{p}_1 - \mathbf{p}_0 & \mathbf{p}_2 - \mathbf{p}_0 & \cdots & \mathbf{p}_n - \mathbf{p}_0 \end{array} \right|. \tag{2.3}$$

The exact calculation of the determinant in Equation 2.3 for pentatopes is described in Appendix C.

It is further useful to define a *convex n-polytope*, which is the convex hull of m points in $\mathbb{R}^n$, $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_m$[59]. This description is referred to as the *V-representation* (VRep). It is sometimes more convenient to work with the *H-representation* (HRep) which describes the interior of the polytope from the intersection of a finite number of halfspaces. The choice of which representation is used is driven by the particular application.

A n-dimensional polytope is bounded by a set of j-dimensional *facets* for $j < n$ which are, themselves, j-dimensional polytopes. The facets of simplices are also simplices and the number of j-simplices of a n-simplex is given by the binomial coefficient

$$f_{n,j} = \binom{n+1}{j+1}. \tag{2.4}$$

For example, a triangle is bounded by three edges (1-simplices) and 3 vertices (0-simplices) whereas a tetrahedron is bounded by 4 triangles (2-simplices), 6 edges (1-simplices) and 4 vertices (0-simplices).

Another special convex polytope is the n-cube which is the n-dimensional analogue of the square and cube. It is bounded by j-cubes ($j < n$) and the number of these entities is given by

$$f_{n,j} = 2^{n-j}\binom{n}{j}. \tag{2.5}$$

The 2^n vertices of the unit n-cube centered at the origin are computed by permuting the n-tuple with entries $\pm\frac{1}{2}$. The 4-cube is called a *tesseract*.

Let $\mathcal{V}$ be a set of n_v vertices: $\mathcal{V} = \{\mathbf{v}_i \,|\, \mathbf{v}_i \in \mathbb{R}^N, i = 1, \ldots, n_v\}$. A *mesh*, $\mathcal{M}$ of some domain Ω, is a pair $(\mathcal{V}, \mathcal{T})$ where $\mathcal{T}$ represents the *topology* that references the set of vertices $\mathcal{V}$. We describe our meshes

Equilateral triangle ($\boldsymbol{\kappa}_{\Delta_2}$) coordinates:
$\mathbf{p}_0 = (\frac{1}{\sqrt{3}}, 0)$,
$\mathbf{p}_1 = (-\frac{1}{2\sqrt{3}}, \frac{1}{2})$,
$\mathbf{p}_2 = (-\frac{1}{2\sqrt{3}}, -\frac{1}{2})$.

Equilateral tetrahedron ($\boldsymbol{\kappa}_{\Delta_3}$) coordinates:
$\mathbf{p}_0 = (\frac{1}{2}\sqrt{\frac{3}{2}}, 0, 0)$,
$\mathbf{p}_1 = (-\frac{1}{2\sqrt{6}}, \frac{1}{\sqrt{3}}, 0)$,
$\mathbf{p}_2 = (-\frac{1}{2\sqrt{6}}, -\frac{1}{2\sqrt{3}}, \frac{1}{2})$,
$\mathbf{p}_3 = (-\frac{1}{2\sqrt{6}}, -\frac{1}{2\sqrt{3}}, -\frac{1}{2})$.

Equilateral pentatope ($\boldsymbol{\kappa}_{\Delta_4}$) coordinates:
$\mathbf{p}_0 = (\sqrt{\frac{2}{5}}, 0, 0, 0)$,
$\mathbf{p}_1 = (-\frac{1}{4}\sqrt{\frac{2}{5}}, \frac{\sqrt{6}}{4}, 0, 0)$,
$\mathbf{p}_2 = (-\frac{1}{4}\sqrt{\frac{2}{5}}, -\frac{1}{4}\sqrt{\frac{2}{3}}, \sqrt{\frac{1}{6}}, 0)$,
$\mathbf{p}_3 = (-\frac{1}{4}\sqrt{\frac{2}{5}}, -\frac{1}{4}\sqrt{\frac{2}{3}}, -\frac{1}{2}\sqrt{\frac{1}{3}}, \frac{1}{2})$,
$\mathbf{p}_4 = (-\frac{1}{4}\sqrt{\frac{2}{5}}, -\frac{1}{4}\sqrt{\frac{2}{3}}, -\frac{1}{2}\sqrt{\frac{1}{3}}, -\frac{1}{2})$.

59. Grünbaum, *"Convex Polytopes"*. 2003

◁ **Did you know?**

A pentatope is bounded by 5 tetrahedra, 10 triangles, 10 edges and 5 vertices. A tesseract is bounded by 8 cubes, 24 squares, 32 edges and 16 vertices.

using the VRep of each polytope, so $\mathcal{T}$ is a collection of element indices:

$$\mathcal{T} = \kappa_0 \cup \kappa_1 \cdots \cup \kappa_m \tag{2.6}$$

where the indices correspond to the vertices in $\mathcal{V}$. Following the terminology of Coupez[55], it is important to define a *mesh topology*.

55. Coupez, *Génération de Maillage et Adaptation de Maillage par Optimisation Locale.* 2000

Definition 1 (Mesh topology)**.** *Let $\mathcal{V}$ be a set of vertices in some domain Ω and $\mathcal{T}$ be a set of n-polytopes with vertices in $\mathcal{V}$. Let $\mathcal{F}$ be the set of faces $(n-1)$-facets of $\mathcal{T}$. $\mathcal{T}$ is called a mesh topology if*

1. $\text{card}(f \cap \mathcal{T}) \leq 2$, $\forall f \in \mathcal{F}$,
2. *$(\mathcal{V}, \partial\mathcal{T})$ is a mesh of $\partial\Omega$.*

Condition 1 simply means that every face of the mesh touches no more than two elements and Condition 2 means that the boundary of the mesh represents the boundary of the domain.

This *boundary*, denoted by $\partial\mathcal{M} = (\mathcal{V}, \partial\mathcal{T})$, is a mesh itself, and is the set of $(n-1)$-facets that appear as a facet of only *one* polytope of the mesh. Any $(n-1)$-facet that is shared by two polytopes is not on the boundary, but is *interior*. The extraction of the boundary is purely topological (i.e., we only need to determine $\partial\mathcal{T}$).

In the general case, every physical element $\boldsymbol{\kappa}$ is a n-polytope ($n \leq N$), therefore, for vertices in $\mathbb{R}^N$, the mesh $\mathcal{M}$ has *topological dimension* n and *physical dimension* N. Unless otherwise stated, we will always take $N = n$. The vertices of an element κ will be denoted by the VRep $\mathcal{V}(\kappa)$ which gives a physical meaning to the purely topological element $\boldsymbol{\kappa}$. We will often refer to this physical element as $\boldsymbol{\kappa}$ to distinguish it from the topological one, κ. In this work, note that $\boldsymbol{\kappa} = \text{conv}(\mathcal{V}(\kappa))$, i.e., the convex hull of the vertices retrieved from the indices in κ.

◁ **Example: a topology with four elements.**

$\kappa_0 = \{5 \quad 3 \quad 1\}$
$\kappa_1 = \{9 \quad 0 \quad 3 \quad 5 \quad 2\}$
$\kappa_2 = \{2 \quad 8 \quad 4\}$
$\kappa_3 = \{1 \quad 7 \quad 6 \quad 3\}$

We strive to generate simplicial (triangular, tetrahedral, pentatopal) meshes for adaptive numerical simulations but also discuss polytopal (polygonal, polyhedral) meshes when exploring visualization techniques and Voronoi diagrams. In the former case, it is sufficient to simply define the mesh using $\mathcal{T}$ and $\mathcal{V}$. In the polytopal case, we also use the *vertex-facet incidence matrix* which is the list of $(n-1)$-facets touched by each vertex. This allows us to convert between the HRep and VRep of a polytope, which we frequently use in Appendix A as well as in our Voronoi diagram calculation[47].

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings.* 2017

As we will see in Chapter 3, a frequent operation in mesh adaptation is the identification of the unique list of *edges*. For a simplicial mesh, this is given by the relation on the vertices of the element κ,

$$\mathcal{E}(\mathcal{T}) = \{e \mid e = \kappa \times \kappa,\ \forall \kappa \in \mathcal{T}\}\,. \tag{2.7}$$

This is a purely topological operation; that is, only the topology $\mathcal{T}$ is required to construct the edges. The identification of the edges of a polytopal mesh is discussed in Appendix A.

2.2 *Metric fields*

A *metric tensor* $\mathbf{m}$ is a $n \times n$ symmetric, positive-definite (SPD) tensor which transforms the way distance, volume, angles and other geometric properties are measured. Given a constant metric field, the length of the vector between points $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{q} \in \mathbb{R}^n$ is computed as

$$\ell_{\mathbf{m}}(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{q} - \mathbf{p})^t \mathbf{m} (\mathbf{q} - \mathbf{p})}. \tag{2.8}$$

The volume of a simplex $\boldsymbol{\kappa}$ is transformed by a constant metric tensor as

$$v_{\mathbf{m}}(\boldsymbol{\kappa}) = \sqrt{\det \mathbf{m}}\, v(\boldsymbol{\kappa}) \tag{2.9}$$

where $v(\boldsymbol{\kappa})$ denotes the volume measured under the usual Euclidean metric.

It is convenient to think about a metric tensor from the eigendecomposition:

$$\mathbf{m} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^t = \mathbf{Q}\mathbf{H}^{-2}\mathbf{Q}^t \tag{2.10}$$

where $\mathbf{H} = \operatorname{diag}(h_0, h_1, \ldots, h_n)$ is the diagonal matrix of principal lengths and the n eigenvectors $\mathbf{Q}$ are the principal directions. This decomposition enables us to think of metric tensors as n-dimensional hyperellipsoids; an example for the case $n = 2$ is shown in Figure 2.2.

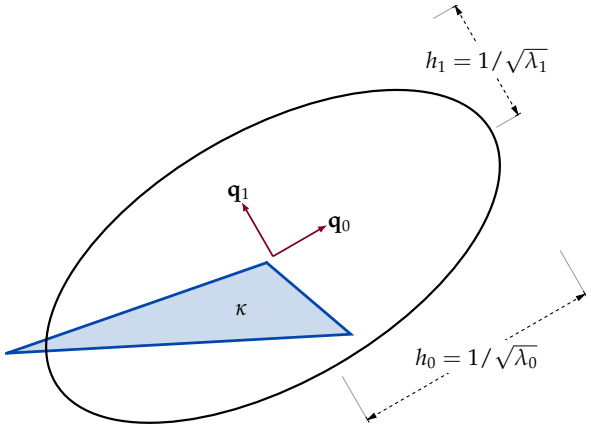


Figure 2.2: Ellipse representation of a metric tensor $\mathbf{m}$ in $2d$. The principal directions are given by the eigenvectors $\mathbf{q}_0$ and $\mathbf{q}_1$ of $\mathbf{m}$ and the semi-major and semi-minor axes are related to the eigenvalues $\boldsymbol{\Lambda}$ of $\mathbf{m}$ via $\mathbf{H} = \boldsymbol{\Lambda}^{-1/2}$. The shaded blue triangle $\boldsymbol{\kappa}$ is said to *conform* to this metric because the lengths of its edges relative to the axes of the ellipse are approximately unit (see Section 2.3). This representation extends to higher dimensions, notably as an ellipsoid in $3d$ and a hyperellipsoid in $4d$.

We work with *Riemannian metric tensors*, that is, spatially varying fields of metric tensors. It is convenient to think of Riemannian metric tensors as the induced metric tensor of some *n-manifold*. Under a Riemannian metric, length and volume should be computed using the usual definitions from differential geometry[60]:

$$\ell_{\mathbf{m}}(\mathbf{p}, \mathbf{q}) = \int_{\boldsymbol{\Gamma}} \sqrt{\boldsymbol{\Gamma}'(t) \mathbf{m}(\boldsymbol{\Gamma}(t)) \boldsymbol{\Gamma}'(t)} \mathrm{d}t \tag{2.11}$$

where $\boldsymbol{\Gamma}(t)$ is the distance-minimizing curve, or *geodesic*, between $\mathbf{p}$ and $\mathbf{q}$. Here, we take $\boldsymbol{\Gamma}(t)$ to be the edge between $\mathbf{p}$ and $\mathbf{q}$, so $\boldsymbol{\Gamma}(t) =$

60. Carmo, *Differential Geometry of Curves and Surfaces*. 1976

$\mathbf{p} + t(\mathbf{q} - \mathbf{p})$. The integration in Equation 2.11 can be carried out with a numerical quadrature scheme but we find it useful to approximate the calculation by assuming the principal lengths, $h(t)$, to vary according to a geometric variation law[61]. In this case, $h(t) = h_p^t h_q^{1-t}$, $t \in [0,1]$ where h_p and h_q are the lengths of the vector $\mathbf{e} = \mathbf{q} - \mathbf{p}$ computed under the metrics evaluated at points $\mathbf{p}$ and $\mathbf{q}$. That is, $h_p = \ell_{\mathbf{m}(\mathbf{p})}(\mathbf{p}, \mathbf{q})/||\mathbf{e}||$ and $h_q = \ell_{\mathbf{m}(\mathbf{q})}(\mathbf{p}, \mathbf{q})/||\mathbf{e}||$. Under this assumption, the length of $\mathbf{e}$ is approximated as

$$\ell_{\mathbf{m}}(\mathbf{p}, \mathbf{q}) \approx \ell_{\mathbf{m}(\mathbf{p})} \frac{r-1}{r \log r} \quad \text{with } r \equiv \frac{\ell_{\mathbf{m}(\mathbf{p})}}{\ell_{\mathbf{m}(\mathbf{q})}}. \tag{2.12}$$

61. Alauzet, *Size Gradation Control of Anisotropic Meshes*. 2010

◁ **Note:**

The difference between Equation 2.8 and 2.12 is that the former assumes a constant metric whereas the latter assumes a spatially varying metric.

In later chapters, when we write $\ell_{\mathbf{m}}(e)$ where e is an edge of the mesh, it should be understood as $\ell_{\mathbf{m}}(\mathbf{e}_0, \mathbf{e}_1)$ where $\mathbf{e}_0$ and $\mathbf{e}_1$ are the coordinates of the edge endpoint vertices. The volume of a physical element $\boldsymbol{\kappa}$ under a Riemannian metric field $\mathbf{m}$ should also be measured along the Riemannian manifold:

$$v_{\mathbf{m}}(\boldsymbol{\kappa}) = \int_{\boldsymbol{\kappa}} \sqrt{\det \mathbf{m}(\mathbf{x})}\, \mathrm{d}\mathbf{x}. \tag{2.13}$$

Again, a quadrature scheme can be used to approximate this integral but, in consistency with the Unstructured Grid Adaptation Working Group[62], we prefer to approximate the volume as

$$v_{\mathbf{m}}(\boldsymbol{\kappa}) \approx \sqrt{\det \mathbf{m}_\nu}\, v(\boldsymbol{\kappa}), \quad \text{with } \nu = \arg\max_{\nu \in \kappa} \det \mathbf{m}_\nu. \tag{2.14}$$

62. Ibanez *et al.*, *First Benchmark of the Unstructured Grid Adaptation Working Group*. 2017

In other words, the volume under the Riemannian metric is approximated as the volume under a constant metric $\mathbf{m}_\nu$. The volume of the simplex under the usual Euclidean metric is denoted by $v(\boldsymbol{\kappa})$. This measure is quicker to evaluate than using a quadrature scheme, which is useful in situations in which the quality (hence, volume) need to be evaluated several times, such as in a mesh adaptation tool.

It is also important to define the interpolation of metrics which can be used to evaluate metrics on a background mesh obtained during an adaptation request. Here, we use the Log-Euclidean interpolation[63] which interpolates a set of m metrics $\{\mathbf{m}_j\}$ with interpolation weights $\mathbf{w} \in \mathbb{R}^m$ as

$$\mathbf{m}_{\text{interp}} = \exp\left(\sum_{j=1}^{m} w_j \log\left(\mathbf{m}_j\right)\right). \tag{2.15}$$

63. Arsigny *et al.*, *Log-Euclidean Metrics for Fast and Simple Calculus on Diffusion Tensors*. 2006

Note that the exponential and logarithm for tensors can be computed from the eigendecomposition $\mathbf{m} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^t$, yielding $\exp(\mathbf{m}) = \mathbf{Q}\exp(\boldsymbol{\Lambda})\mathbf{Q}^t$ and $\log(\mathbf{m}) = \mathbf{Q}\log(\boldsymbol{\Lambda})\mathbf{Q}^t$.

When computing the metric of a point $\mathbf{p}$ using a simplicial background mesh, the simplex containing the query point is first located ($\boldsymbol{\kappa}_b$) and then the interpolation weights become the barycentric coordinates of $\mathbf{p}$ with respect to $\boldsymbol{\kappa}_b$. The interpolant metrics are the metrics defined at the vertices of the background simplex $\boldsymbol{\kappa}_b$. See Figure 2.3.

Target metric

In order to interpolate new metric tensors when vertices are moved or created, it is more convenient for mesh adaptation algorithms to employ a continuous (vertex-valued) representation of the target metric field. For this reason, we expect the mesh adaptation algorithm to provide a set of target metrics $\{\mathbf{m}_t\}$ defined at the vertices of the input mesh. This input mesh then becomes a *static* background mesh on which new metrics can be evaluated by marching through element neighbours, locating a containing element and using barycentric interpolation on the vertex metrics of the containing element. For efficiency, our mesh adaptation software retains information as to which element in the background mesh contains every vertex in the working mesh. Vertices created from edge splits can lie outside this background mesh (which can happen with curved geometries). The metric at the new point is then evaluated as the log-Euclidean average of the metrics stored at the edge endpoints – see Figure 2.3.

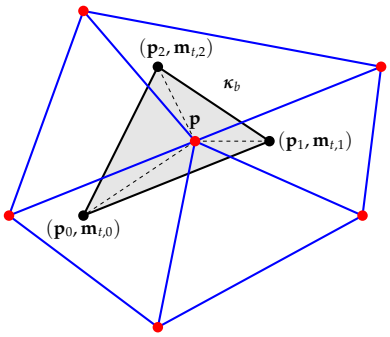


Figure 2.3: Interpolation of metric tensors using a background mesh. The background simplex κ_b (in shaded gray) containing the query point $\mathbf{p}$ is first located and the barycentric coordinates of $\mathbf{p}$ in κ_b are used to interpolate the target metrics stored at the vertices of the background simplex. The blue edges are those of the working mesh. After the interpolation, the vertex at $\mathbf{p}$ retains the information that it is contained within κ_b. This makes the search through the background mesh more efficient since κ_b can be used as a starting guess for subsequent interpolations.

2.3 *Metric-conforming meshing*

An element is said to be *unit* with respect to some metric if all its edge lengths are unit under the metric and the metric volume of the element is that of the ideal element (v_{Δ_n} for a n-simplex). A mesh is then *unit* if all its elements are unit with respect to the metric.

In general, it is not possible to satisfy these conditions and they need to be relaxed[64]. An element is said to be *quasi-unit* if its edge lengths satisfy

$$\frac{1}{\sqrt{2}} \leq \ell_{\mathbf{m}}(e) \leq \sqrt{2}, \forall e \in \mathcal{E}(\kappa). \tag{2.16}$$

Let us justify the use of the quasi-unit length bounds of $[1/\sqrt{2}, \sqrt{2}]$ which is frequently used in the literature[64][65]. In the words of Frey: *the coefficient $\sqrt{2}$ is related to the fact that an edge can be split if the lengths of the two sub-edges minimize the error distance to the unit length as compared with the initial length*[65]. First, consider the upper bound, $\ell_{\max}$. The goal is to set $\ell_{\max}$ as close to one as possible with the constraint imposed by Frey's suggestion. We can therefore pose the following optimization problem

$$\ell_{\max} = \arg\min_{\alpha} \ (\alpha - 1)^2 \quad \text{such that } g(\alpha) = 2\left(\frac{\alpha}{2} - 1\right)^2 - (\alpha - 1)^2 \leq 0$$

Solving this optimization problem leads to $g(\alpha) = 0$, therefore, $\ell_{\max} = \sqrt{2}$. The lower bound of the quasi-unit range is $\sqrt{2}/2$ since the two sub-edges will have a length of $\ell_{\max}/2$ and we want to avoid cycling between edge refinement and coarsening in an adaptation loop.

64. Loseille *et al.*, *Continuous Mesh Framework Part I: Well-Posed Continuous Interpolation Error*. 2011

65. Frey *et al.*, *Mesh Generation: Application to Finite Elements: Second Edition*. 2008

Instead of directly controlling the volume, it is more practical to control some chosen *quality* of the elements. For simplex elements, we employ the quality measure

$$q_{\mathbf{m}}(\kappa) = \beta_n \frac{v_{\mathbf{m}}(\kappa)^{2/n}}{\sum_{e \in \mathcal{E}(\kappa)} \ell^2_{\mathbf{m}}(e)} \in [0, 1], \tag{2.17}$$

where β_n is a normalizing factor such that the ideal simplex has unit quality. Note that a degenerate simplex (with zero volume) has a quality of zero. When calculating the volume and edge lengths in Equation 2.17, the same metric $\mathbf{m}$ is used which ensures the quality is bounded between zero and one. Again, in consistency with the UGAWG, we use the metric defined in Equation 2.14 to compute the quality in Equation 2.17.

Ideally, the quality of the simplices should be greater than 0.8[66] but this is not always possible. An average simplex quality of approximately 0.8 is acceptable though, again, not always possible, especially in the presence of a noisy target metric. A metric-conforming simplex is a simplex whose edge lengths are in the bounds of Equation 2.16 and whose quality under the metric is greater than 0.8. An example of a metric-conforming triangle is given in Figure 2.2. The mesh is then said to be metric-conforming if all simplices are metric-conforming, though it may not be possible for all elements to be quasi-unit under the metric. In general, the edge lengths will exhibit the behaviour of Figure 2.4 whereby the edge lengths are within the bounds of Equation 2.16 and the average edge length slightly deviates from unity.

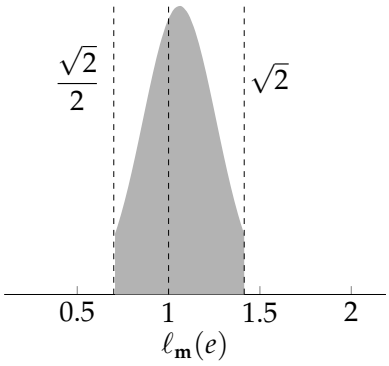


Figure 2.4: Typical distribution of edge lengths for a metric-conforming mesh in which all edge lengths are within the bounds of Equation 2.16.

66. Loseille, *Metric-Orthogonal Anisotropic Mesh Generation*. 2014

Mesh metric

A mesh itself defines a Riemannian metric, which we denote as $\mathbf{m}_{\mathcal{M}}$. In the same way a mesh is the union of a set of elements to cover a domain, the *mesh metric* is the discontinuous set of *element metrics*:

$$\mathbf{m}_{\mathcal{M}} = \bigcup_{\kappa \in \mathcal{M}} \mathbf{m}_\kappa. \tag{2.18}$$

Each element metric $\mathbf{m}_\kappa$ can be computed using the Jacobian $\mathbf{A} \in \mathbb{R}^{n \times n}$ of the transformation from the reference unit-length element to the physical element ($\boldsymbol{\phi}$):

$$\mathbf{m}_\kappa = \left(\mathbf{A}\mathbf{A}^t\right)^{-1}. \tag{2.19}$$

For an n-simplex, it may be more convenient to compute $\mathbf{A}$ from a two-step transformation:

$$\mathbf{m}_\kappa = \left(\mathbf{A}\mathbf{A}^t\right)^{-1} = \left(\mathbf{A}_\perp \mathbf{A}_\Delta^{-1} \mathbf{A}_\Delta^{-t} \mathbf{A}_\perp^t\right)^{-1}, \tag{2.20}$$

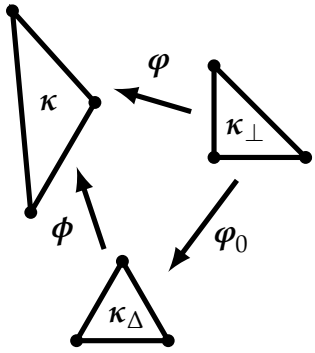
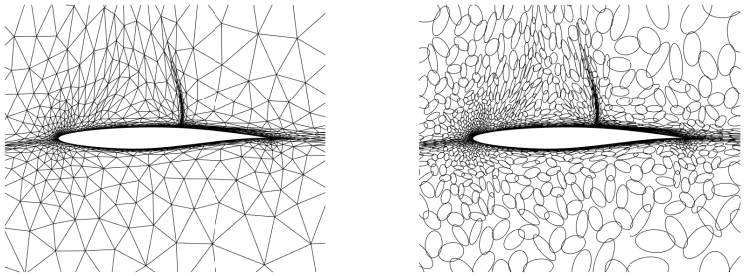


Figure 2.5: Relationship between the physical (κ) and reference elements (κ_Δ and $\kappa_\perp$) for the calculation of the element-implied metric.

where $\mathbf{A}_\Delta$ is the Jacobian matrix of the transformation from the orthogonal simplex to the equilateral one ($\boldsymbol{\varphi}_0$) and $\mathbf{A}_\perp$ is that from the orthogonal simplex to the physical element ($\boldsymbol{\varphi}$). See Figure 2.5.

For n-simplices, this element metric can also be computed by solving for the $n(n+1)/2$ unique values in the tensor from the $n(n+1)/2$ equations that result by requiring that the length of each simplex edge computed under $\mathbf{m}_\kappa$ is unit in Equation 2.8. An illustration of the mesh metric is given in Figure 2.6. This duality enables the discrete mesh optimization problem at the heart of our adaptation algorithm to be posed in terms of a continuous metric.



(a) Mesh **(b)** Metric

Figure 2.6: Illustration of the duality between a mesh and its metric.

Various methods exist to transform the discontinuous (element-valued) mesh metric field, $\mathbf{m}_\mathcal{M}$, to a continuous (vertex-valued) one $\mathbf{m}_I \in \mathcal{C}^0(\Omega)$. Here, we compute the continuous mesh metric by solving an optimization problem. The goal of the optimization is to match the *complexity* of the continuous mesh metric with that of the actual mesh. In particular, the complexity of the continuous mesh metric is described by

$$c(\mathbf{m}_I) = \sum_{\kappa \in \mathcal{M}} v_{\mathbf{m}_I}(\kappa) = \sum_{\kappa \in \mathcal{M}} \int_\kappa \sqrt{\det \mathbf{m}_I(\mathbf{x})}\, \mathrm{d}\mathbf{x}, \tag{2.21}$$

and the complexity of the mesh is $c(\mathcal{M}) = |\mathcal{M}| v_{\Delta_n}$. In contrast to the approximation in Equation 2.14, the integral in Equation 2.21 is evaluated with numerical quadrature since the number of times it is evaluated is much less than in a mesh adaptation tool. Furthermore, since the edge lengths of the mesh under its own metric should be unit, the objective function also employs a penalty term

$$p(\mathbf{m}_I) = \sum_{e \in \mathcal{E}(\mathcal{M})} \delta_e^n, \tag{2.22}$$

with $\delta_e = [\ell_{\mathbf{m}_I}(e) > \sqrt{2}](\ell_{\mathbf{m}_I}(e) - \sqrt{2}) + [\ell_{\mathbf{m}_I}(e) < \sqrt{2}/2](\sqrt{2}/2 - \ell_{\mathbf{m}_I}(e))$ (note the use of the Iverson bracket). The resulting objective function is

$$f(\mathbf{m}_I; \mathcal{M}) = (c(\mathbf{m}_I) - c(\mathcal{M}))^2 + p(\mathbf{m}_I)^2. \tag{2.23}$$

This objective function may not provide an accurate calculation of the implied metric when the edge lengths are within the quasi-unit range. An alternative may include a volume- or quality-based term to improve the implied metric. The latter is not done in this work which we identify as a limitation.

◁ **Limitation:**

When the edge lengths are in the quasi-unit bounds (Equation 2.16), the optimizer will only be driven by the cost term in Equation 2.23. A volume- or quality-based term should be added to the objective function to find a better representation of the continuous metric field implied by the mesh.

2.4 *Numerical discretization of partial differential equations*

Model problem

The ultimate goal of this work is to demonstrate a fully unstructured $n = 3d + t$ spacetime mesh adaptation driven by the solution to a partial differential equation (PDE). As such, it is appropriate to describe our model problem and discretization mechanics.

We seek the solution $u(\mathbf{x}, t) \in \mathbb{R}$ to the linear advection-diffusion equation,

$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{c}u - \nu \nabla u) = s(\mathbf{x}, t), \quad \forall \mathbf{x} \in \Omega, t \in [0, T] \tag{2.24}$$

where $\mathbf{c} \in \mathbb{R}^d$ is the convection velocity, ν is the viscosity, $s(\mathbf{x}, t)$ is the source term and $\Omega \subset \mathbb{R}^d$ is the domain in which we wish to solve the equations between times $t = 0$ and $t = T$.

The above can be recast in a spacetime formulation:

$$\widehat{\nabla} \cdot [\hat{\mathbf{c}}u - \mathbf{f}_d(u)] = s(\hat{\mathbf{x}}), \quad \forall \hat{\mathbf{x}} \in \widehat{\Omega}, \tag{2.25}$$

where the spacetime differential operator is $\widehat{\nabla} \equiv [\nabla, \partial/\partial t]^t$ and the spacetime coordinates are given by $\hat{\mathbf{x}} \equiv [\mathbf{x}, t]^t$ which are contained in the spacetime domain $\widehat{\Omega} \equiv \Omega \times [0, T] \subset \mathbb{R}^{d+1}$ (see Figure 2.7). The spacetime convective velocity ($\hat{\mathbf{c}}$) and diffusive flux ($\mathbf{f}_d$) are, respectively,

$$\hat{\mathbf{c}} \equiv \begin{bmatrix} \mathbf{c} \\ 1 \end{bmatrix}, \quad \mathbf{f}_d(u) \equiv \begin{bmatrix} \nu \nabla u \\ 0 \end{bmatrix}. \tag{2.26}$$

In this work, Dirichlet boundary conditions are applied on the spatial boundaries ($\partial\Omega$) and the initial condition is specified with $u(\mathbf{x}, 0)$.

For brevity, we will drop the hat notation in the following but it should be understood that we are employing the spacetime formulation of Equation 2.25 and solving the problem in $\widehat{\Omega}$.

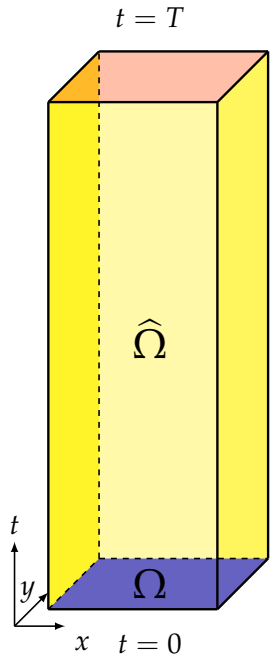


Figure 2.7: Example of a spacetime domain $\widehat{\Omega}$ for solving a $2d$ unsteady problem within a square (Ω). The initial condition is applied at $t = 0$ (the blue face) and the spatial boundary conditions are applied on the yellow faces. The final time ($t = T$) is shown in red.

Discretization

We will demonstrate the adaptation mechanics using the discontinuous Galerkin method, which we review here.

Let $u \in V$, with V an appropriate function space. The weak form of the governing equations is obtained by multiplying Equation 2.25

by some test function $v \in V$ and integrating by parts over the domain, yielding the semilinear residual $R(\cdot,\cdot) : V \times V \to \mathbb{R}$:

$$R(u,v) \equiv f(v) - a(u,v) = 0, \quad \forall\, v \in V, \tag{2.27}$$

where $f(\cdot)$ is a linear form obtained from the integration of the source term and also accounts for the Dirichlet boundary conditions. The bilinear form $a(\cdot,\cdot)$ is obtained from the integration by parts of the convective and diffusive terms.

In the finite-dimensional setting, we decompose Ω into a simplicial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$ and introduce the discontinuous finite element space

$$V_{h,p} = \left\{ v \in L^2(\Omega) : v_{h,p} \circ \varphi_q(\boldsymbol{\kappa}) \in \mathcal{P}^p(\boldsymbol{\kappa}_0), \forall\, \boldsymbol{\kappa} \in \mathcal{M} \right\}, \tag{2.28}$$

where $\varphi_q(\boldsymbol{\kappa})$ is the q-th order diffeomorphic mapping from physical element $\boldsymbol{\kappa}$ to master element $\boldsymbol{\kappa}_0$ and $\mathcal{P}^p(\boldsymbol{\kappa}_0)$ denotes the complete p-th order polynomial space on the reference element $\boldsymbol{\kappa}_0$. Since this work focuses on simplicial meshes, $\boldsymbol{\kappa}_0$ is the reference n-simplex with an orthogonal corner ($\boldsymbol{\kappa}_{\perp_n}$). We also restrict our attention to linear elements in this work ($q = 1$).

Representing the discrete solution as $u_{h,p} \in V_{h,p}$ and taking $v_{h,p} \in V_{h,p}$ the discrete residual operator $R_{h,p}(\cdot,\cdot) : V_{h,p} \times V_{h,p} \to \mathbb{R}$ becomes

$$R_{h,p}(u_{h,p}, v_{h,p}) \equiv f_{h,p}(u_{h,p}) - a_{h,p}(u_{h,p}, v_{h,p}) = 0. \tag{2.29}$$

Equation 2.29 is linear in $u_{h,p}$ and we solve the resulting system of equations using PETSc[67]. Roe's approximate Riemann solver[68] is used to discretize the convective flux and the second form of Bassi and Rebay for the diffusive flux[69]. Boundary conditions are weakly imposed by prescribing the fluxes on faces that lie on $\partial\Omega$.

For four-dimensional problems, the integrals in Equation 2.29 are approximated with the conical product of Stroud[70]. Furthermore, nodal Lagrange basis functions are used to represent the polynomial space of the solution.

67. Balay *et al.*, *PETSc Users Manual*. 2017

68. Roe, *Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes*. 1981

69. Bassi *et al.*, *GMRES Discontinuous Galerkin Solution of the Compressible Navier-Stokes Equations*. 2000

70. Stroud, *Approximate Calculation of Multiple Integrals*. 1971

2.5 *Mesh Optimization via Error Sampling and Synthesis*

The adaptation algorithm used in this work is an extension of the Mesh Optimization via Error Sampling and Synthesis (MOESS) framework of Yano[3] to four-dimensional problems. The method relies on an error localization method such that error models can be synthesized from sampled data and optimized to drive the creation of the next mesh in the adaptation sequence.

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

In general, the mesh adaptation problem is posed as: *find the optimal mesh $\mathcal{M}^*$ with a complexity no greater than c_0 that minimizes the error $\mathcal{E}$ in*

the output of interest. Mathematically,

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \ \mathcal{E}(\mathcal{M}), \quad \text{such that } c(\mathcal{M}) \leq c_0. \tag{2.30}$$

Unfortunately, this discrete optimization problem is intractable. However, the duality between the mesh and its metric can be used to recast the optimization as a continuous one for the metric field:

$$\mathbf{m}^* = \arg\min_{\mathbf{m}} \ \mathcal{E}(\mathbf{m}), \quad \text{such that } c(\mathbf{m}) \leq c_0. \tag{2.31}$$

In this light, the metric can be optimized instead of directly optimizing the mesh. With this metric, a metric-conforming mesher would then return the next mesh in the adaptation sequence. In order to compute this optimal metric, we need to develop techniques for estimating and modeling the error in the current mesh.

Error estimation

In the most general setting, our output of interest is computed by some functional $\mathcal{J}(\cdot) : V \to \mathbb{R}$. The true error in the discretization is

$$\mathcal{E}_{\text{true}} \equiv \mathcal{J}(u) - \mathcal{J}_{h,p}(u_{h,p}). \tag{2.32}$$

where $\mathcal{J}_{h,p}(\cdot) : V_{h,p} \to \mathbb{R}$ calculates the discrete version of the output. Since the exact output is not known, the dual-weighted residual (DWR) method is used to estimate Equation (2.32). This method, proposed by Becker and Rannacher[71], weights the residual operator by the solution to the adjoint of the governing PDE:

$$\mathcal{E}_{\text{true}} = R(u_{h,p}, \psi), \tag{2.33}$$

where $\psi \in W \equiv V + V_{h,p}$ is the exact solution to the dual (adjoint) problem,

$$\bar{R}'_{h,p}[u, u_{h,p}](w, \psi) = -\bar{\mathcal{J}}'_{h,p}[u, u_{h,p}](w), \quad \forall\, w \in W, \tag{2.34}$$

where $\bar{R}'_{h,p}[u, u_{h,p}](\cdot,\cdot) : W \times W \to \mathbb{R}$ and $\bar{\mathcal{J}}'_{h,p}[u, u_{h,p}](\cdot) : W \to \mathbb{R}$ are the mean value linearizations of the residual operator and output functional, respectively, defined by

$$\bar{R}'_{h,p}[u, u_{h,p}](w, v) \equiv \int_0^1 R'_{h,p}[\theta u + (1-\theta)u_{h,p}](w, v)\, \mathrm{d}\theta, \tag{2.35}$$

$$\bar{\mathcal{J}}'_{h,p}[u, u_{h,p}](w) \equiv \int_0^1 \mathcal{J}'_{h,p}[\theta u + (1-\theta)u_{h,p}](w)\, \mathrm{d}\theta, \tag{2.36}$$

where $R'_{h,p}[z](\cdot,\cdot)$ and $\mathcal{J}'_{h,p}[z](\cdot)$ denote the Fréchet derivative of $R_{h,p}(\cdot,\cdot)$ and $\mathcal{J}_{h,p}(\cdot)$ with respect to the first argument evaluated about the state z.

◁ **Note:**

The general setting for error estimation is presented here because the output functional considered in Chapter 4 is nonlinear.

71. Becker *et al.*, *An Optimal Control Approach to A Posteriori Error Estimation in Finite Element Methods*. 2001

◁ **Note:**

For our linear PDE:

$\bar{R}'_{h,p}[u, u_{h,p}](w, \psi) = -a_{h,p}(w, \psi)$

which is the transpose of the bilinear form $a_{h,p}(\cdot,\cdot)$.

Since the exact adjoint solution is unknown, an approximate solution, $\psi_{h,p+1} \in V_{h,p+1}$, satisfying

$$\bar{R}'[u_{h,p}](v_{h,p+1}, \psi_{h,p+1}) = -\bar{\jmath}'_{h,p}[u_{h,p}](v_{h,p+1}), \quad \forall\, v_{h,p+1} \in V_{h,p+1}, \tag{2.37}$$

is obtained where $\mathcal{M}$ is fixed but the solution space of the adjoint equation is enriched to $p+1$; note this is necessary to ensure the enriched residual operator acting on a prolongated version of $u_{h,p}$ to $V_{h,p+1}$ does not vanish. The DWR estimate employed in this work thus takes the form

$$\mathcal{E}_{\text{true}} \approx R_{h,p}(u_{h,p}, \psi_{h,p+1}). \tag{2.38}$$

This error can be localized over each element $\boldsymbol{\kappa}$ by restricting the evaluation of Equation (2.38) to each element,

$$\eta_{\boldsymbol{\kappa}} \equiv |R_{h,p}(u_{h,p}, \psi_{h,p+1}|_{\boldsymbol{\kappa}})|. \tag{2.39}$$

This gives an indication of the local elemental error and is useful in driving an adaptive process. In Chapter 4, we will refer to Equation 2.38 as the *error estimate* and will be simply denoted by $\mathcal{E}$. The *error indicator* will refer to the sum of the element errors of Equation 2.39 and will be denoted by $\mathcal{E}_I$:

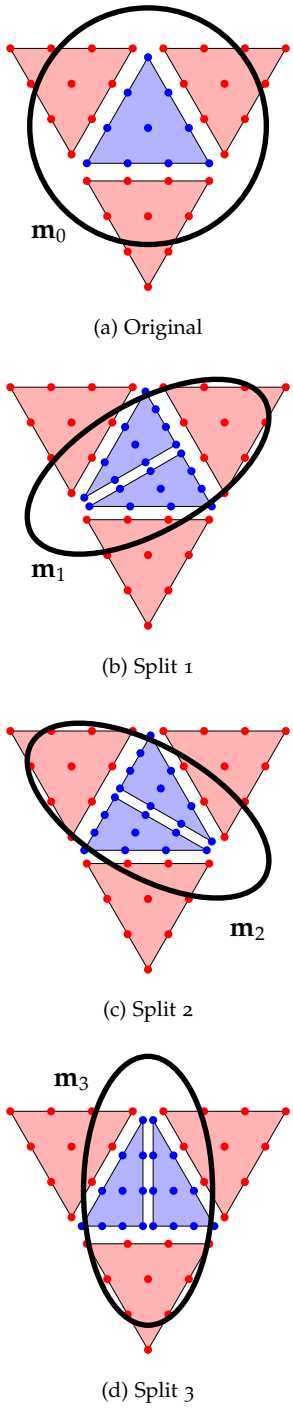
$$\mathcal{E}_I = \sum_{\boldsymbol{\kappa} \in \mathcal{M}} \eta_{\boldsymbol{\kappa}}. \tag{2.40}$$

Error Sampling

Equation 2.39 gives a measure of the local error over an element $\boldsymbol{\kappa}_0 \in \mathcal{M}$ due to the discretization. Each element, $\boldsymbol{\kappa}_0$, can be further split and the error over each child element can be recalculated. Our local sampling method for discontinuous Galerkin discretizations consists of extracting the parent simplex along with its immediate neighbours (opposite each facet). One by one, each edge of the parent simplex is then divided at its midpoint and the solution is recalculated within the local split mesh. During this solution process, Dirichlet boundary conditions are applied to the DOF of the neighbouring simplices such that only the solution of the split elements of the original parent are recomputed. The implementation details for this local split procedure are provided in Appendix C. The edge split configurations used for $n = 2d$ are shown in Figure 2.8.

The implied metric associated with each split configuration is taken as the log-Euclidean mean (Equation 2.15) of the implied metrics of the split elements and the error introduced by the split elements is summed to complete the set of n_s metric-error pairs:

$$\{\mathbf{m}_i, \eta_{\boldsymbol{\kappa}_i}\}. \tag{2.41}$$



(a) Original

(b) Split 1

(c) Split 2

(d) Split 3

Figure 2.8: Split configurations used to sample the error over a triangle for a $p = 3$ discontinuous Galerkin discretization.

Error model synthesis

The metric-error pairs are then used to construct a model of the error as a function of the metric field. This model is constructed in terms of the *step matrix*, $\mathbf{s}$, which represents the difference between two metric tensors. In the context of the discrete set of sampled data, these tensors are computed from $\mathbf{m}_{\boldsymbol{\kappa}_i}$ and $\mathbf{m}_{\boldsymbol{\kappa}_0}$ as

$$\mathbf{s}_{\boldsymbol{\kappa}_i} = \log\left(\mathbf{m}_{\boldsymbol{\kappa}_0}^{-1/2}\mathbf{m}_{\boldsymbol{\kappa}_i}\mathbf{m}_{\boldsymbol{\kappa}_0}^{-1/2}\right), \quad i = 1, \ldots, n_s. \tag{2.42}$$

A linear model is then constructed from the n_s metric-error samples,

$$f_{\boldsymbol{\kappa}}(\mathbf{s}) \equiv \log\left(\eta(\mathbf{s})/\eta_0\right) = \mathbf{r}_{\boldsymbol{\kappa}} : \mathbf{s}, \tag{2.43}$$

where $\mathbf{r}_{\boldsymbol{\kappa}}$ is referred to as the *rate matrix*. A linear regression of the sampled data is used to determine $\mathbf{r}_{\boldsymbol{\kappa}}$.

Over an element $\boldsymbol{\kappa}$, the resulting local error model can finally be written in terms of the average element step matrix $\mathbf{s}_{\boldsymbol{\kappa}}$ as

$$\eta_{\boldsymbol{\kappa}}(\mathbf{s}_{\boldsymbol{\kappa}}) = \eta_{\boldsymbol{\kappa},0}\exp(\mathrm{tr}(\mathbf{r}_{\boldsymbol{\kappa}}\mathbf{s}_{\boldsymbol{\kappa}})) + \frac{p}{n}\left(||\widetilde{\mathbf{s}}_{\boldsymbol{\kappa}}||_F^2 + \sum_{e\in\mathcal{E}(\kappa)}||\widetilde{\mathbf{s}}_{e_1} - \widetilde{\mathbf{s}}_{e_2}||_F^2\right) \tag{2.44}$$

where $\widetilde{\mathbf{s}}_{(\cdot)}$ denotes the trace-free portion of the step matrix and $||\cdot||_F$ is the Frobenius norm. The Frobenius norm penalties in Equation 2.44 are introduced to control the trace-free portion of the step matrix.

In general, we optimize the vertex-valued step matrices, therefore, the average step matrix over the element $\mathbf{s}_\kappa$ is computed from

$$\mathbf{s}_\kappa = \frac{1}{|\mathcal{V}(\kappa)|}\sum_{\nu\in\mathcal{V}(\kappa)}\mathbf{s}_\nu$$

The error in the mesh is then the sum of the local element contributions.

Metric optimization

The last step in an adaptation iteration is to optimize the Riemannian metric field that is passed to a metric-conforming meshing algorithm. Instead of optimizing the metric field, Yano suggests optimizing the vertex-valued field of step matrices $\{\mathbf{s}_v\}_{v\in\mathcal{V}}$. The optimization of the step matrices is performed using the gradient-based approach of Kudo[72].

72. Kudo, *Robust Adaptive High-Order RANS Methods*. 2014

An important property of this optimization is the lower and upper bounds placed on the step matrix entries. Since the local elemental error model is only assumed reliable for metrics that are similar to the current metric, the entries of these step matrices are individually

bounded by $2\log(h_{\text{ref}})$; we refer to h_{ref} as the *refinement factor*. This places bounds on the entries of the step matrices as

$$|\mathbf{s}_{i,j}| \leq 2\log h_{\text{ref}}. \tag{2.45}$$

This refinement factor controls how large the mesh is allowed to grow (or shrink) from one adaptation iteration to the next. Since the error model is constructed from local edge splits, we often set $h_{\text{ref}} = 2$. Lower values of the refinement factor will result in less aggressive adaptation iterations at the expense of slower convergence to the optimal mesh for a given cost.

2.6 Summary

This chapter introduced the fundamentals of meshing, numerical discretizations and the mechanics used to perform PDE-driven mesh adaptation. We are now ready to describe our algorithm for adapting four-dimensional meshes.

Chapter 3

Four-dimensional anisotropic mesh adaptation

It's all about the geometry.

— *Bob*

◁ **Length, volume and quality:**

In this chapter, any mention of geometric quantities such as *length*, *volume* or *quality* should be understood as being measured under the input metric field unless stated otherwise.

3.1 Background

This chapter develops a mesh adaptation algorithm which takes advantage of the expected limits on the incoming metric edge lengths. Furthermore, the mesh adaptation procedure is treated in a dimension-independent manner. This differs from existing software implementations such as `feflo.a`[56], `EPIC`[50], `refine`[51], `gamanic3d`[73], `pragmatic`[74], `omega_h`[75] and `MMG`[76] which are specialized for either $2d$ or $3d$ mesh adaptation. The closest software implementation for dimension-independent mesh adaptation is `MTC` (*Mailleur Topologique en C*) by Thierry Coupez[55] and his student Cyril Gruau[57]. Though dimension-independent in theory, Gruau's demonstrations focus on three-dimensional applications and some very simple four-dimensional metric-conforming cases. In fact, these four-dimensional demonstrations were restricted to the special case of the Euclidean metric. Here, we strive to demonstrate four-dimensional metric-conforming mesh adaptation with more general metrics fields.

Local mesh modification operators are traditionally viewed explicitly as *edge splits*, *edge collapses*, *edge swaps*, *face swaps*, *vertex smoothing*, etc. In fact, these mesh modification operators are all special cases of the more general cavity operators, whereby an existing set of elements is removed and a new set of elements is inserted. As such,

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

50. Michal *et al.*, *Anisotropic Mesh Adaptation through Edge Primitive Operations*. 2012

51. Park *et al.*, *Parallel Anisotropic Tetrahedral Adaptation*. 2008

73. George, *Gamanic3d, Adaptive Anisotropic Tetrahedral Mesh Generator*. 2002

74. Rokos *et al.*, *A Thread-Parallel Algorithm for Anisotropic Mesh Adaptation*. 2013

75. Ibanez, *Conformal Mesh Adaptation on Heterogeneous Supercomputers*. 2016

76. Dobrzynski, *MMG3D: User Guide*. 2012

55. Coupez, *Génération de Maillage et Adaptation de Maillage par Optimisation Locale*. 2000

57. Gruau, *Metric Generation for Anisotropic Mesh Adaptation with Numerical Applications to Material Forming Simulation*. 2005

our dimension-independent method is a fusion of the *star operator* of Coupez[55] with the *cavity operator* of Loseille[56].

Furthermore, it is unclear in the literature how to best schedule local operators to achieve a metric-conforming mesh. Each of the aforementioned software implementations suggest a schedule but provide little empirical justification for the selection. Loseille's[56] recent work provides a general overview of the mesh adaptation schedule and makes an important suggestion that point insertions (via edge splits) should be done so long as no short edges are created in the process. Other work in local operator scheduling is that of Klingner and Shewchuk[77] whereby mesh operators are compounded to achieve higher quality tetrahedra by disregarding the overall computational cost of the algorithm. Here, we provide the details of our mesh adaptation schedule and perform some experiments to highlight important features of our algorithm.

77. Klingner *et al.*, *Aggressive Tetrahedral Mesh Improvement*. 2007

The next sections develop and justify the design and implementation of the dimension-independent local operators. Next, we highlight the geometry hierarchy checks which are necessary to ensure the validity of the dimension-independent operators. We then discuss the scheduling of the local operator and justify the chosen schedule of our mesh adaptation algorithm. Finally, we demonstrate the developed mesh adaptation algorithm on benchmark three- and four-dimensional cases.

◁ **Four-dimensional domains:**

The hierarchical description of the domain geometry is a necessary input to our algorithm. The interested reader is urged to study Appendix A to understand how the geometry hierarchy of our four-dimensional domains (here, the unit tesseract) are constructed.

3.2 *Dimension-independent local operators*

Given an initial n-simplicial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$ of some domain $\Omega \subset \mathbb{R}^n$, a local operation on $\mathcal{M}$ consists of the transformation of its topology $\mathcal{T}$:

$$\mathcal{T}^{k+1} = \mathcal{T}^k \setminus \mathcal{C}^k(f) \cup \mathcal{B}^k(p, \partial\mathcal{C}^k) \tag{3.1}$$

where the superscripts represent the sequence of meshes at each application of the cavity removal, $\mathcal{C}^k(f)$, and the insertion, $\mathcal{B}^k(p, \partial\mathcal{C}^k)$. $\mathcal{C}^k(f)$ denotes the set of *cavity* elements about a j-dimensional facet $f \subset \mathcal{T}^k$ which might be enlarged to ensure topological and geometric validity. This set of cavity elements can be written as

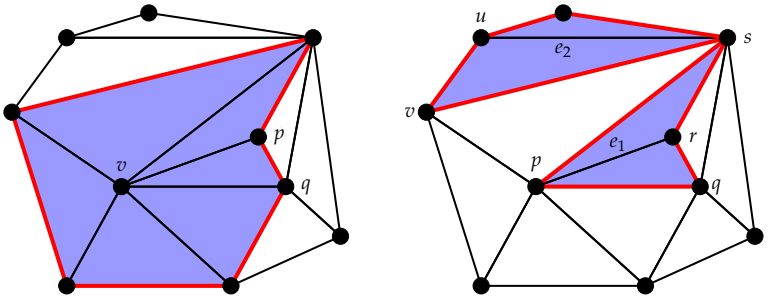
$$\mathcal{C}^k(f) = \left\{ \kappa \,|\, f \subset \kappa, \forall \kappa \in \mathcal{T}^k \right\}. \tag{3.2}$$

Often f is chosen from the set vertices (as integers) or edges of $\mathcal{T}^k$, as in the work of Gruau and Coupez[78]. For a vertex v and edges e_1 and e_2, the corresponding cavities $\mathcal{C}(v)$, $\mathcal{C}(e_1)$ and $\mathcal{C}(e_2)$ are shown in the blue triangles of Figure 3.1. The boundary of the cavity, $\partial\mathcal{C}^k(f)$, is outlined in red. Note that two possible cavities are shown in Figure 3.1(b) about edges e_1 and e_2.

◁ **Topological operation:**

It is important to view Equation 3.1 as an operation on the *topology* of the mesh. The full operation on the mesh $\mathcal{M}$ consists of this topological operation as well as a (possible) modification of its vertices $\mathcal{V}$ such as a vertex insertion, removal or modification of the coordinates.

78. Gruau *et al.*, *3D Tetrahedral, Unstructured and Anisotropic Mesh Generation with Adaptation to Natural and Multidomain Metric*. 2005



(a) Cavity about vertex v.

(b) Cavities about edges e_1 and e_2.

Figure 3.1: Identification of a set of cavity elements. Elements in blue form the cavities $\mathcal{C}(v)$, $\mathcal{C}(e_1)$ and $\mathcal{C}(e_2)$. The boundaries of each cavity are outlined in red.

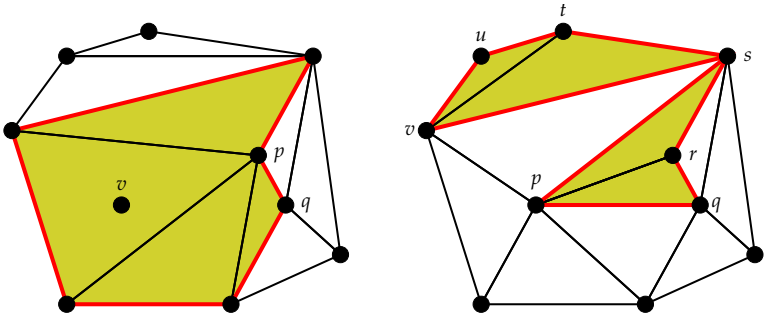
The insertion operator, $\mathcal{B}^k(p, \partial\mathcal{C}^k)$, is obtained by connecting some (possibly new) vertex p with the set of $(n-1)$-dimensional facets in $\partial\mathcal{C}^k$ which do not contain p. Mathematically,

$$\mathcal{B}^k(p, \partial\mathcal{C}^k) = \left\{ \kappa \,\middle|\, \kappa = \{p\} \cup g,\, g \in \partial\mathcal{C}^k,\, p \notin g \right\}. \tag{3.3}$$

where g is a $(n-1)$-dimensional facet on the boundary $\partial\mathcal{C}^k$. The *re-insertion vertex* p is often chosen from the set of vertices in the cavity (though Loseille allows it to be outside the cavity).

◁ **Re-insertion vertex:**

When we refer to the re-insertion vertex p, it should be clear that this is an integer identifier into the vertices $\mathcal{V}$. The coordinates of p are $\mathcal{V}(p)$.



(a) Valid insertion for the cavity of Figure 3.1(a).

(b) Valid insertions for the cavities of Figure 3.1(b).

Figure 3.2: Selection of the re-insertion vertex to produce valid meshes. The set of insertion elements is shown in yellow which is obtained by selecting a vertex and connecting it to the boundary of the set of cavity elements, shown in red. After extracting the set of cavity elements $\mathcal{C}(e_1)$, possible re-insertion candidates include vertices p, q, r and s. However, candidates q and s create invalid meshes. Also observe that selecting either p or r as the re-insertion vertex maintains the original configuration of Figure 3.1(a). In comparison, any of the re-insertion vertices s, t, u or v produce valid meshes when connected to the boundary of the cavity $\mathcal{C}(e_2)$.

Superscripts will now be dropped for brevity. Following the example of Figure 3.1, possible insertions are shown in yellow of Figure 3.2. Note that in Figure 3.2(a), the vertex p was selected as the re-insertion vertex because the re-insertion vertex q would create an invalid mesh. To see this, observe that not every facet in the boundary is *visible* to q. As a result, Loseille proposes to iteratively enlarge the cavity until q *is* visible to the boundary of the cavity. The set of re-insertion elements

should satisfy a *volume* criterion:

$$v(\underbrace{\text{conv}(\mathcal{V}(\overbrace{\{p\} \cup g}^{\kappa}))}_{\boldsymbol{\kappa}}) > 0, \forall g \in \partial\mathcal{C},\ p \notin g. \tag{3.4}$$

This volume can be computed with exact geometric predicates, which we describe in Appendix C.

Similarly, Coupez enlarges an initial set of cavity elements $\mathcal{C}(f)$ to include those simplices in $\mathcal{T}$ which are in the *closure* of the vertices of the original cavity $\mathcal{N}(\mathcal{C}(f))$, where $\mathcal{N}(\cdot)$ retrieves the set of vertices (as integers) in the provided set of elements. Mathematically, the closure is written as

$$\overline{\mathcal{C}}(f) = \left\{\kappa \,|\, \kappa \subset \mathcal{N}(\mathcal{C}(f))\right\}, \tag{3.5}$$

Combined with his *minimum volume principle*, Coupez initially demonstrated this guarantees the topological validity of the mesh[55]. The initial proof was incomplete and later formalized in the thesis work of Cyril Gruau[57], a student of Coupez. This validity check is elaborated upon in the following sections.

◁ **Formation of a physical element:**

Recall the distinction between the topological element κ and the physical one $\boldsymbol{\kappa}$. Here, we are checking the volume of the physical element, formed from the convex hull of the vertices in $\{p\} \cup g$.

55. Coupez, *Génération de Maillage et Adaptation de Maillage par Optimisation Locale.* 2000

57. Gruau, *Metric Generation for Anisotropic Mesh Adaptation with Numerical Applications to Material Forming Simulation.* 2005

Maintaining a valid mesh in theory

To ensure the insertion $\mathcal{B}(p, \partial\mathcal{C})$ is a valid mesh topology, Gruau provides the following lemma along with a simple proof in Proposition 2.2 of his thesis[57].

Lemma 1. *Let $\mathcal{F}$ be a boundaryless mesh topology ($\partial\mathcal{F} = \emptyset$) with a topological dimension of $n-1$. The insertion $\mathcal{B}(p, \mathcal{F})$ is also a mesh topology (with topological dimension n) for any vertex p and the boundary of the insertion is $\mathcal{F}$, i.e., $\partial\mathcal{B} = \mathcal{F}$.*

Finally, we need to ensure the application of a local operator produces a valid mesh topology. In fact, this is true provided the two conditions in the following lemma are satisfied.

Lemma 2. *Let $\mathcal{T}$ be a mesh topology and $\mathcal{C}(f)$ be a set of elements to remove from $\mathcal{T}$ about a facet f. Let $\mathcal{B}(p, \partial\mathcal{C})$ be a set of elements to be inserted in the mesh obtained from Equation 3.3 with a vertex p. Furthermore, let $\mathcal{K}$ be the set of n-simplices in $\mathcal{T}$ with exactly n vertices in the vertices of $\mathcal{C}$:*

$$\mathcal{K} = \left\{\kappa \;\middle|\; |\kappa \cap \mathcal{N}(\mathcal{C})| = n, \forall\, \kappa \in \mathcal{T}\right\}. \tag{3.6}$$

$\mathcal{T} \setminus \mathcal{C}(f) \cup \mathcal{B}(p, \partial\mathcal{C})$ is a mesh topology if

1. *$\partial\mathcal{C}$ is a mesh topology and*
2. *the faces of $\partial\mathcal{C}$ are the faces of $\mathcal{K}$ (with vertices in $\mathcal{N}(\mathcal{C})$).*

The complete proof can be found in Gruau's thesis[57]. Roughly speaking, Condition 2 means that the boundary of the cavity, as seen from the mesh remaining after cavity removal, matches the boundary of the inserted elements. Think of $\mathcal{K}$ as the set of elements on the *other side* of the cavity, see Figure 3.3.

Gruau's proof is quite complicated, with the clearest version written in French. Though we do not claim the following as our own work, we sketch the proof below for two reasons. First, it provides a useful translated reference of the original work of Gruau. Second, it highlights an important feature that can be used to check for validity in practice.

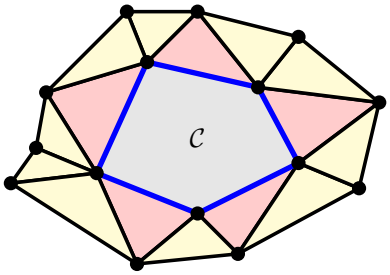


Figure 3.3: Terminology of Lemma 2. The cavity $\mathcal{C}$ is shown in grey with the cavity boundary in blue. The triangles in light red are in $\mathcal{K}$. The yellow triangles are in the remainder of the mesh and are not in $\mathcal{K}$.

◁ **Warning!**

Heavy material ahead! The most important thing you need to remember from the discussion is that the mesh is valid if every element is unique. Click to skip.

Proof. Sketch of Gruau's proof that a local operation maintains a valid mesh. To show that the operation of modifying $\mathcal{T}$ by removing the cavity $\mathcal{C}(f)$ and inserting $\mathcal{B}(p, \partial\mathcal{C})$ maintains a mesh topology, we need to satisfy both properties of a mesh topology (Definition 1). Before doing so, let us modify some notation for brevity and clarity: the removed cavity will now be represented as $\mathcal{C}$ and the insertion will be denoted as $\mathcal{B}$.

The first part of Definition 1 requires that every face of $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$ is counted no more than twice.

Assume there exists some face F in $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$ which is counted more than twice. Now, $\mathcal{T} \setminus \mathcal{C}$ is clearly a mesh topology since $\mathcal{T}$ is assumed to be a mesh topology and the removal of elements in $\mathcal{C}$ clearly maintains a mesh topology. We also know that $\mathcal{B}$ is a mesh topology by Lemma 1. With $\mathcal{T} \setminus \mathcal{C}$ and $\mathcal{B}$ both being mesh topologies (in which every face is counted no more than twice), then for F to be counted more than twice, it *must* be a face of both.

Now, let κ be an element of $\mathcal{T} \setminus \mathcal{C}$ which has F as a face (think of κ as an element in $\mathcal{T}$ on the *other side* of the cavity). Since $\kappa \notin \mathcal{C}$ and F must be a subset of $\mathcal{N}(\mathcal{K})$, then $\kappa \in \mathcal{K}$, meaning $F \in \partial\mathcal{C}$.

Since we know $F \in \partial\mathcal{C}$, then either $F \in \partial\mathcal{T}$ **or** $F \in \partial(\mathcal{T}\setminus\mathcal{C})$. But F always touches at most one element of $\mathcal{T} \setminus \mathcal{C}$ since it is a mesh topology! Furthermore, Lemma 1 states that $\partial\mathcal{B} = \partial\mathcal{C}$ so F also touches at most one element of $\mathcal{B}$. Thus F cannot touch more than two simplices. This contradiction means all faces of $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$ are counted no more than twice.

The next thing to prove is that $\partial\mathcal{T} = \partial(\mathcal{T}\setminus\mathcal{C} \cup \mathcal{B})$. Gruau breaks this down into proving that $\partial\mathcal{T} \subset \partial(\mathcal{T}\setminus\mathcal{C} \cup \mathcal{B}) \wedge \partial(\mathcal{T}\setminus\mathcal{C} \cup \mathcal{B}) \subset \partial\mathcal{T}$.

To prove the first side, consider a face F of the boundary of the mesh $\partial\mathcal{T}$. Since $\mathcal{T}$ is a mesh topology and F is on the boundary, then F touches a single element κ of $\mathcal{T}$. By considering the two cases (1) $\kappa \notin \mathcal{C}$ and (2) $\kappa \in \mathcal{C}$, Gruau demonstrates that for (1) κ is a unique element of $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$ and for (2) that since κ is unique in $\mathcal{C}$, then F is

a face of some other element κ' which is unique in $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$. This means F is in both $\partial\mathcal{T}$ and $\partial\mathcal{T}\backslash\mathcal{C} \cup \mathcal{B}$.

To prove the second side, Gruau now considers a face F of the boundary of the mesh after the operation, $F \in \partial(\mathcal{T}\backslash\mathcal{C} \cup \mathcal{B})$. Again, by considering a unique element $\kappa \in \mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$, there are two cases to consider: (1) $\kappa \notin \mathcal{B}$ and (2) $\kappa \in \mathcal{B}$. If κ is not in $\mathcal{B}$ (1), then it must be in $\mathcal{T} \setminus \mathcal{C}$. This means $\kappa \in \mathcal{T}$ and hence F is on the boundary of $\partial\mathcal{T}$. Now, if $\kappa \in \mathcal{B}$ (2) and κ being unique, then F is on the boundary of $\mathcal{B}$. Since the boundary of $\mathcal{B}$ is the boundary of $\mathcal{C}$ by Lemma 1, then there must also be a unique element of $\mathcal{C}$ with F as a face, meaning this unique element is in $\mathcal{T}$ (since $\mathcal{C} \subset \mathcal{T}$). Thus the boundaries match. □

Again, the reason the proof was presented above is because it highlights an important component that we can use, in practice, to check for validity. Notably, that every element created during the local operation is *unique*. This will be elaborated upon in the next section.

Obviously, we do not want $\partial\mathcal{T} = \partial\left(\mathcal{T}\backslash\mathcal{C} \cup \mathcal{B}\right)$ since we would like to modify the topology of the geometry discretization. In practice, however, we can close the mesh such that $\partial\mathcal{T} = \emptyset$ and thus ensure that the resulting mesh from the local operation retains an empty boundary.

Maintaining a valid mesh in practice

The first thing we do when checking for the validity of a local operator is to check whether the proposed re-insertion vertex is visible to the boundary of the removed cavity. This allows us to geometrically filter out operators that would create invalid meshes.

To topologically guarantee the mesh is valid upon application of each local operator we *close* the mesh such that it is without boundary, similar to the work of Coupez and Gruau. To achieve a mesh without boundary, a ghost (fictitious) vertex is created for each connected boundary. This vertex is then connected to the boundary facets of the original mesh and the resulting (ghost) simplices are appended to this incoming mesh. Without loss of generality, consider a mesh with a single connected boundary, such as that of a n-cube with no interior holes. Denote the ghost vertex as v_0 and the incoming mesh as $\mathcal{T}_0$. The closed mesh is obtained by defining a set of ghost simplices:

$$\mathcal{T}_g = \left\{\{v_0\} \cup g \,|\, \forall g \in \partial\mathcal{T}_0\right\}. \tag{3.7}$$

The full topology of the mesh now becomes the union of the simplices in the incoming mesh with the ghost simplices:

$$\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_g. \tag{3.8}$$

By working with a closed mesh, we ensure that $\partial\mathcal{T} = \emptyset$ and should remain so upon application of each local operation. If the mesh were not

closed, then $\partial\mathcal{T} \neq \emptyset$ and we would have no rigorous way of knowing whether the mesh, after the application of a local operation, is valid.

Furthermore, as remarked by Gruau, Condition 2 of Lemma 2 may not be satisfied, even for valid mesh topologies. This is quite restrictive and suggests this condition should not be directly used to check for mesh validity. Instead, observe that an important byproduct of Gruau's proof (outlined above) is that every element in the resulting mesh topology is *unique*. As a result, we check that every element created by the insertion is unique in $\mathcal{T}$. Directly checking that every inserted element is unique through an exhaustive search of $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$ would be too costly to be used in practice. Instead, we propose the following and only check a subset of the mesh.

Proposition 1. *Let $\mathcal{H}$ denote the set all elements in $\mathcal{T}$ with the proposed re-insertion vertex p as a vertex. That is $\mathcal{H} = \{\kappa \mid p \subset \kappa, \forall \kappa \in \mathcal{T}\}$. Now, let the remainder $\mathcal{R}$ denote the set of all members of $\mathcal{H}$ that are not in the proposed cavity $\mathcal{C}$: $\mathcal{R} = \{ \kappa \mid \kappa \in \mathcal{H} \wedge \kappa \notin \mathcal{C} \}$. See Figure 3.4 for an illustration of this terminology. If*

$$\text{card}(\kappa) = 1, \ \forall \kappa \in (\mathcal{B}(p, \partial\mathcal{C}) \cup \mathcal{R}), \tag{3.9}$$

then $\text{card}(\kappa) = 1, \forall \kappa \in \mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$. That is, every inserted element is unique if it does not appear in the set of elements in the ball of p leftover after the cavity removal.

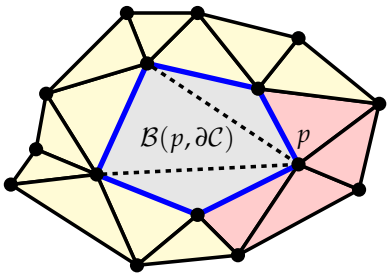


Figure 3.4: Terminology of Proposition 1. The set of inserted elements in $\mathcal{B}(p, \partial\mathcal{C})$ do not already exist in the remainder $\mathcal{R}$ which is described by the set of red triangles attached to vertex p. This may seem impossible in $2d$ but it can occur in higher dimensions.

Proof. Assuming every element of the original mesh $\mathcal{T}$ is unique before the operation, then $\mathcal{T} \setminus \mathcal{C}$ also has unique elements, as does $\mathcal{T} \setminus (\mathcal{C} \cup \mathcal{R})$ (since $\mathcal{C} \subset \mathcal{T}$ and $\mathcal{R} \subset \mathcal{T}$). Suppose the proposition is false, then $\exists\, \kappa^*$ with $\text{card}(\kappa^*) > 1$ in $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$. However, observe that any element $\kappa' \in \mathcal{B} \cup \mathcal{R}$ must contain the re-insertion vertex p. Furthermore, no element in $\mathcal{T} \setminus (\mathcal{C} \cup \mathcal{R})$ can contain the vertex p. Thus if κ^* is single-counted in $\mathcal{B} \cup \mathcal{R}$, then it cannot appear in $\mathcal{T} \setminus (\mathcal{C} \cup \mathcal{R})$ since it would need to contain the vertex p. We thus have a contradiction and κ^* must also be unique in $\mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$. □

Conversely, given a valid mesh topology without boundary, then every element is unique since every face must be counted exactly twice.

To ensure our implementation matches the aforementioned theory, we employ a data structure that tracks the simplex-to-simplex neighbour relations. Observe that because the mesh is closed, then every n-simplex is adjacent to $n+1$ simplex neighbours. Upon application of the cavity operator, the faces on the boundary of the cavity, as seen from the remaining mesh, are first cached. Next, the boundary of the insertion operator is computed and the neighbour-relations are updated by progressively removing the facet in the cache that exists in the boundary of the insertion operator. After the operator terminates, the cache should be empty, which we strictly enforce.

Recovery of common mesh modification operators

The advantage of employing the cavity-insertion framework is that, with the appropriate selection of the cavities and re-insertion vertices, all other mesh operators can be recovered. In fact, Figures 3.1(a) and Figures 3.2(a) are the sequence of steps taken to perform an edge collapse (or vertex removal). Similarly, the sequence of steps in Figures 3.1(b) and Figures 3.2(b) show two possible edge swaps. Table 3.1 gives the cavities and re-insertion vertices which recover common mesh modification operators.

Operator	j	Facet f	Vertex $p(\mathbf{x})$
collapse	1	vertex v_0	$v_1(\mathbf{x}_1)$
split	1	edge $e = (v_0, v_1)$	$v_s(\mathbf{x}_s)$
edge swap	1	edge $e = (v_0, v_1)$	$p(\mathbf{x}_p) \in \mathcal{N}(\mathcal{C})$
facet swap	$d-1$	simplex f	$p(\mathbf{x}_p) \in \mathcal{N}(\mathcal{C})$
smooth	0	vertex p	$p(\tilde{\mathbf{x}}_p)$

Table 3.1: Choice of re-insertion vertices (with associated coordinates) for local operators. The cavities are obtained using Equation 3.2 about the j-dimensional facet f listed in the second column.

In addition to its mathematical rigour, the cavity framework further benefits from a simple software implementation. The latter advantage, however, is lost if cavities are allowed to enlarge. Instead, when edge collapses or splits are rejected, we use swaps to weave out of restrictive geometric and topological configurations which may be causing the rejection. In general, this may not guarantee the operator is ultimately applied but it increases the chance of generating a higher quality mesh. We elaborate on this concept when discussing the schedule of the local operators.

The coordinates for the re-insertion vertices can also be modified during the local operation. For collapses and swaps, the vertex coordinates are not modified and simply fixed at the coordinates of the selected re-insertion vertex. For edge splits, the re-insertion vertex inherits the coordinates of the midpoint of the edge upon which the split occurs ($\mathbf{x}_s$ in Table 3.1). For vertex smoothing, the new coordinates are computed from the local edge lengths surrounding the vertex:

$$\tilde{\mathbf{x}}_p = \mathbf{x}_p + \omega \sum_{e \in \bar{\mathcal{E}}(p)} \left(1 - \ell^4_{\mathbf{m}}(e)\right) \exp(-\ell^4_{\mathbf{m}(e)})\mathbf{e} \tag{3.10}$$

where $\bar{\mathcal{E}}(p)$ is a selected set of the edges that surround the vertex p and $\mathbf{e}$ is the unit vector along that edge. The relaxation factor is selected as $\omega = 0.2$, similar to the inspiring work of Bossen and Heckbert[30]. For interior vertices, $\bar{\mathcal{E}}$ is the full set of edges connected to a vertex. However, for vertices on geometric entities, $\bar{\mathcal{E}}$ is a subset of these edges. This subset is defined from the vertex-to-geometry metadata which is described below.

30. Bossen *et al.*, *A Pliant Method for Anisotropic Mesh Generation*. 1996

3.3 *The importance of the geometry metadata*

It is critical to check whether the proposed operator violates the discretization of the geometry. To perform this check, we strictly enforce that all vertices are tagged with geometry metadata. This *must* be the *lowest-dimensional* geometry entity. For example, a vertex on a geometry Edge is also on Faces (and Volumes for a tesseract geometry) but the associated geometry for this vertex must be the Edge. For interior vertices, this geometry entity is empty ($\emptyset$).

◁ **Warning!**

The days of meshing without a geometry are over!

A common operation in the mesh adaptation algorithm is to determine which geometry entity a facet lies on. For a j-dimensional facet $f \subset \mathcal{T}$, denote the geometric entities of the vertices of f as $\{g_v\}_{v \in f}$. Now, define the set of parents of a geometry entity g as the set of all geometry entities higher in the geometry hierarchy $\mathcal{G}$:

$$\mathcal{P}(g) = \left\{ h \;\middle|\; g \preceq h,\ \forall h \in \mathcal{G} \right\}. \tag{3.11}$$

For each vertex of f, we have the set of parents $\{\mathcal{P}(g_v)\}_{v \in f}$. The geometry entity of this facet g_f is computed from the lowest-dimensional member of the intersection of all these parents. Denote all common parents as G_f:

$$G_f = \bigcap_{v \in f} \mathcal{P}(g_v). \tag{3.12}$$

◁ **dim**

The function dim returns the topological dimension of the entity, i.e., zero for a Node, one for an Edge, two for a Face, etc.

The geometry entity on which this facet lies is then

$$g_f = \underset{g \in G_f}{\arg\min}\ \dim(g). \tag{3.13}$$

Of course, g_f can be empty even if $g_v \neq \emptyset, \forall v \in f$. For curved geometries, g_f can be nonempty despite f being an interior facet of the mesh (think about a concave geometry near a mesh boundary). The *closed* mesh is handy in treating this scenario. Every facet on a geometry entity *must* be adjacent to a ghost simplex. That is,

$$\mathcal{C}(f) \in \mathcal{T}_g \quad \text{where } \mathcal{C}(f) \text{ is from Equation 3.2} \tag{3.14}$$

for facets on geometry discretizations.

We are now equipped to check whether a mesh operator violates the geometry discretization. In the following, assume the full geometry hierarchy is denoted, as a partially ordered set, by $\mathcal{G}$.

For an edge collapse with edge $e = (v_0, v_1)$, if $g_e \neq \emptyset$, the removed vertex v_0 must be higher in $\mathcal{G}$ than v_1. That is $g_{v_1} \preceq g_{v_0}$.

When swapping an edge $e = (v_0, v_1)$ with a re-insertion vertex p, the geometry of the re-insertion vertex g_p must be lower than (or equal to) g_e: $g_p \preceq g_e$. Note that if $\dim(g_e) = 1$, then the swap is rejected because we do not want to swap along a geometry Edge.

For straight-sided domains, a split along an edge e always creates a valid insertion, both geometrically and topologically. However, to avoid mistakes in subsequent mesh operations, the created vertex must be tagged with g_e, the geometry entity of the edge being split. When adapting a mesh within a curved domain, the created vertex needs to be placed on g_e and the cavity, $\mathcal{C}(e)$, may need to be enlarged to support this insertion. We note this is the *only* time cavities are allowed to enlarge. This enlargement is done according to the algorithm of Loseille[56], in which the original cavity is iteratively enlarged by stepping into those facets which cannot "see" the inserted vertex.

As hinted in the last section, a subset of the edges connected to a geometry vertex p are used to smooth the vertex coordinates. This subset is taken as the set of all edges such that the geometry attached to the opposite vertex q is lower than (or equal to) the geometry of vertex p: $g_q \preceq g_p$. Thus vertices on straight-sided geometry entities are smoothed along the entity. For curved geometries, this should be performed in the parameter space of the corresponding entity, but that is left for future work. At the current time, the new vertex coordinates are simply placed on g_p through an *inverse evaluation*.

◁ **Vertices on geometry entities:**

A vertex can be placed on a geometry entity by *projecting* the vertex coordinates to the given entity or by performing an *inverse evaluation* whereby the closest point on the geometry entity (relative to the original vertex coordinates) is found through an optimization procedure. This closest point then forms the coordinates of the vertex. Alternatively, local operators can be performed in the parametric space of the geometry entity, thereby obtaining the vertex coordinates by direct evaluation. Here, we use *inverse evaluation*.

3.4 *Scheduling the local operators*

The local operator schedule is inspired by the work of Loseille[56,66]. However, some changes were made in an attempt to improve the metric conformity of the final mesh.

For reference, Loseille proposes to first perform collapses and splits to create a unit mesh, followed by swaps and smoothing to optimize the mesh quality. An important feature of Loseille's work is to ensure that no short edges are created during any edge split operation as this would require another pass of the collapse operator. Building upon this idea, we investigate the idea of interleaving swaps within the collapse and split operators to weave out of restrictive (geometry- or visibility-related) topological configurations.

The operator schedule, listed in Algorithm 3.6, is composed of three main stages. The first stage consists of targeting edges longer than ℓ_0 (typically, $\ell_0 = 2$) in the metric space whereas the second stage targets edge lengths longer than $\sqrt{2}$. We found this necessary in an attempt to avoid overshooting the number of elements in the adaptation process. Performing two iterations of splits and collapses in each stage is motivated by the fact that restrictive topological configurations might cause an operator to be rejected on a first pass, but a global pass of the swap operator may be helpful in weaving out of these configurations before attempting another sub-stage of collapses and splits. In addition, the swap operator has been directly interleaved within the

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

66. Loseille, *Metric-Orthogonal Anisotropic Mesh Generation*. 2014

◁ **Notation reminder:**

In the algorithms that follow, remember that a topological element is denoted by $\kappa \in \mathcal{T}$ whereas a physical one is denoted by $\boldsymbol{\kappa} \in \mathcal{M} = (\mathcal{V}, \mathcal{T})$. We will often interchange between a topological element κ and its associated physical one $\boldsymbol{\kappa}$.

split and collapse operators to immediately weave out of these restrictive configurations. Before a swap is accepted, the inserted topology is checked to ensure the current minimum and maximum edge lengths do not get worse with the application of the swap. This ensures the edge length bounds can only improve during the mesh adaptation procedure.

Passes on the swap operator are divided into two target qualities, first targeting simplices of quality less than 0.4 and next targeting those of quality less than 0.8. Since there will be fewer simplices with quality < 0.4, swaps can be attempted recursively until no change is made to the mesh. However, there may be several simplices with quality < 0.8, therefore, the global loop over edges on Line 2 of Algorithm 3.6 is limited to five iterations. All other sub-stages are performed recursively, i.e., until no further operator in the sub-stage can be performed.

Note that the same swap kernel, described in Algorithm 3.3 is used by the passes of collapses and splits when these operators are rejected (Algorithms 3.5 and 3.4, respectively) as well as the global swap pass (Algorithm 3.2). The `swapout` parameter, which is an input to both the split and collapse algorithms, is used to control whether swaps are attempted when these operators are rejected.

Collapses always target edges which are shorter than $\sqrt{2}/2$, however, splits more generally target edges that are longer than some target length ℓ_t. As will be seen in the next section, the ability to parametrize splits in terms of a target length is desirable in order to control the number of simplices generated by the adaptation algorithm. Since edge splits are always performed after a call to the edge collapse algorithm, edges on geometry entities are prioritized in each edge split pass. The reason for this decision is because the mesh will automatically be at its coarsest following the collapse algorithm, thus edge splits requiring a geometry projection will be more likely to be accepted without cavity enlargement.

The split operator accepts an additional parameter, f_{dof}, which is referred to as the degree-of-freedom (DOF) control factor. This parameter was introduced, particularly for $4d$ applications because upon the insertion of a single vertex, the number of pentatopes grows quite large. In $2d$ the total number of triangles is increased by 2 whereas in $3d$, assuming the ball of a vertex is roughly the shape of an icosahedron[79,80], the total number of tetrahedra increases by six or seven. In $4d$, we have experimentally observed the number of pentatopes attached to an edge can be on the order of 15-20, which means 15-20 new pentatopes are created upon the insertion of a single vertex. As a result, the DOF-insertion factor, f_{dof} is introduced to control the metric volume of the inserted simplices. For the inserted cavity $\mathcal{B}$, we require $|\mathcal{B}| \approx v_{\mathbf{m}}(\mathcal{B})/v_{\Delta}$. That is, the number of simplices expected

◁ **Remember:**

Cavities are only allowed to enlarge during splits along geometry entities. But if this enlargement causes vertices to be deleted, the split is rejected.

79. Caplan, *An Adaptive Framework for High-Order, Mixed-Element Numerical Simulations.* 2014

80. Huerta *et al.*, *Efficiency of High-Order Elements for Continuous and Discontinuous Galerkin Methods.* 2013

◁ **Increase in DOF upon insertion:**

For a discontinuous Galerkin discretization of order p, each pentatope accounts for $(p+1)(p+2)(p+3)(p+4)/24$ DOF in the mesh. Even for a linear discretization ($p = 1$), the number of DOF is increased by ≈ 100 (assuming 20 new pentatopes are created) which means we need to be very certain about a decision to insert a vertex.

by the metric volume should be approximately equal to the number of simplices inserted. In practice, this condition is relaxed by allowing $|\mathcal{B}| < f_{\text{dof}} v_{\mathbf{m}}(\mathcal{B})/v_{\Delta}$ so as to not be too restrictive with insertions. We do not impose this criterion in $2d$ and $3d$ since the number of inserted simplices is not as high as it is in $4d$, in which we set $f_{\text{dof}} = \sqrt{2}$. In the next section, we will examine how this factor impacts metric conformity as well as the number of simplices in the mesh.

The local vertex smoothing procedure is listed in Algorithm 3.1. For a vertex p, the set of neighbouring vertices lower in the geometry hierarchy than p are determined and used to compute the new coordinates of p using Equation 3.10 which tends to drive the edge lengths surrounding p to unity. Though we observe good results with this method, an alternative may be to use a quality-based objective for vertex smoothing.

smoothVertices

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}$
output: $\mathcal{M}$
1 **for** $p \in \mathcal{V}$ ▷ all vertices of the mesh
2 $\mathcal{C} \leftarrow \mathcal{C}(p)$ from Equation 3.2
3 $\overline{\mathcal{E}} \leftarrow \{e = (p, q) \mid g_q \preceq g_p,\ \forall q \in \mathcal{N}(\mathcal{C})\}$
4 $\widetilde{\mathbf{x}}_p \leftarrow$ **computeCoordinates**$(\mathbf{x}_p, \mathbf{m}, \overline{\mathcal{E}})$ (Equation 3.10)
5 **if** $g_p \neq \emptyset$
6 $\widetilde{\mathbf{x}}_p \leftarrow$ **placeOnGeometry**$(g_p, \widetilde{\mathbf{x}}_p)$
7 **if** $\exists\ \kappa \in \mathcal{C}$ with $v(\boldsymbol{\kappa}) < 0$ ▷ with coordinates $\widetilde{\mathbf{x}}_p$
8 **continue**
9 $\mathbf{x}_p \leftarrow \widetilde{\mathbf{x}}_p$ ▷ accept the coordinates

Algorithm 3.1: Vertex smoothing algorithm. The inputs consist of the initial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, a set of metric tensors at each vertex of the mesh $\mathbf{m}$. The proposed coordinates are computed according to Equation 3.10 with the edges connected to the vertex defined on Line 3. Should the vertex lie on a geometry entity (g_p), it must be placed on the geometry (Line 6). The visibility of the vertex with the proposed coordinates is checked on Line 7. The topology $\mathcal{T}$ is not modified.

swapEdges

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}$, q_t
output: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$
1 **while** $\mathcal{T}$ is modified
2 **for** $e \in \mathcal{E}(\mathcal{T})$
3 $\mathcal{C} \leftarrow \mathcal{C}(e)$ from Equation 3.2
4 $q_{\text{min}} \leftarrow \min(q_{\mathbf{m}}(\kappa)),\ \kappa \in \mathcal{C}$
5 **if** $q_{\text{min}} > q_t$
6 **continue**
7 $\mathcal{M} \leftarrow$ **trySwap**$(\mathcal{M}, \mathbf{m}, e, q_{\text{min}})$

Algorithm 3.2: Edge swap algorithm. The inputs consist of the initial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$ and a set of metric tensors at each vertex of the mesh $\mathbf{m}$ and a target quality to improve, q_t. In contrast to the split and collapse algorithms, swaps attempt to *locally* improve the worst quality of the mesh (Line 4) unless it is already greater than the target q_t (Line 5). Only the topology $\mathcal{T}$ is modified.

trySwap

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}$, e, q_0
output: $\mathcal{M}$
1 $\ell_{\text{min}} \leftarrow \min(\ell_{\mathbf{m}}(e))\ e \in \mathcal{E}(\mathcal{T})$
2 $\ell_{\text{max}} \leftarrow \max(\ell_{\mathbf{m}}(e))\ e \in \mathcal{E}(\mathcal{T})$
3 $q_{\text{min}} \leftarrow q_0$
4 $m \leftarrow \emptyset$ ▷ the candidate re-insertion vertex
5 $\mathcal{C} \leftarrow \mathcal{C}(e)$ from Equation 3.2
6 **for** $p \in \mathcal{N}(\mathcal{C})$ ▷ all the vertices of the cavity
7 **if** $p \in e$
8 **continue**
9 **if** $g_p \not\preceq g_e$
10 **continue**
11 $\mathcal{B} \leftarrow \mathcal{B}(p, \partial\mathcal{C})$ from Equation 3.3
12 **if** $\exists\ \kappa \in \mathcal{B}$ with $v(\boldsymbol{\kappa}) < 0$
13 **continue**
14 $q \leftarrow \min(q_{\mathbf{m}}(\boldsymbol{\kappa}))$, $\boldsymbol{\kappa} \in (\mathcal{V}, \mathcal{B})$
15 **if** $q < q_{\text{min}}$
16 **continue**
17 **if** $\exists\ e \in \mathcal{E}(\mathcal{B})$, with $\ell_{\mathbf{m}}(e) < \ell_{\text{min}}$ or $\ell_{\mathbf{m}}(e) > \ell_{\text{max}}$
18 **continue**
19 **if** $\exists\ \kappa \in \mathcal{B}$ violating Proposition 1
20 **continue**
21 $q_{\text{min}} \leftarrow q$
22 $m \leftarrow p$
23 **if** $m = \emptyset$
24 **return**
25 $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{C}\ \cup\ \mathcal{B}(m, \partial\mathcal{C})$

Algorithm 3.3: Edge swap kernel. The inputs consist of the initial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, a set of metric tensors at each vertex of the mesh $\mathbf{m}$ and the edge e about which swaps will be attempted. The input q_0 refers to the quality that will be used as the benchmark for improvement. Line 9 corresponds to the geometry hierarchy check and Line 12 corresponds to the visibility check of Equation 3.4. If the swap creates edge lengths that are outside the edge length bounds of the current (entire) mesh (computed on Lines 1 and 2), then the swap is rejected (Line 17). The optimal re-insertion vertex m is selected as the one which most improves the incoming worst quality q_0. If no re-insertion vertex is found such that this q_0 is improved, then no swap is performed (Line 23). The operator is applied on Line 25. Note the vertices $\mathcal{V}$ are not modified.

collapseEdges

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}$, `swapout`
output: $\mathcal{M}$

1 $q_{\min} \leftarrow \min(q_{\mathbf{m}}(\boldsymbol{\kappa}))$, $\boldsymbol{\kappa} \in \mathcal{M}$
2 **repeat**
3 $n_c = 0$ ▷ initialize number of collapses
4 $\mathcal{E}_{\text{short}} \leftarrow \emptyset$ ▷ list of short edges
5 **for** $e \in \mathcal{E}(\mathcal{T})$
6 **if** $\ell_{\mathbf{m}}(e) < \sqrt{2}/2$
7 $\mathcal{E}_{\text{short}} \leftarrow \mathcal{E}_{\text{short}} \cup (v_0, v_1) \cup (v_1, v_0)$
8 $\mathcal{E}_{\text{short}} \leftarrow$ **sort**$(\mathcal{E}_{\text{short}})$
9
10 **for** $e \in \mathcal{E}_{\text{short}}$
11 $\mathcal{C} \leftarrow \mathcal{C}(v_0)$ from Equation 3.2
12 **if** $g_{v_0} \not\preceq g_{v_1}$
13 **if** (`swapout`)
14 $\mathcal{M} \leftarrow$ **trySwap**$(\mathcal{M}, \mathbf{m}, e, q_{\min})$
15 $\mathcal{E}_{\text{short}} \leftarrow$ **update**$(\mathcal{E}_{\text{short}})$
16 **continue**
17 $\mathcal{B} \leftarrow \mathcal{B}(v_1, \partial\mathcal{C})$ from Equation 3.3
18 **if** $\exists\, \kappa \in \mathcal{B}$ with $v(\boldsymbol{\kappa}) < 0$
19 **if** (`swapout`)
20 $\mathcal{M} \leftarrow$ **trySwap**$(\mathcal{M}, \mathbf{m}, e, q_{\min})$
21 $\mathcal{E}_{\text{short}} \leftarrow$ **update**$(\mathcal{E}_{\text{short}})$
22 **continue**
23 **if** $\exists\, \kappa \in \mathcal{B}$ with $q_{\mathbf{m}}(\boldsymbol{\kappa}) < q_{\min}$
24 **continue**
25 **if** $\exists\, \kappa \in \mathcal{B}$ violating Proposition 1
26 **continue**
27 $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$
28 $\mathcal{V} \leftarrow \mathcal{V} \setminus \mathcal{V}(v_0)$
29 $n_c \leftarrow n_c + 1$
30 **until** $n_c = 0$

Algorithm 3.4: Edge collapse algorithm. The inputs consist of the initial mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, a set of metric tensors at each vertex of the mesh $\mathbf{m}$ and a boolean flag, `swapout` corresponding to whether edge swaps will be attempted when collapses are rejected. Furthermore, Line 18 corresponds to the visibility check of Equation 3.4 and Line 23 ensures the worst quality of the mesh does not degrade. If a swap is performed on Line 14 or 20 then the list of long edges in $\mathcal{E}_{\text{short}}$ must be updated. Note that the collapse is considered in both directions (each endpoint of a short edge onto the other) – see Line 7. Both the topology $\mathcal{T}$ and the vertices $\mathcal{V}$ are modified.

splitEdges

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}$, ℓ_t, `swapout`, f_{dof}
output: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$
1 $\ell_{\min} \leftarrow \min(\ell_{\mathbf{m}}(e))$, $e \in \mathcal{E}(\mathcal{T})$
2 **if** $\ell_{\min} < 0.5$
3 $\quad \ell_{\min} = 0.5$
4 $q_{\min} \leftarrow \min(q_{\mathbf{m}}(\kappa))$, $\kappa \in \mathcal{T}$
5 **repeat**
6 $\quad n_s = 0 \triangleright$ initialize number of splits
7 $\quad \mathcal{E} \leftarrow \mathcal{E}(\mathcal{T})$
8 $\quad \mathcal{E}_{\text{long}} \leftarrow \emptyset$
9 $\quad$ **for** $e \in \mathcal{E}(\mathcal{T})$
10 $\quad\quad$ **if** $\ell_{\mathbf{m}}(e) > \ell_t$
11 $\quad\quad\quad \mathcal{E}_{\text{long}} \leftarrow \mathcal{E}_{\text{long}} \cup e$
12 $\quad \mathcal{E}_{\text{long}} \leftarrow$ **sort**$(\mathcal{E}_{\text{long}})$
13
14 $\quad$ **for** $e \in \mathcal{E}_{\text{long}}$
15 $\quad\quad \mathcal{C} \leftarrow \mathcal{C}(e)$ from Equation 3.2
16 $\quad\quad p \leftarrow |\mathcal{V}| + 1$
17 $\quad\quad \mathbf{x}_p \leftarrow$ **midpoint**(e)
18 $\quad\quad \mathcal{B} \leftarrow \mathcal{B}(p, \partial\mathcal{C})$ from Equation 3.3
19 $\quad\quad$ **if** $g_e \neq \emptyset$
20 $\quad\quad\quad \mathbf{x}_p \leftarrow$ **placeOnGeometry**$(g_e, \mathbf{x}_p)$
21 $\quad\quad\quad \mathcal{C} \leftarrow$ **enlarge**$(\mathcal{C}, p)$
22 $\quad\quad$ **if** $\exists\, q \in \mathcal{N}(\mathcal{C})$ with $\ell_{\mathbf{m}}(p, q) < \ell_{\min}$
23 $\quad\quad\quad$ **if** (`swapout`)
24 $\quad\quad\quad\quad \mathcal{M} \leftarrow$ **trySwap**$(\mathcal{M}, \mathbf{m}, e, q_{\min})$
25 $\quad\quad\quad\quad \mathcal{E}_{\text{long}} \leftarrow$ **update**$(\mathcal{E}_{\text{long}})$
26 $\quad\quad\quad$ **continue**
27 $\quad\quad$ **if** $\exists\, \kappa \in \mathcal{B}$ with $q_{\mathbf{m}}(\kappa) < q_{\min}$
28 $\quad\quad\quad$ **continue**
29 $\quad\quad$ **if** $f_{\text{dof}} \cdot v_{\mathbf{m}}(\mathcal{B}) / v_{\Delta} < |\mathcal{B}|$
30 $\quad\quad\quad$ **continue**
31 $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{C} \cup \mathcal{B}$
32 $\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup p$
33 $\quad\quad n_s \leftarrow n_s + 1$
34 **until** $n_s = 0$

Algorithm 3.5: Edge split algorithm. The inputs consist of the initial mesh $\mathcal{M}$, a set of metric tensors at each vertex of the mesh $\mathbf{m}$, a target length for which edges will be split ℓ_t and a boolean flag, `swapout` corresponding to whether edge swaps will be attempted when insertions are rejected due to the creation of short edges (Line 22). This check is relaxed when e is on a geometry Edge. First, the list of edges with length greater than ℓ_t is identified and then sorted according to length; also, edges on the geometry are placed at the beginning of the list. Furthermore, Line 27 ensures the worst quality of the mesh does not degrade. Line 29 restricts insertions by checking the number of inserted elements does not grow too large relative to the metric volume of the inserted elements. An input factor f_{dof} controls this behaviour. Should an edge be on the geometry, then the vertex is placed on the geometry (Line 20). Note that if a swap occurs on Line 24, then the list of long edges $\mathcal{E}_{\text{long}}$ must be updated. Both the vertices $\mathcal{V}$ and the topology $\mathcal{T}$ of the mesh $\mathcal{M}$ are modified.

adaptMesh

input: $\mathcal{M}_{\text{in}} = (\mathcal{V}_{\text{in}}, \mathcal{T}_{\text{in}})$, $\mathbf{m}$, `swapout`, ℓ_0, f_{dof}
output: $\mathcal{M}_{\text{out}}$ (modified)

$\mathcal{T} \leftarrow \mathcal{T}_{\text{in}} \cup \mathcal{T}_g$ ▷ close the mesh

▷ stage 1: target edges longer than ℓ_0
do (twice)
  $\mathcal{M} \leftarrow$ **collapseEdges**$(\mathcal{M}, \mathbf{m}, \texttt{swapout})$
  $\mathcal{M} \leftarrow$ **splitEdges**$(\mathcal{M}, \mathbf{m}, \ell_0, \texttt{swapout}, f_{\text{dof}})$
  $\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.4)$
  $\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.8)$
  $\mathcal{M} \leftarrow$ **smoothVertices**$(\mathcal{M}, \mathbf{m})$

▷ stage 2: target edges longer than $\sqrt{2}$
do (twice)
  $\mathcal{M} \leftarrow$ **collapseEdges**$(\mathcal{M}, \mathbf{m}, \texttt{swapout})$
  $\mathcal{M} \leftarrow$ **splitEdges**$(\mathcal{M}, \mathbf{m}, \sqrt{2}, \texttt{swapout}, f_{\text{dof}})$
  $\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.4)$
  $\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.8)$
  $\mathcal{M} \leftarrow$ **smoothVertices**$(\mathcal{M}, \mathbf{m})$

▷ stage 3: do a last pass of swaps
$\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.4)$
$\mathcal{M} \leftarrow$ **swapEdges**$(\mathcal{M}, \mathbf{m}, 0.8)$

$\mathcal{T}_{\text{out}} \leftarrow \mathcal{T} \setminus \mathcal{T}_g$ ▷ remove ghosts
$\mathcal{M}_{\text{out}} \leftarrow (\mathcal{V}, \mathcal{T}_{\text{out}})$

Algorithm 3.6: Mesh adaptation algorithm. The inputs to the algorithm are the input mesh ($\mathcal{M}_{\text{in}}$), a metric field ($\mathbf{m}$), an option to allow splits and collapses to swap out of restrictive topological configurations (`swapout`), a target edge length for the first pass of edge splits (ℓ_0) and an option to restrict the degrees-of-freedom during edge splits (f_{dof}). When active, this parameter is set to $f_{\text{dof}} = \sqrt{2}$. Observe that swaps are broken into two stages, first targeting elements with a quality less than 0.4, subsequently targeting those with quality less than 0.8.

3.5 Assessment of the mesh adaptation capability

Let us now assess our mesh adaptation adaptation algorithm by studying three-dimensional benchmark cases from the Unstructured Grid Adaptation Working Group[62] as well as some four-dimensional cases. We will further evaluate the utility of the proposed mesh adaptation components (splits, collapses, swaps, smoothing) along with the parameters we propose.

62. Ibanez *et al.*, *First Benchmark of the Unstructured Grid Adaptation Working Group*. 2017

Assessment procedure

The goal, here, is to produce a metric-conforming mesh for some domain of interest Ω. In the following, we begin with some initial mesh and prescribe an analytic target metric, $\mathbf{m}_t$, at the vertices of the initial mesh.

Directly applying the analytic metric at the vertices of the input mesh would generally be inconsistent with our assumption about the metric with respect to the initial mesh. That is, the lengths of the edges of the input mesh may not be within the expected range of $\approx [\frac{1}{2}, 2]$. As a result, we propose a simple metric limiting procedure, described in Algorithm 3.7 to limit the analytic metric so as to emulate the behaviour we expect from our adaptive simulation where metrics are provided by MOESS[3]. As the algorithm converges, the metric should no longer be limited, but should approach the analytic target metric. Michal and Krakos suggest a similar metric limiting procedure[50].

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

50. Michal *et al.*, *Anisotropic Mesh Adaptation through Edge Primitive Operations*. 2012

Each iteration of Algorithm 3.7 starts by computing the vertex-valued implied metric $\mathbf{m}_I$ of the current mesh by optimizing the objective function of Equation 2.23. Next, the step matrices from this vertex-valued implied metric to the analytic target one, $\mathbf{m}_t$, are computed and then limited in the same manner as the MOESS algorithm in Equation 2.45. The refinement factor is set to $h_{\text{ref}} = 2$ which, in general, does not guarantee that every mesh will have a length in the range $[\frac{1}{2}, 2]$ (because of the coupling between the vertex step matrices along each edge) but should be close to being in the range.

Metric conformity is assessed by evaluating the edge lengths and simplex qualities under the original (unlimited) analytic target metric. The percentage of edges ($\%\ell_{\text{unit}}$) within the characteristic range of $[\sqrt{2}/2, \sqrt{2}]$ for a quasi-unit mesh is reported as well as the fraction of simplices with a quality greater than 0.8 ($\%q_{\text{unit}}$).

We further report the total number of simplices and compare that with the analytic expected number of simplices:

$$n_s = \frac{\int_\Omega \sqrt{\det \mathbf{m}_t(\mathbf{x})} \mathrm{d}\mathbf{x}}{v_\Delta}. \tag{3.15}$$

generateMetricConformingMesh

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, $\mathbf{m}_t$
output: $\mathcal{M}$
1 **for** $i = 1$ **to** n_{iter}
2 $\quad \mathbf{m}_I \leftarrow$ **impliedMetric**$(\mathcal{M})$ by minimizing Equation 2.23
3 $\quad \mathbf{m} \leftarrow \mathbf{m}_t(\mathcal{V})$ ▷ evaluate target metric at vertices
4 $\quad$ **for** $v \in \mathcal{V}$
5 $\quad\quad \mathbf{s}_v \leftarrow \log\left(\mathbf{m}_{I,v}^{-1/2}\mathbf{m}_v\mathbf{m}_{I,v}^{-1/2}\right)$
6 $\quad\quad \mathbf{s}_v \leftarrow$ **limit**$(\mathbf{s}_v)$ using Equation 2.45 with $h_{\text{ref}} = 2$
7 $\quad\quad \mathbf{m}_v \leftarrow \exp\left(\mathbf{m}_{I,v}^{-1/2}\mathbf{s}_v\mathbf{m}_{I,v}^{-1/2}\right)$
8 $\quad \mathcal{M} \leftarrow$ **adaptMesh**$(\mathcal{M}, \mathbf{m})$

Algorithm 3.7: Target metric assessment procedure. The analytic metric $\mathbf{m}_t$ is first evaluated at the current mesh vertices (Line 3) and then limited in Line 6 according to the step from the implied metric of the mesh to the target (Line 5). A new mesh is then generated on Line 8 from which metric conformity under the analytic metric can be assessed.

For some metric fields, this may be analytically computed.

Demonstration on three-dimensional benchmark cases

The three-dimensional benchmark cases studied in this section are selected from the first benchmark of the Unstructured Grid Adaptation Working Group[62]. The geometries and descriptions of the metrics can be accessed at the group's repository[81] and are briefly reviewed here.

62. Ibanez *et al.*, *First Benchmark of the Unstructured Grid Adaptation Working Group*. 2017

81. Unstructured Grid Adaptation Working Group, *UGAWG GitHub repository*. 2019

Cube Linear (CL) The Cube Linear (CL) case consists of generating a mesh within a unit cube $\Omega = \Omega_c \in [0,1]^3]$ for the metric $\mathbf{m}_t(x,y,z) = \text{diag}(h_x^{-2}, h_y^{-2}, h_z^{-2})$ with $h_z(z) = h_0 + 2(0.1 - h_0)|z - 0.5|$ ($h_0 = 0.001$) and $h_y = h_z = 0.1$. Evaluating Equation 3.15 yields an expected $n_{\text{CL}} =$ 39k tetrahedra.

Cube-Cylinder Linear (CCL) The Cube-Cylinder Linear (CCL) case exploits the same metric as the Cube Linear case but generates the mesh in a domain represented by the Boolean subtraction of a cylinder Ω_r, centered along the z-axis with radius 0.5, from the unit cube $\Omega_c = [0,1]^3$: $\Omega = \Omega_c - \Omega_r$. The expected number of simplices is computed from the volume fraction of the resulting domain from the unit cube: $n_{\text{CCL}} = (1 - \frac{1}{4}\pi)n_{\text{CL}} \approx 31.7$k tetrahedra.

Cube-Cylinder Polar 1 (CCP1) The Cube-Cylinder Polar 1 (CCP1) case consists of generating a mesh within the same geometry as in the

Cube-Cylinder case but defines an analytic metric

$$\mathbf{m}(\mathbf{x}) = \mathbf{Q}\,\mathrm{diag}(h_r^{-2}, h_t^{-2}, h_z^{-2})\,\mathbf{Q}^t, \quad \text{with } \mathbf{Q} = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.16}$$

where $r = \sqrt{x^2 + y^2}$, $t = \arctan(y, x)$, $h_t = h_z = 0.1$ and $h_r = h_0 + 2(0.1 - h_0)|r - 0.5|$ ($h_0 = 0.001$). The expected number of tetrahedra is 20.2k. As noted by Ibanez, this metric is quite difficult to conform to, mostly because of the large gradation in the mesh requested near the curved surface. Ibanez, in fact, proposes a metric smoothing procedure to make the mesh generation procedure slightly easier for this case which ultimately forms the Cube-Cylinder Polar 2 metric in the following.

Cube-Cylinder Polar 2 (CCP2) To relax the gradation near the curved surface of the Cube-Cylinder geometry, Ibanez[62] proposes to modify the metric of the Cube-Cylinder Polar 1 case by adjusting the spacing in the tangential direction h_t:

62. Ibanez *et al.*, *First Benchmark of the Unstructured Grid Adaptation Working Group*. 2017

$$h_t = \begin{cases} 0.1 & \text{if} \quad d < 0 \\ \frac{d}{40} + 0.1(1 - d) & \text{if} \quad d \geq 0 \end{cases}, \quad \text{with } d = 10(0.6 - r). \tag{3.17}$$

We expect 34.01k tetrahedra for this case.

Results The meshes generated by our algorithm are shown in Figure 3.7 which appear similar to those produced by the UGAWG[62]. A more quantitative assessment is given by the metric conformity statistics of Table 3.2 along with the edge length and tetrahedron quality histograms of Figure 3.5. For all cases except the Cube-Cylinder Polar 1 case, we see excellent metric conformity whereby at least 95% of the edges are within the quasi-unit range. Furthermore, the number of simplices with quality greater than 0.8 is excellent, ranging from 79% to 92% except for the Cube-Cylinder Polar 1 case. Again, we note that this case was deemed difficult because of the high gradation requested by the metric near the curved surface of the cylinder.

An important distinction of our algorithm in contrast to the participants of the UGAWG is that the number of tetrahedra achieved by our algorithm is much closer to the analytic expected number. In particular, observe that we are often within $5 - 6\%$ of the expected number (omitting the Cube-Cylinder Polar 1 case) as shown in the number of simplices of Table 3.2. The convergence of the number of tetrahedra over the course of the 20 adaptation iterations for each case is shown in Figure 3.6 which demonstrates that our algorithm steadily achieves the expected number of tetrahedra (except in the Cube-Cylinder Polar

1 case which roughly exhibits a 15% overshoot). Furthermore, the convergence of the edges in the quasi-unit edge length range (measured under the analytic metric) converges to 95-99%, again, with the exception of the Cube-Cylinder Polar 1 case which steadily converges to approximately 86% metric conformity in the quasi-unit edge lengths.

Of all of the cases, metric-conformity is poorest for our algorithm when applied to the Cube-Cylinder Polar 1 case, which can be observed in both the statistics of Table 3.2 and the wider distributions of Figure 3.5.

Table 3.2 also reports the metric-conformity statistics obtained from the meshes generated by the `feflo.a`[56] and `EPIC-ICSM`[50] softwares. In addition to matching the expected number of tetrahedra more closely, our algorithm compares well with existing technologies. With the exception of the Cube-Cylinder Polar 1 case, the fraction of edges in the quasi-unit range is highest with our algorithm. For all cases, the fraction of simplices in the quasi-unit range is highest with our algorithm.

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes.* 2017

50. Michal *et al.*, *Anisotropic Mesh Adaptation through Edge Primitive Operations.* 2012

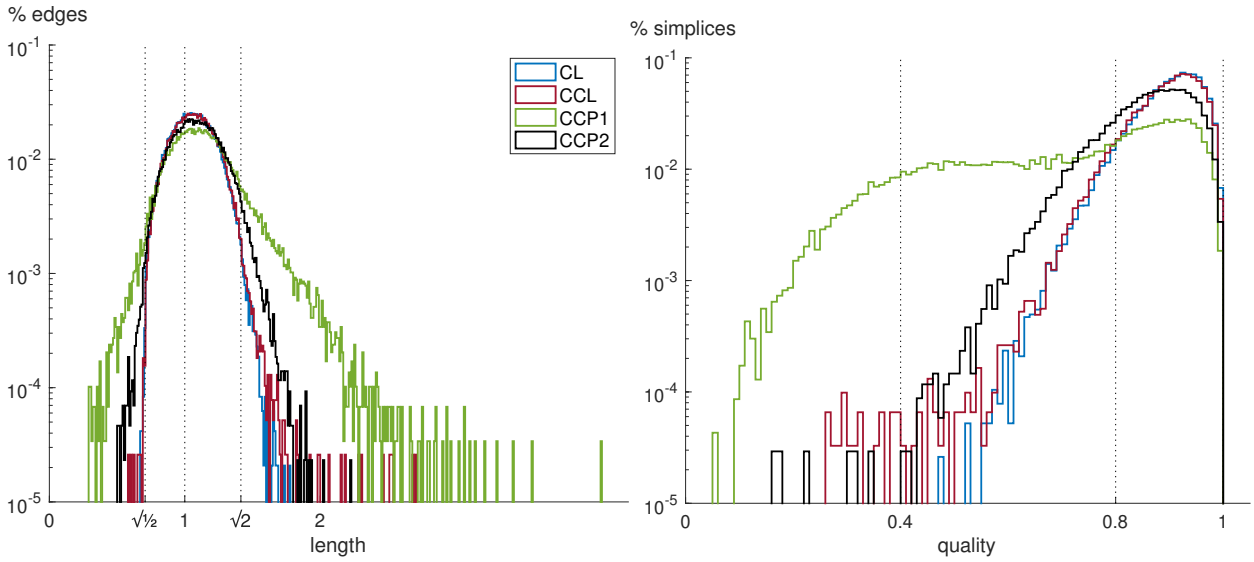
CL (39.0k)	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices
Current	0.67	1.77	1.06	99.10 %	0.47	0.90	92.15 %	38.30k
`feflo.a`	0.45	1.80	1.03	98.28 %	0.49	0.85	74.28 %	45.16k
`EPIC-ICSM`	0.32	1.95	1.03	93.04 %	0.35	0.81	59.52 %	47.55k

CCL (31.7k)	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices
Current	0.59	2.70	1.07	98.79 %	0.27	0.89	90.88 %	30.45k
`feflo.a`	0.21	2.55	0.97	93.73 %	0.04	0.80	55.65 %	46.29k
`EPIC-ICSM`	0.34	2.44	1.03	92.07 %	0.20	0.81	56.71 %	38.30k

CCP1 (20.2k)	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices
Current	0.29	4.07	1.14	85.66 %	0.06	0.70	43.22 %	23.25k
`feflo.a`	0.18	17.40	1.03	89.16 %	0.01	0.68	33.59 %	35.31k
`EPIC-ICSM`	0.16	3.14	1.04	86.19 %	0.10	0.69	35.95 %	30.38k

CCP2 (36.4k)	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices
Current	0.50	2.22	1.09	95.76 %	0.17	0.86	78.76 %	34.20k
`feflo.a`	0.18	2.65	0.98	93.83 %	0.06	0.80	55.92 %	53.12k
`EPIC-ICSM`	0.37	2.30	1.03	91.77 %	0.24	0.81	58.52 %	44.28k

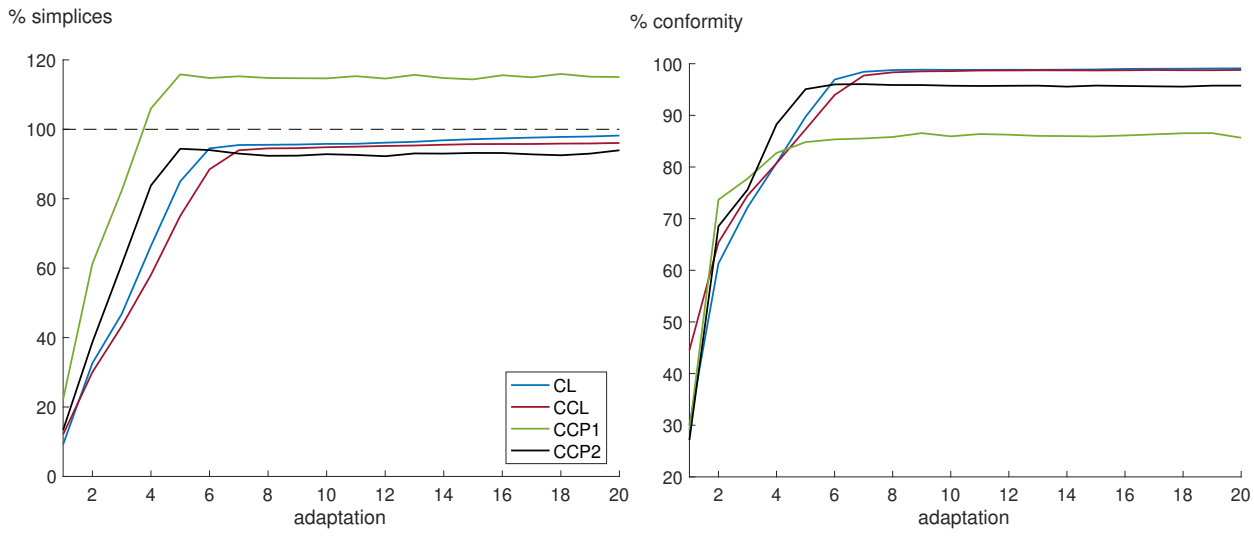
Table 3.2: Metric-conformity statistics for the UGAWG benchmark cases: Cube Linear (CL), Cube-Cylinder Linear (CCL), Cube-Cylinder Polar 1 (CCP1) and Cube-Cylinder Polar 2 (CCP2). The current algorithm is compared with the `feflo.a`[56] and `EPIC-ICSM`[50] software implementations. The expected number of simplices are shown in the top-left cells alongside the case labels.



(a) Edge lengths

(b) Simplex quality

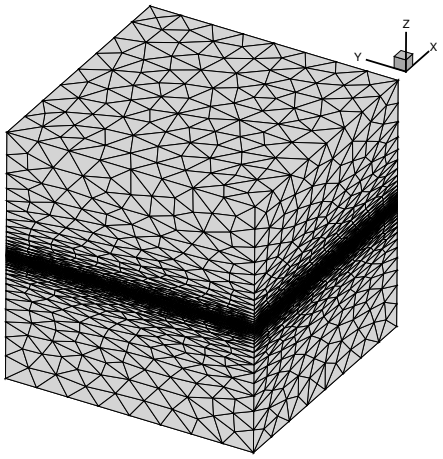
Figure 3.5: Metric conformity statistics for our algorithm applied to the UGAWG benchmark cases.



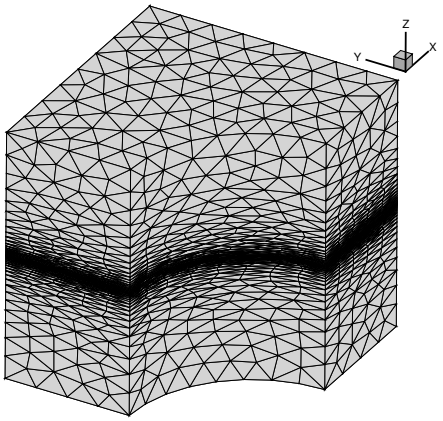
(a) % simplices

(b) % conformity

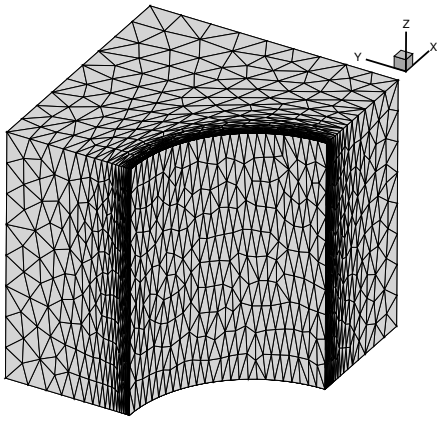
Figure 3.6: Normalized number of simplices (% simplices) and fraction of edges in the quasi-unit range (% conformity) produced by our algorithm for each 3*d* benchmark UGAWG case.



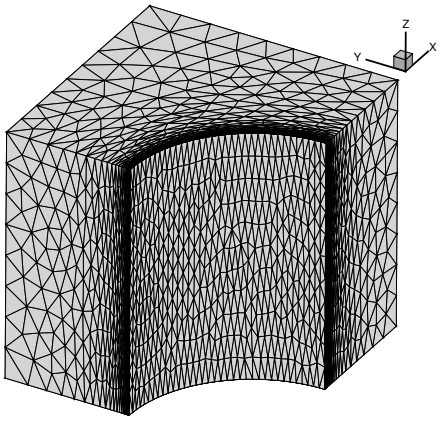
(a) Cube Linear



(b) Cube-Cylinder Linear



(c) Cube-Cylinder Polar 1



(d) Cube-Cylinder Polar 2

Figure 3.7: Meshes generated for the three-dimensional UGAWG benchmark cases.

Assessing the utility of various mesh adaptation components

We previously observed that our algorithm conforms well overall to some standard benchmark cases put forth by the Unstructured Grid Adaptation Working Group. An important question arises: what is the impact on metric conformity for the specific choices made in our adaptation algorithm? To answer this question, let us study some variants of our algorithm by turning off (or modifying) certain parameters that were previously discussed and used to produce a metric-conforming mesh.

In particular, we will study five variants applied to the Cube Linear case. The first variant consists of performing both stages with a target split length of $\ell_t = \sqrt{2}$ in contrast to performing the first stage with $\ell_t = \ell_0 = 2$ and a second stage of $\ell_t = \sqrt{2}$. This is referred to as the *Same Length* variant in the results that follow. Metric conformity, as reported in the edge length and quality histograms of Figure 3.9 appears better than the current algorithm (almost 100% in edge lengths and 93.47% in quality), however, observe the number of tetrahedra produced. Figure 3.10(a) as well as Table 3.3 highlight that this variant overshoots the expected number of tetrahedra by roughly 10%.

The second variant, consists of keeping all components of the algorithm with the exception of the `swapout` flag which was intended to swap out of restrictive topological configurations. Hence, this is referred to as the *No Swapout* variant. The results suggest that little difference is observed in metric conformity as compared to our current algorithm for this particular case.

◁ **No Swapout?**

This does not mean that we do not use swaps at all for this variant. We still use the global pass on swaps at Line 8,16 and 21 of Algorithm 3.6 but do not use them on Line 14 and 20 of Algorithm 3.4 or Line 24 of Algorithm 3.5.

The third variant consists of allowing for insertions without checking for the creation of short edges (*No Limit*). The result is a noticeable shift towards the creation of shorter edges and a significant 71% overshoot in the expected number of elements. Furthermore, the quality of the tetrahedra has degraded which is observed in Figure 3.9. This result provides a quantitative reasoning behind Loseille's suggestion[56] to perform edge splits without creating short edges in the process.

56. Loseille *et al., Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes.* 2017

The fourth variant consists of performing all stages of the algorithm in the absence of vertex smoothing (*No Smoothing*). Here, metric conformity drops to roughly 97% in edge lengths and 64% in tetrahedron quality and the number of simplices is overshot by 23%. The edge length histogram of Figure 3.9 is noticeably more sporadic and element quality is also compromised. The mesh produced by this variant (see Figure 3.8) also appears quite irregular. This result suggests that vertex smoothing is critical in achieving metric conformity and respecting the number of simplices expected by the metric.

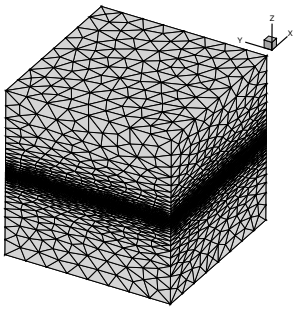
Finally, we attempt to mimic Loseille's algorithm by performing only a single stage of the overall algorithm (*Single Stage*). That is, we

perform a single stage of collapses and splits (with a target edge length of $\ell_t = \sqrt{2}$) followed by a single stage of mesh optimization using edge swaps and vertex smoothing. As in our current algorithm, metric conformity is excellent (almost 100% in edge lengths and 93% in tetrahedron quality) but the number of tetrahedra is overshot by roughly 12% to 43.6k. This is close to the 45.15k value reported for Loseille's software implementation `feflo.a` in the UGAWG benchmark paper. The results for the *Single Stage* variant seem to parallel those for the *Same Length* variant, though the latter achieves slightly better metric conformity for this case, possibly because it is effectively performing twice as many passes through the adaptation algorithm. The use of multiple stages obviously has an effect on the runtime of the adaptation algorithm but with a total runtime of 30-60 seconds per adaptation iteration, this is certainly acceptable, considering these meshes translate to roughly 160k DOF for a $p = 1$ discontinuous Galerkin discretization.

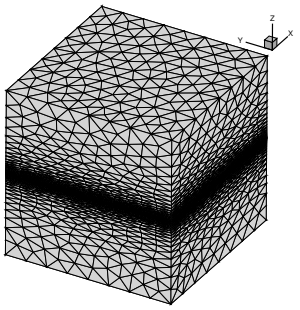
Let us conclude this section by remarking that, although we studied how the various adaptation components affect metric conformity, our algorithm is still heuristic. It is possible that a single measure of metric-conformity could be used to drive all adaptation components towards a common goal. An algorithm that consists of a single iterative loop in which each operator is appropriately selected to improve this measure of metric-conformity would be worthwhile to investigate.

Variant	Notes	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices
Current	-	0.67	1.77	1.06	99.10 %	0.47	0.90	92.15 %	38.30k
Variant 1	Same Length	0.57	1.66	1.03	99.92 %	0.09	0.90	93.46 %	42.67k
Variant 2	No Swapout	0.64	1.69	1.06	99.06 %	0.46	0.90	92.08 %	38.20k
Variant 3	No Limit	0.06	1.62	0.95	89.60 %	0.10	0.80	62.45 %	58.62k
Variant 4	No Smoothing	0.58	2.50	1.03	96.48 %	0.09	0.83	67.44 %	47.65k
Variant 5	Single Stage	0.55	1.72	1.02	99.74 %	0.17	0.90	92.25 %	43.60k

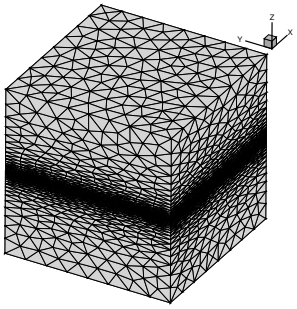
Table 3.3: Metric-conformity statistics obtained by slightly modifying Algorithm 3.6 when generating a mesh for the Cube Linear case. Recall that 39k tetrahedra are expected for this case.



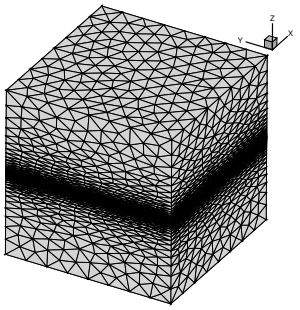
(a) Current



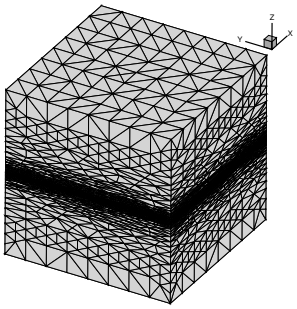
(b) Same Length



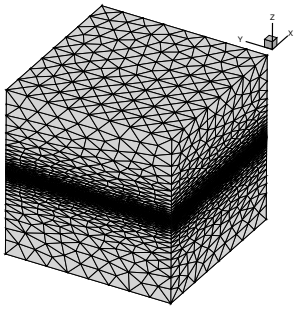
(c) No Swapout



(d) No Limit

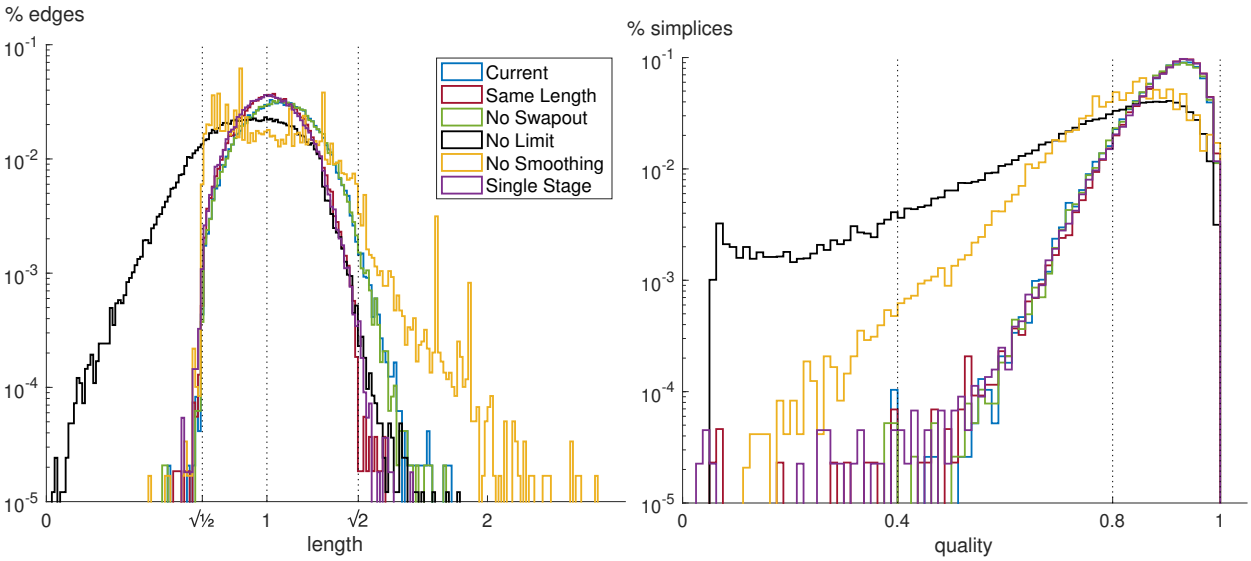


(e) No Smoothing



(f) Single Stage

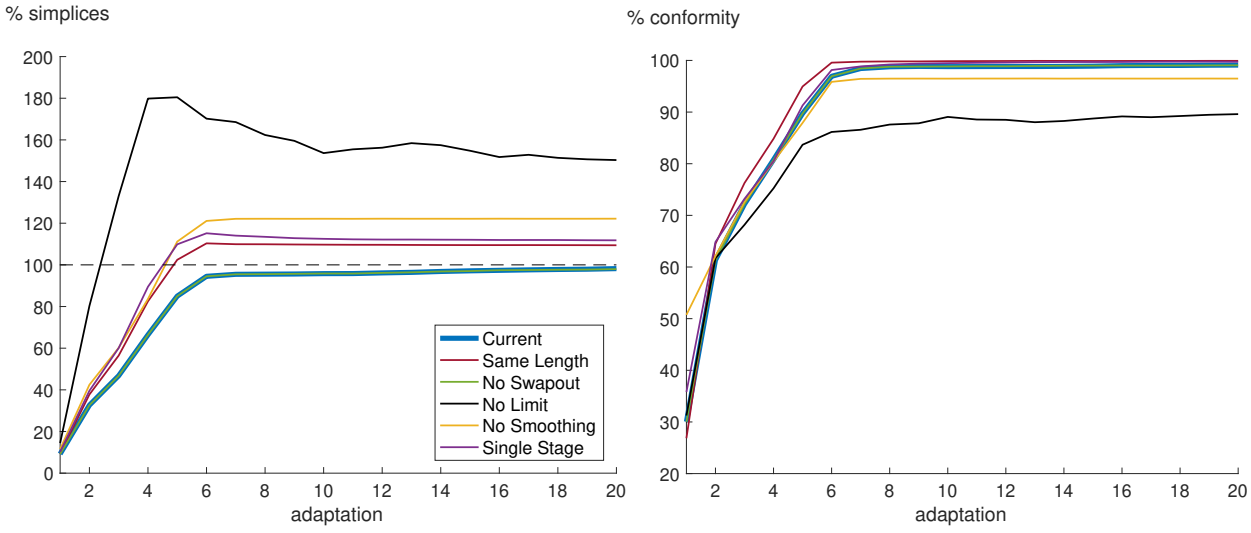
Figure 3.8: Meshes generated by variants of our mesh adaptation algorithm applied to the Cube Linear UGAWG case. With the exception of the case in which vertex smoothing is disabled, it is difficult to visually distinguish between the resulting meshes. The metric-conformity statistics of Figures 3.9 and 3.10 and Table 3.3 provide a better quantitative comparison of each variant.



(a) Edge lengths

(b) Simplex quality

Figure 3.9: Edge length and tetrahedra quality histograms obtained by slightly modifying Algorithm 3.6 when generating a mesh for the Cube Linear case.



(a) % simplices

(b) % conformity

Figure 3.10: Normalized number of tetrahedra (% simplices) and fraction of edges in the quasi-unit range (% conformity) when modifying Algorithm 3.6 for the Cube Linear case.

Demonstration on four-dimensional cases

Having demonstrated that Algorithm 3.6 (and its components) works well on three-dimensional benchmark problems, let us now study some four-dimensional metric-conforming problems. In so doing, the objective of this thesis related to demonstrating the first anisotropic four-dimensional mesh adaptation capability will be satisfied. Furthermore, we will assess the utility of the aforementioned DOF control factor.

Tesseract Linear (TL) The first problem, inspired by the UGAWG Linear metric, is represented analytically by

$$\mathbf{m}(\mathbf{x}) = \text{diag}\left(h_x^{-2}, h_y^{-2}, h_z^{-2}, h_t^{-2}\right), \tag{3.18}$$

where $h_x = h_y = h_z = h_{\max}$ and $h_t = h_0 + 2(h_{\max} - h_0)|t - 0.5|$, with $h_0 = 0.01 h_{\max}$. Meshes are generated for the two cases where $h_{\max} = [0.25, 0.125]$ so as to assess the performance at both low and moderate mesh sizes. Here, we expect to see six of the eight bounding hypercubes (non-constant t hyperplanes) to show refinement in the t direction. The constant t hyperplanes should exhibit uniform meshes. This case will be referred to as the Tesseract Linear (TL) case and will be further classified as the Tesseract Linear 1 ($h_{\max} = 0.25$) and Tesseract Linear 2 ($h_{\max} = 0.125$). The Linear 1 case expects 51k pentatopes whereas the Linear 2 case expects 818k pentatopes.

Tesseract Wave (TW) The second metric field is modeled after an expanding spherical wave in 3d (see Fig. 3.11). Consider a spherical wave of radius $R_0 = 0.4$ centered about the origin at time $t = 0$. If the wave expands at a constant velocity v_p to a radius $R_f = 0.8$ at time $t = 1$, then the expanding sphere traces the geometry of a hypercone in 4d.

Figure 3.12 exhibits the behaviour of the expanding $(d-1)$-sphere in a spherical-temporal coordinate system. Note that a slice of the $(d+1)$-dimensional cone with a hyperplane with non-constant temporal component yields a d-cone. Here, this appears as a line but rotational symmetry implies the hypercone sliced by a hyperplane with non-constant temporal component yields the three-dimensional cone.

The metric used to capture the propagation of this wave is $\mathbf{m}(\mathbf{x}) = \mathbf{Q}\,\text{diag}\left(h_r^{-2}, h_\theta^{-2}, h_\phi^{-2}, h_t^{-2}\right)\mathbf{Q}^t$. The eigenvectors $\mathbf{Q}$ are readily derived by rotating the spherical coordinate unit vectors by an angle α corresponding to the angle made by the velocity vector with the temporal axis. Only the radial unit vector is affected by the rotation which results in a basis given by

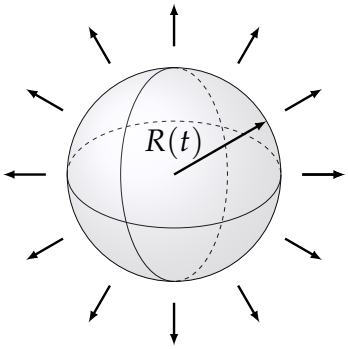


Figure 3.11: Sphere expanding at constant velocity.

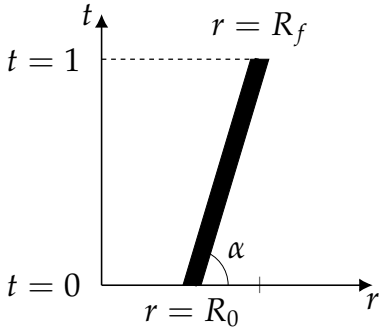


Figure 3.12: Temporal slice of a $d+1$-cone produced from the expansion of a d-sphere.

◁ **How do you compute the metric at the origin ($r = 0$)?**

Actually, we shift the spatial domain such that $\mathbf{x} \in [\epsilon, 1+\epsilon]^3$ for $\epsilon = 0.001$. To achieve the initial and final prescribed radii, these are $R_0 = 0.4 + \sqrt{3\epsilon^2}$ and $R_f = 0.8 + \sqrt{3\epsilon^2}$.

$$\mathbf{Q} = \begin{bmatrix} \sin\alpha\cos\phi\sin\theta & \cos\phi\cos\theta & -\sin\phi & \cos\alpha\cos\phi\sin\theta \\ \sin\alpha\sin\phi\sin\theta & \cos\theta\sin\phi & \cos\phi & \cos\alpha\sin\phi\sin\theta \\ \sin\alpha\cos\theta & -\sin\theta & 0 & \cos\alpha\cos\theta \\ -\cos\alpha & 0 & 0 & \sin\alpha \end{bmatrix}. \tag{3.19}$$

The spacings in the tangential directions are

$$h_\theta, h_\phi = \begin{cases} h_1, & |r - R(t)| > \delta, \\ (h_1 - h_2)|r - R(t)|/\delta + h_2, & |r - R(t)| \leq \delta, \end{cases} \tag{3.20}$$

with $R(t) = R_0 + (R_f - R_0)t$ is the position of the spherical wave with time. The spacing in the radial direction is $h_r = h_0 + 2(h_1 - h_0)|r - R(t)|$ and the spacing in the temporal direction is $h_t = 0.5$. Note the use of spherical coordinates, $r = \sqrt{x^2 + y^2 + z^2}$, $\theta = \arccos(z/r)$ and $\phi = \arctan(y, x)$. The rotation angle is equal to $\alpha = \arctan(t_f - t_0, R_f - R_0)$. The remaining parameters are $h_0 = 0.0025$, $h_1 = 0.125$, $h_2 = 0.05$ and $\delta = 0.1$. We were unable to determine the analytic number of pentatopes for this case, however, using numerical quadrature on the resulting meshes provides an estimate of 275k pentatopes.

Results We follow the same assessment procedure as was done for the three-dimensional benchmarks (see Algorithm 3.7). Here, we additionally assess the utility of the DOF control factor, f_{dof}, which was proposed in an attempt to reduce the overshoot in the number of pentatopes.

Meshes for the Tesseract Linear 1 case are provided in Figure 3.16 and those for the Tesseract Wave case are provided in Figure 3.17. The edge length and quality histograms of Figure 3.14 demonstrate an acceptable level of metric conformity for all cases. The poorest metric conformity is observed in the difficult Tesseract Wave case with the enabled DOF control during insertions, though this is only 3% lower than that without control.

More importantly, the DOF control factor is successful in more closely achieving the expected number of simplices for both Tesseract Linear cases without sacrificing metric conformity. In particular, observe that the fraction of edges in the quasi-unit range is between 97-98% for both Tesseract Linear 1 and 2 cases, with or without DOF control – see Table 3.4. Furthermore, the number of pentatopes overshoots the expected number by 17% and 12% for the Tesseract Linear 1 and 2 cases, respectively, without DOF control (Figure 3.15). With the control factor enabled, we obtain an overshoot of 9% for the Tesseract Linear 1 case and an undershoot of 0.4% for the Tesseract Linear 2 case. This significant improvement in achieving the expected number of pentatopes suggest (1) larger meshes are needed to achieve good levels of metric conformity and (2) the DOF control factor is effective at resisting

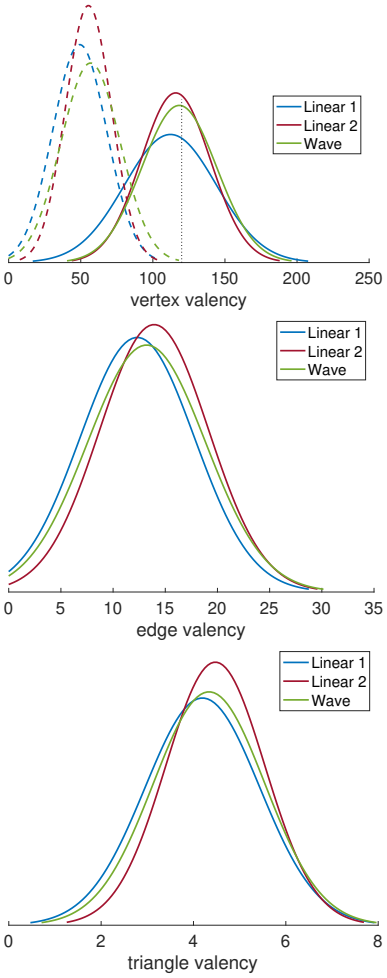


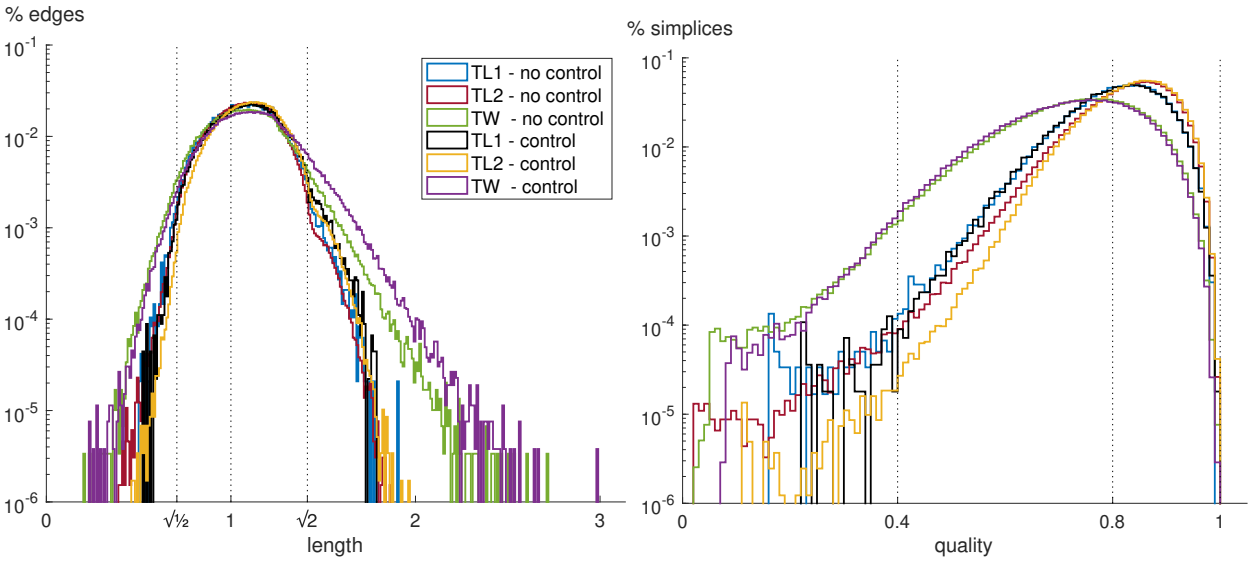
Figure 3.13: Fitted vertex, edge and triangle valencies for the metric-conforming meshes of the four-dimensional benchmark cases. The dashed lines for the vertex figure represents the valency for boundary vertices whereas the solid one represents the valency for interior vertices. The dotted line in the same figure marks a valency of 120.

pentatope overshoot without sacrificing metric conformity.

Furthermore, we provide valency statistics for the produced four-dimensional meshes in Figure 3.13. We define the valency as the number of pentatopes touching a lower dimensional facet. That is, the vertex valency is the number of pentatopes attached to a vertex, the edge valency is the number of pentatopes attached to an edge, etc. Observe the peak in interior vertex valency (the solid lines in the top figure) near 120, which aligns with the valency of a vertex within a 120-cell. The boundary vertex valencies are drawn in dashed. The full statistics for the vertex, edge, triangle and face valencies are tabulated in Table 3.5. Note that for the larger mesh (Linear 2) the average tetrahedron valency (which accounts for both the interior and boundary tetrahedra) approaches two.

		ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
Linear 1	(no control)	0.50	1.91	1.08	97.29 %	0.16	0.80	56.67 %	59.56k	16.78 %
Linear 1	(control)	0.53	1.78	1.10	96.46 %	0.23	0.80	56.35 %	55.66k	9.14 %
Linear 2	(no control)	0.40	1.86	1.08	98.01 %	0.02	0.83	67.13 %	915.30k	11.90 %
Linear 2	(control)	0.47	1.96	1.11	97.27 %	0.11	0.83	70.00 %	814.50k	-0.43 %
Wave	(no control)	0.20	2.71	1.08	92.48 %	0.02	0.72	28.75 %	394.07k	43.30 %
Wave	(control)	0.23	2.99	1.12	88.96 %	0.08	0.72	26.89 %	347.19k	26.25 %

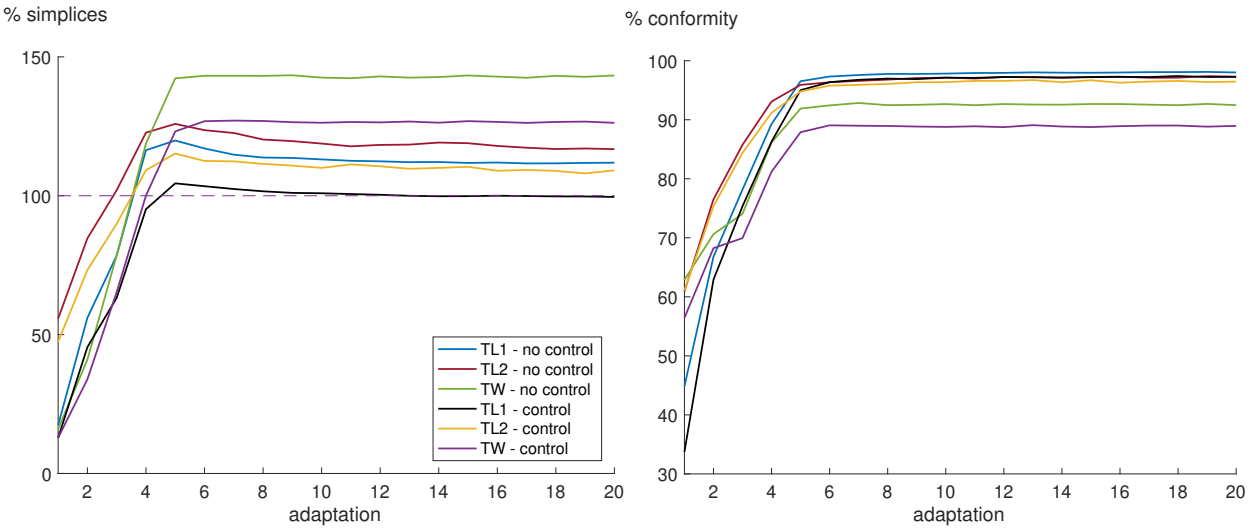
Table 3.4: Metric-conformity statistics for the 4*d* benchmark cases with and without DOF control enabled.



(a) Edge lengths

(b) Simplex quality

Figure 3.14: Metric conformity statistics obtained from Algorithm 3.6 for the four-dimensional benchmark cases.



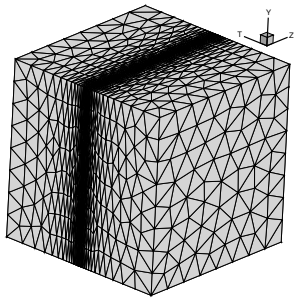
(a) % simplices

(b) % conformity

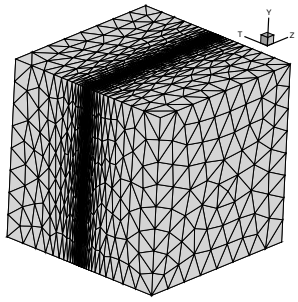
Figure 3.15: Normalized number of simplices (% simplices) and fraction of edges in the quasi-unit range (% conformity) for the four-dimensional benchmark cases. Since the number of pentatopes expected for the Tesseract Wave case is unknown, the normalization is done with respect to the number of pentatopes obtained with the DOF control factor enabled.

Property	Linear 1	Linear 2	Wave
# interior vertices, v_i	1.0k	26.0k	9.0k
mean valency, $\bar{v}_{v_i}$	112.21	115.85	118.33
# boundary vertices, v_b	3.0k	20.0k	13.0k
mean valency, $\bar{v}_{v_b}$	49.28	55.33	56.79
# edges, e	45.0k	588.0k	267.0k
mean valency, $\bar{v}_e$	12.31	13.93	13.21
# triangles, t	133.0k	1831.0k	813.0k
mean valency, $\bar{v}_t$	4.19	4.47	4.34
# tetrahedra, f	147.0k	2108.0k	920.0k
mean valency, $\bar{v}_f$	1.89	1.94	1.92

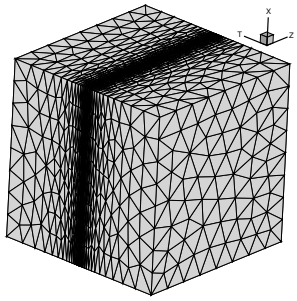
Table 3.5: Number of vertices, edges, triangles and tetrahedra along with the corresponding mean valencies for the metric-conforming meshes of the four-dimensional benchmark cases.



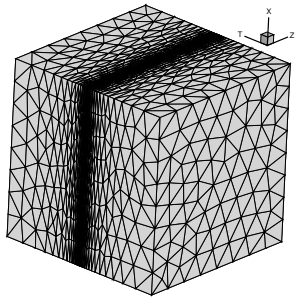
(a) $x = 0$



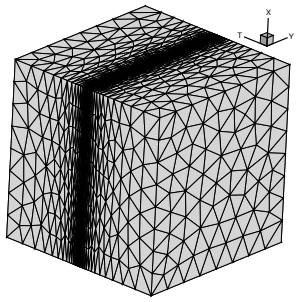
(b) $x = 1$



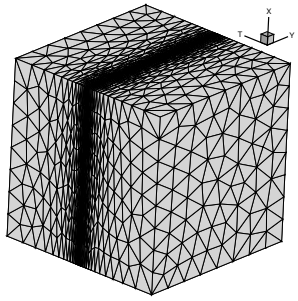
(c) $y = 0$



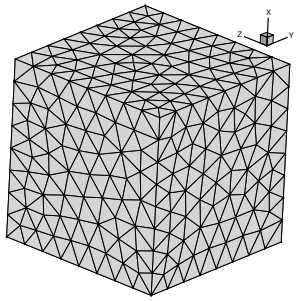
(d) $y = 1$



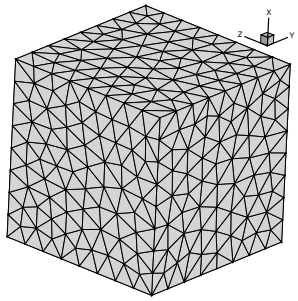
(e) $z = 0$



(f) $z = 1$

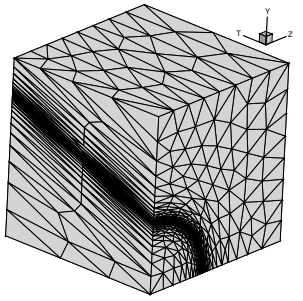


(g) $t = 0$

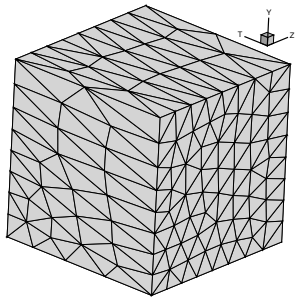


(h) $t = 1$

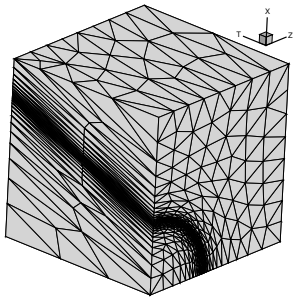
Figure 3.16: Meshes generated by Algorithm 3.6 for the Tesseract Linear 2 case.



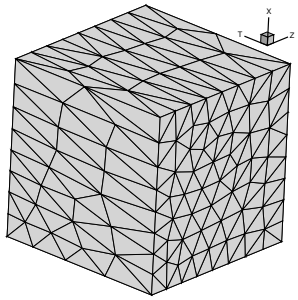
(a) $x = 0$



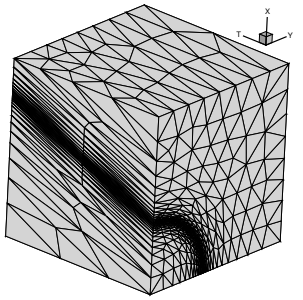
(b) $x = 1$



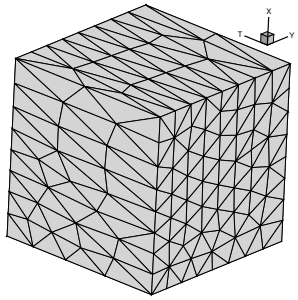
(c) $y = 0$



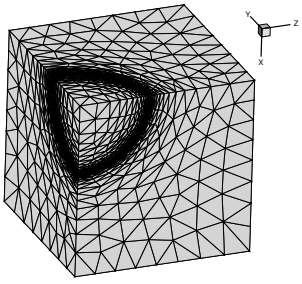
(d) $y = 1$



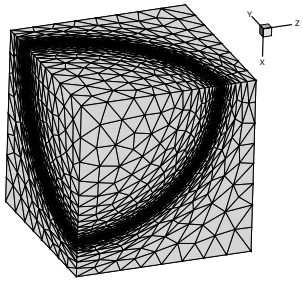
(e) $z = 0$



(f) $z = 1$



(g) $t = 0$



(h) $t = 1$

Figure 3.17: Meshes generated from our adaptation algorithm for the Tesseract Wave case.

3.6 Perspectives

This chapter developed the theory and implementation details of our mesh adaptation algorithm. We then demonstrated our algorithm on three-dimensional benchmark cases, inspired by the recent work of the Unstructured Grid Adaptation Working Group. The utility of the mesh adaptation components was studied by assessing how each component affects metric conformity, both in the sense of edge length and simplex quality histograms as well as in achieving the expected number of simplices. However, we re-iterate that our algorithm is heuristic. Future work may involve unifying these criteria for metric-conformity into a single quality criterion that would drive a more rigorous mesh adaptation schedule.

Finally, the algorithm was demonstrated on some newly introduced four-dimensional benchmark cases. We re-iterate that this is the first time four-dimensional anisotropic mesh adaptation has been demonstrated in the literature. An important feature of the algorithm for four-dimensional cases was the introduction of a DOF control factor, used to restrict insertions from causing a significant overshoot in the expected number of simplices.

Though the four-dimensional cases studied here are contained within the unit tesseract, the algorithm can handle time-dependent geometries with changing topologies, provided the vertex-to-geometry associations are specified. The same algorithm used to construct the unit tesseract geometry in Appendix A can be used to construct more complicated straight-sided geometries that represent these moving boundaries. Otherwise, some suggestions for generating time-varying geometry descriptions are provided at the end of the the aforementioned appendix. Generating the initial four-dimensional mesh for more complicated geometries remains an open problem.

Order	$p = 1$		$p = 2$		$p = 3$		$p = 4$	
Case	dG	cG	dG	cG	dG	cG	dG	cG
Linear 1	277.9k	4.0k	833.9k	49.2k	1.9M	226.9k	3.9M	537.1k
Linear 2	4.1M	46.0k	12.3M	633.8k	28.7M	3.1M	57.3M	7.3M
Wave	1.8M	21.6k	5.3M	288.6k	12.3M	1.4M	24.7M	3.3M

Table 3.6: Cost of the discontinuous (dG) and continuous (cG) discretizations with various polynomial orders p for the metric-conforming meshes produced for the four-dimensional benchmark cases.

The adaptation algorithm is currently restricted to a serial implementation, the performance of which is acceptable for the problem sizes studied in this work. One iteration of the mesh adaptation algorithm takes approximately 30-60 seconds for moderately sized (about 40k tetrahedra) three-dimensional problems and 15-20 minutes for the larger four-dimensional meshes with 800-900k pentatopes. Future work consists of parallelizing the mesh adaptation components using both

shared- and distributed-memory approaches. This might consist of adapting partitions of the input mesh while keeping the boundaries of the partitions fixed, followed by an adaptation of these partition boundaries as in the recent work of Loseille[56] or the work of Digonnet[82]. Furthermore, a heterogeneous approach similar to that of Ibanez[75] or of Tsolakis[83] is also attractive.

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

82. Digonnet *et al.*, *Massively Parallel Anisotropic Mesh Adaptation*. 2017

75. Ibanez, *Conformal Mesh Adaptation on Heterogeneous Supercomputers*. 2016

83. Tsolakis *et al.*, *Parallel Anisotropic Unstructured Grid Adaptation*. 2019

Software implementation

We conclude this chapter by introducing our software, `avro`, the source code of which is openly available. The name for the software was motivated by the initial intent to develop an adaptation tool using Voronoi diagrams with isometric embeddings (which is clearly not the case anymore), hence an Adaptive VoROnoi mesher. Despite the functionality in the software moving in a different direction, the name stuck, mostly in tribute to the Canadian Avro CF-105 Arrow of the 1950s.

Metric-conformity was the primary goal of our mesh adaptation software implementation. However, it is equally important to consider the performance of the algorithm components and design the data structures to attain a reasonable level of efficiency. Ideally, the computational cost of the algorithm should be dependent on the number of local operations needed and not on the size of the mesh. For example, the most frequent operation in the mesh adaptation tool is the identification of the set of elements sharing a common facet f: $\mathcal{C}(f)$ (Equation 3.2). An exhaustive search through the mesh would incur an order $\mathcal{O}(|\mathcal{T}|)$ cost which would be too costly to be done when looping over all edges or vertices of the mesh. A more efficient implementation for this operation is achieved by exploiting the simplex neighbour relations along with what is called the *inverse* topology. Further details are provided in Appendix C.

◁ **Did you know?**

The Avro Arrow program was abruptly shutdown in February of 1959. For more information on the Arrow, please click here.

Chapter 4

Applications to adaptive numerical simulations

> It comes down to the types of problems you want to solve.
>
> — Siva Nadarajah

4.1 Background

This chapter employs the mesh adaptation algorithm of Chapter 3 in conjunction with the Mesh Optimization via Error Sampling and Synthesis Algorithm of Section 2.5 to demonstrate the first fully unstructured four-dimensional mesh adaptation capability to solve partial differential equations. The focus is to demonstrate the capability with discontinuous Galerkin solution fields applied to linear and quadratic polynomial orders.

The implementation details related to performing the local splits for the sampling described in Section 2.5 within the *Solution Adaptive Numerical Simulator* (SANS)[84] framework are provided in Appendix C. The remaining discretization tools are achieved by extending the framework in SANS to support four-dimensional meshes and solution fields. In particular, Lagrange basis functions over simplex elements are used to represent the high-order polynomial bases and the conical product of Stroud is used to perform the numerical integration. It should be noted that the aforementioned tensor-product quadrature rules can be computationally expensive when integrating high-order polynomials, however, this has not been a computational bottleneck for the problems in this thesis.

84. Galbraith *et al.*, *A Verification Driven Process for Rapid Development of CFD Software.* 2015

◁ **Warning!**

We first tried the Grundmann-Möeller quadrature rules[85] but, since these rules allow for negative quadrature weights, they were insufficient because the L^2 error (which should be positive) in the discrete solution sometimes evaluated to a negative value due to these weights.

4.2 L^2 error control

As a first step towards fully unstructured $3d + t$ spacetime PDE-driven adaptation, let us demonstrate that our algorithm finds the optimal L^2 approximant of a prescribed four-dimensional function. That is, we will find the mesh that best approximates some scalar function u on a prescribed polynomial basis of order p for a fixed computational cost, represented as the number of degrees-of-freedom (DOF). The discrete solution $u_{h,p}$, for some polynomial order p is obtained from the L^2 projection of the analytic function, u, to the p-th order polynomial basis:

$$u_{h,p} = \underset{v_{h,p} \in \mathcal{V}_{h,p}}{\arg\inf} \, ||u - v_{h,p}||^2_{L^2(\Omega)} = \underset{v_{h,p} \in \mathcal{V}_{h,p}}{\arg\inf} \int_\Omega (u - v_{h,p})^2 \mathrm{d}\mathbf{x} \tag{4.1}$$

where u is the analytic function we wish to approximate with some mesh $\mathcal{M}$ on a domain Ω. Again, $\Omega = [0,1]^4$ is the unit tesseract.

The advantage of studying the performance of the adaptation algorithm on this problem is that the error used to drive the adaptation algorithm can be analytically computed from the L^2 error in contrast to employing the error estimation techniques of Section 2.5. That is, the output of interest is the exact L^2 error in the solution:

$$\mathcal{J}(u) = \sqrt{\int_\Omega \left(u - u_{h,p}\right)^2 \mathrm{d}\mathbf{x}} \tag{4.2}$$

Furthermore, Yano[3] provides a framework within which the resulting mesh size and aspect ratio distributions can be compared with analytic optimums so as to verify the mesh adaptation components in the absence of approximations arising from error estimation techniques. The rate matrices in the local error models (Section 2.5) obtained with the exact L^2 error provide an excellent representation of the error over a single element. The exact L^2 error from Equation 4.2 should asymptotically converge at a rate of h^{p+1} ($h = \sqrt[4]{\mathrm{DOF}}$) for smooth solutions as the mesh is refined[3,86,87].

Algorithm 4.1 describes the steps used to perform the adaptation in the following subsections. Starting with an initial mesh $\mathcal{M}_0$ (here, the Kuhn-Freudenthal triangulation[88]), the algorithm performs `maxIter` iterations of projecting the analytic function u onto the current mesh $\mathcal{M}$ and performing the MOESS algorithm of Section 2.5, driven by the exact L^2 error of Equation 4.2 with a fixed target DOF c_t. The refinement factor, h_{ref}, is set to 1.6. With this low refinement factor, one hundred adaptation iterations are used to ensure there are enough adaptations to obtain the optimal mesh. This optimal mesh, $\mathcal{M}^*$, is then evaluated for metric conformity and compared with analytic mesh size and aspect ratio distributions, where available.

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

86. Houston *et al.*, *Adaptivity and A Posteriori Error Estimation for DG Methods on Anisotropic Meshes*. 2006

87. Cao, *An Interpolation Error Estimate on Anisotropic Meshes in R^n and Optimal Metrics for Mesh Refinement*. 2007

88. Kuhn, *Simplicial Approximation of Fixed Points*. 1968

adaptL2Error

input: $\mathcal{M}_0$, c_t, p, u, h_{ref}, `maxIter`
output: $\mathcal{M}^*$

$\mathcal{M} \leftarrow \mathcal{M}_0$
for `iter = 1,...,maxIter`
  $u_{h,p} \leftarrow$ solve Equation 4.1 on current mesh $\mathcal{M}$
  $\mathbf{m} \leftarrow$ **moess**$(\mathcal{M}, u_{h,p}, h_{\text{ref}})$
  $\mathcal{M} \leftarrow$ **adapt**$(\mathcal{M}, \mathbf{m})$ using Algorithm 3.6
$\mathcal{M}^* \leftarrow \mathcal{M}$

Algorithm 4.1: Adaptation algorithm to compute the optimal L^2 approximant of a $4d$ function u with a target computational cost c_t, polynomial order p of the discrete solution. The algorithm starts from an initial mesh $\mathcal{M}_0$ and performs `maxIter` adaptation iterations to produce the optimal mesh $\mathcal{M}^*$.

Boundary layer

The first function we consider is an extension of the regularized boundary layer studied by Yano[3] to four dimensions:

$$u(x,y,z,t) = \exp(-x/\epsilon) + \frac{\beta_y}{(p+1)!}y^{p+1} + \frac{\beta_z}{(p+1)!}z^{p+1} + \frac{\beta_t}{(p+1)!}t^{p+1}. \tag{4.3}$$

The first term causes a strong gradation in the x direction whereas the remaining three regularization terms ensure the aspect ratios in the y, z and t directions remain bounded.

The mesh which best approximates the function in Equation 4.3 has an optimal mesh grading (h_x) perpendicular to the wall ($x = 0$) with[3]

$$h_x = h_{x,0}\exp(k_{h_x}x),\ k_{h_x} = \frac{2p+5}{\epsilon(p+1)(2p+6)}, \tag{4.4}$$

where $h_{x,0}$ is a constant determined by the computational cost constraint. The aspect ratio distributions, a_i, in the three remaining directions are

$$a_i = a_{i,0}\exp(k_{a_i}x),\ a_{i,0} = \frac{1}{\epsilon\beta_i^{\frac{1}{p+1}}},\ k_{a_i} = -\frac{1}{\epsilon(p+1)},\ i = y,z,t. \tag{4.5}$$

Here, $\epsilon = 0.01$, $\beta_y = 2^{p+1}$, $\beta_z = 4^{p+1}$ and $\beta_t = 6^{p+1}$.

We restrict our attention to the discontinuous Galerkin (dG) solution spaces (Equation 2.28) and study linear ($p = 1$) and quadratic ($p = 2$) polynomial orders.

The convergence of the L^2 error and DOF counts are shown in Figure 4.1 for meshes optimized at 64k, 128k, 256k and 512k DOF for both $p = 1$ and $p = 2$. The 64k DOF $p = 2$ adaptation sequence exhibits the least overshoot (less than 20%) which is surprising considering this mesh contains the fewest number of pentatopes (about

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes*. 2012

◁ **Computing $h_{x,0}$:**

For the dG discretization, the cost c_t relates to the metric complexity (by combining Equations 4.4 and 4.5) via

$$c(\mathbf{m}) = \int_0^1 h_x^4 a_y a_z a_t \mathrm{d}x = c_t v_\Delta$$

where v_Δ is the volume of the equilateral pentatope (see Chapter 2). The only unknown is the wall mesh size $h_{x,0}$.

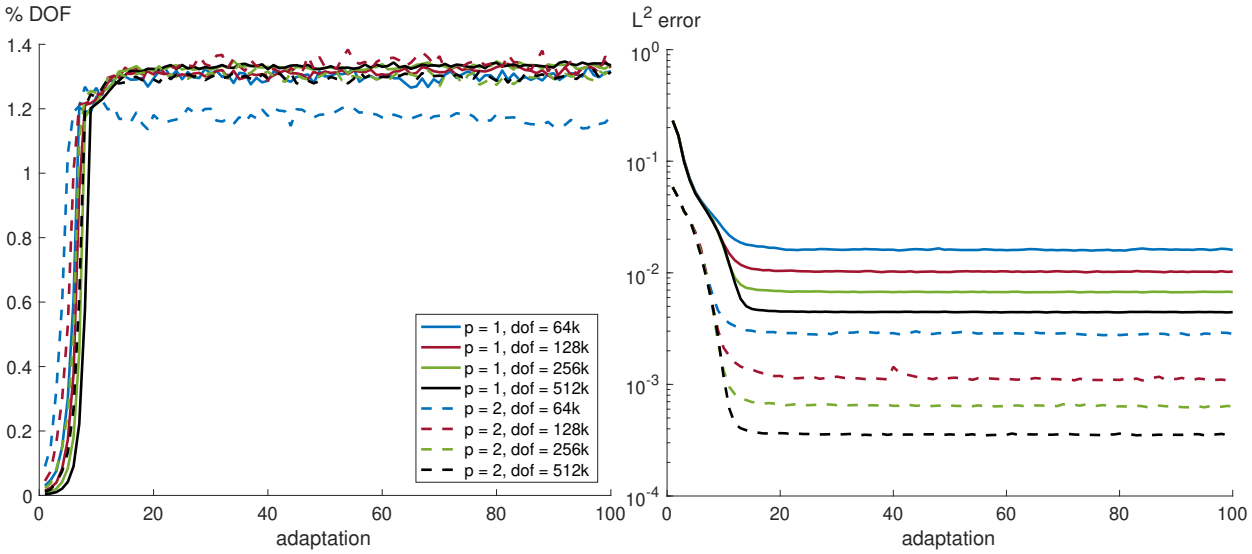
5k) and, in general, this would make the mesh adaptation task more difficult. All other cases exhibit a DOF overshoot of about 30% but this does not severely affect the convergence of the error as Figure 4.1(b) shows a very smooth and steady convergence of the error as the mesh is optimized over the adaptation iterations. Though 30 adaptation iterations were certainly enough to compute the optimal meshes, the steady behaviour of the error over all 100 adaptation iterations is a good verification of the adaptation algorithm.

Metric conformity is not as high as in the four-dimensional benchmark cases (see Table 4.1). Though the fraction of edges within the quasi-unit range is acceptable (92-96%), the quality of the pentatopes is quite poor ($\%q_{\text{unit}} \approx 35\%$) which, in turn, causes the overshoot in the expected number of pentatopes and, hence, DOF counts. The analytic metric for this case is described by Equations 4.4 and 4.5 and can be passed into Algorithm 3.7 (of Chapter 3). However, note, that the mesh size in Equation 4.4 grows exponentially away from the wall and would directly yield an unwieldy mesh size, much larger than the length of the domain. As a result, the expressions for the mesh size and aspect ratios are assumed reliable near the boundary layer (here, we take $[0, 0.05]$). Beyond this region, a cubic fit of the mesh size obtained from the MOESS-optimized meshes (at a selected DOF) is obtained and the aspect ratios are assumed constant beyond this region. The fit for the $p = 1$ 64k-optimized mesh results in $h_x(x) \approx -1.69x^3 + 1.34x^2 + 0.71x + 0.02$, $h_y \approx 0.41$, $h_z \approx 0.22$ and $h_t \approx 0.15$ which are used beyond the region $[0, 0.05]$. Providing this analytic metric through Algorithm 3.7 yields metric conformity statistics of 95.42% in quasi-unit edge lengths and 48.1% in quasi-unit pentatopes (quality greater than 0.8). Furthermore, the number of pentatopes is 12,532 which is very close to the expected value of 12.8k for a 64k $p = 1$ discretization for which the metric was optimized. Comparing these values with the first column of Table 4.1 shows that only slight improvement is obtained with the analytic metric, though the number of DOF is matched much better.

To verify the optimality of the produced meshes, consider the mesh size and aspect ratios obtained near the $x = 0$ wall. Equations 4.4 and 4.5 suggest the mesh size and aspect ratios should be linear in x versus $\log(h_x)$ and x versus $\log(a_i)$ $(i = y, z, t)$. Near the wall, the mesh sizes are computed directly from the diagonal entries of the implied metric of each pentatope. That is, $h_i(\kappa) \approx (\mathbf{m}_\kappa)_{i,i}^{-1/2}$ $(i = 1, 2, 3, 4$, representing x, y, z and t directions, respectively). The aspect ratios are then $a_i = h_i/h_x$ $(i = y, z, t)$. For both linear and quadratic polynomial bases, the distributions of the mesh size and aspect ratios for the 512k-optimized meshes are shown in Figure 4.2. A linear regression of the size and aspect ratios in $x - \log(h_x)$ and $x - \log(a_i)$ $(i = y, z, t)$ spaces

◁

This is, in fact, pessimistic since the eigenvectors of the implied metric may not be directly aligned with the Cartesian axes. The fact that the results are in good agreement with the analytic distributions is a good sign that the various components are working correctly.



(a) DOF fraction

(b) L^2 error

Figure 4.1: Convergence of the DOF (as a fraction of the target) and the L^2 error in the solution for the boundary layer L^2 error control case.

$p = 1$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.49	2.35	1.10	92.40 %	0.14	0.74	35.64 %	16.84k	31.59 %
128k	0.39	2.20	1.11	92.35 %	0.11	0.75	36.19 %	33.98k	32.73 %
256k	0.44	2.28	1.10	92.18 %	0.11	0.75	36.99 %	68.54k	33.86 %
512k	0.31	2.54	1.10	92.21 %	0.09	0.75	37.53 %	136.65k	33.45 %

$p = 2$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.60	1.77	1.08	96.42 %	0.37	0.79	52.95 %	5.02k	17.66 %
128k	0.50	2.02	1.10	92.48 %	0.18	0.74	33.89 %	11.50k	34.72 %
256k	0.41	2.26	1.10	93.20 %	0.20	0.75	36.47 %	22.65k	32.71 %
512k	0.42	2.05	1.10	92.74 %	0.18	0.75	36.47 %	44.88k	31.50 %

Table 4.1: Metric conformity statistics at the final adaptation iteration for the L^2 boundary layer error control case with both $p = 1$ (top) and $p = 2$ (bottom) discretizations.

shows the distributions are well-aligned with the analytic mesh distributions (shown in dashed). Table 4.2 tabulates the full set of regression coefficients for all target DOF with both $p = 1$ and $p = 2$ discretizations. For $p = 1$, better alignment with the analytic values for both the wall aspect ratios and gradings away from the wall is obtained as the number of DOF is increased. For $p = 2$, the wall aspect ratios are all fairly good and an improvement is seen in the gradings as the number of target DOF is increased.

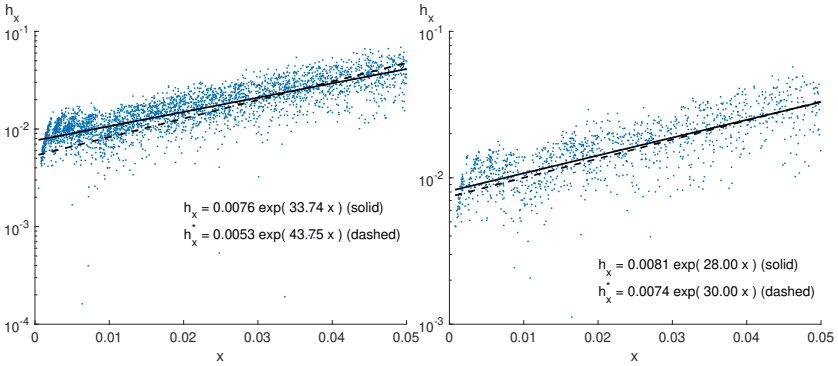
For a larger boundary layer thickness ($\epsilon = 0.1$), the fitted mesh size and aspect ratio distributions are provided in Table 4.3. These coefficients were obtained by fitting the mesh size and aspect ratios for each pentatope with a centroid x-coordinate between $x \in [0, 0.1]$. Again, good conformity with the analytic optimal distributions are observed for the $p = 1$ and $p = 2$ meshes. Specifically, the wall values ($h_{x,0}$, $a_{y,0}$, $a_{z,0}$ and $a_{t,0}$) are close to the analytic ones whereas the rates exhibit a larger deviation with the analytic values.

The meshes obtained at 512k for both $p = 1$ and $p = 2$ polynomial orders are shown in Figures 4.3 and 4.4. As expected, the meshes at constant x-hyperplanes exhibit the least anisotropy as there is little gradation in the solution in the y, z or t directions. All other bounding cubes with a variation in x effectively resolve the boundary layer with anisotropic simplices near the $x = 0$ boundary.

$p = 1$	$h_{x,0}$	$h^*_{x,0}$	k_{h_x}	$a_{y,0}$	k_{a_y}	$a_{z,0}$	k_{a_z}	$a_{t,0}$	k_{a_t}
Analytic	$h^*_{x,0}$	-	43.75	50.00	-50.00	25.00	-50.00	16.67	-50.00
64k	0.0155	0.0090	25.14	25.52	-26.57	14.32	-26.84	9.59	-26.11
128k	0.0121	0.0075	28.39	29.46	-31.14	16.02	-31.79	10.75	-31.44
256k	0.0095	0.0063	31.29	32.15	-35.22	17.09	-35.92	11.76	-35.58
512k	0.0076	0.0053	33.74	33.76	-38.05	18.18	-38.92	12.47	-39.25

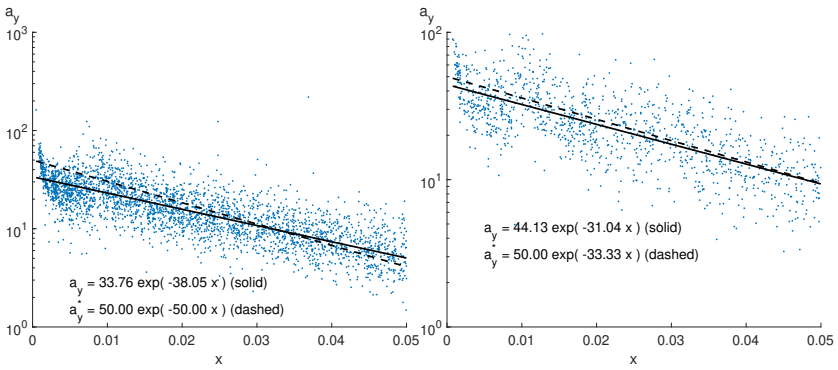
$p = 2$	$h_{x,0}$	$h^*_{x,0}$	k_{h_x}	$a_{y,0}$	k_{a_y}	$a_{z,0}$	k_{a_z}	$a_{t,0}$	k_{a_t}
Analytic	$h^*_{x,0}$	-	30.00	50.00	-33.33	25.00	-33.33	16.67	-33.33
64k	0.0146	0.0125	25.46	38.06	-24.69	22.41	-25.96	15.39	-26.60
128k	0.0119	0.0105	24.56	42.92	-28.12	21.51	-27.69	14.55	-28.50
256k	0.0100	0.0088	27.36	40.45	-29.80	22.11	-30.55	14.64	-30.14
512k	0.0081	0.0074	28.00	44.13	-31.04	22.06	-30.68	15.22	-31.74

Table 4.2: Mesh size and aspect ratio regression coefficients obtained from the $p = 1$ (top) and $p = 2$ (bottom) optimized meshes for the L^2 error control boundary layer case with $\epsilon = 0.01$. Analytic values for the regression coefficients are listed in the top rows.



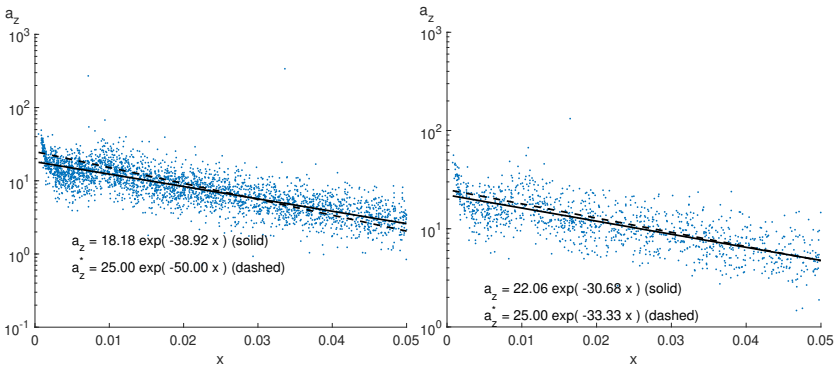
(a) h , $p = 1$

(b) h , $p = 2$



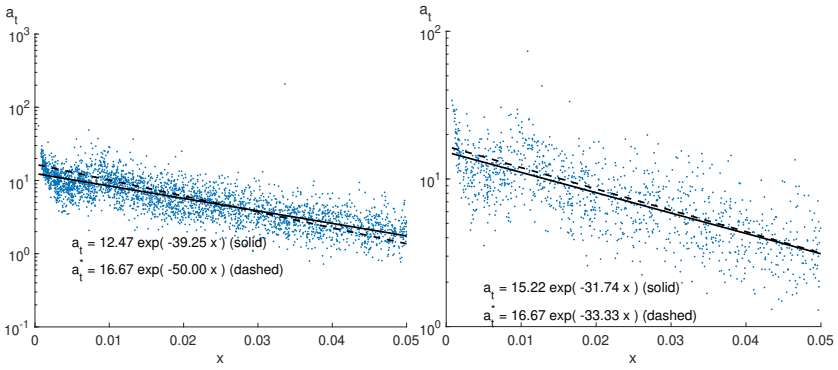
(c) a_y , $p = 1$

(d) a_y , $p = 2$



(e) a_z , $p = 1$

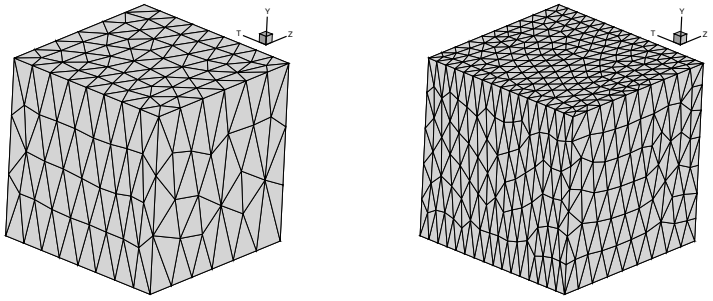
(f) a_z , $p = 2$



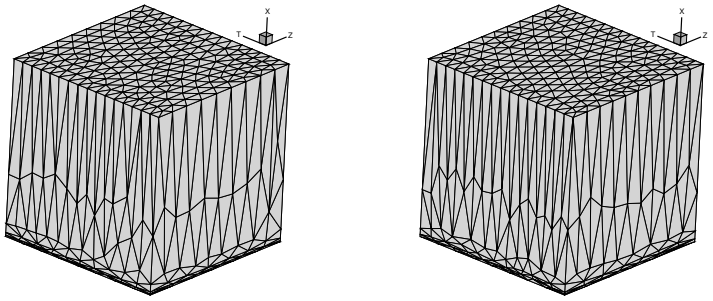
(g) a_t , $p = 1$

(h) a_t , $p = 2$

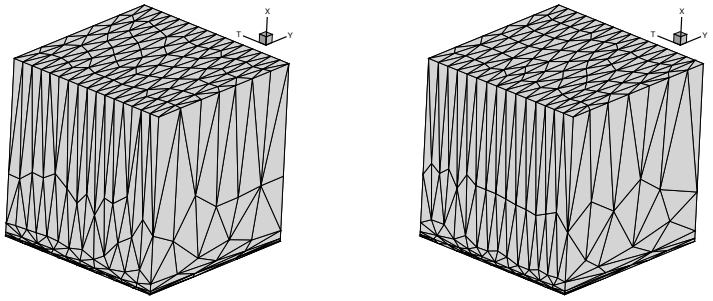
Figure 4.2: Mesh size and aspect ratio distribution perpendicular to the $x = 0$ wall of the 512k-optimized meshes when adapted to the L^2 error between the discrete solution and the $4d$ boundary layer function of Equation 4.3. The analytic distributions are plotted in dashed whereas the fitted ones are plotted with the solid lines.



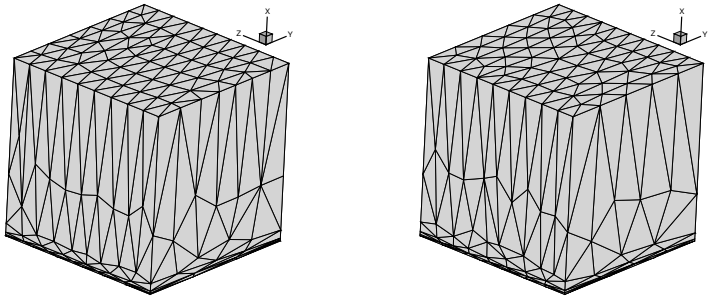
(a) $x = 0$ **(b)** $x = 1$



(c) $y = 0$ **(d)** $y = 1$

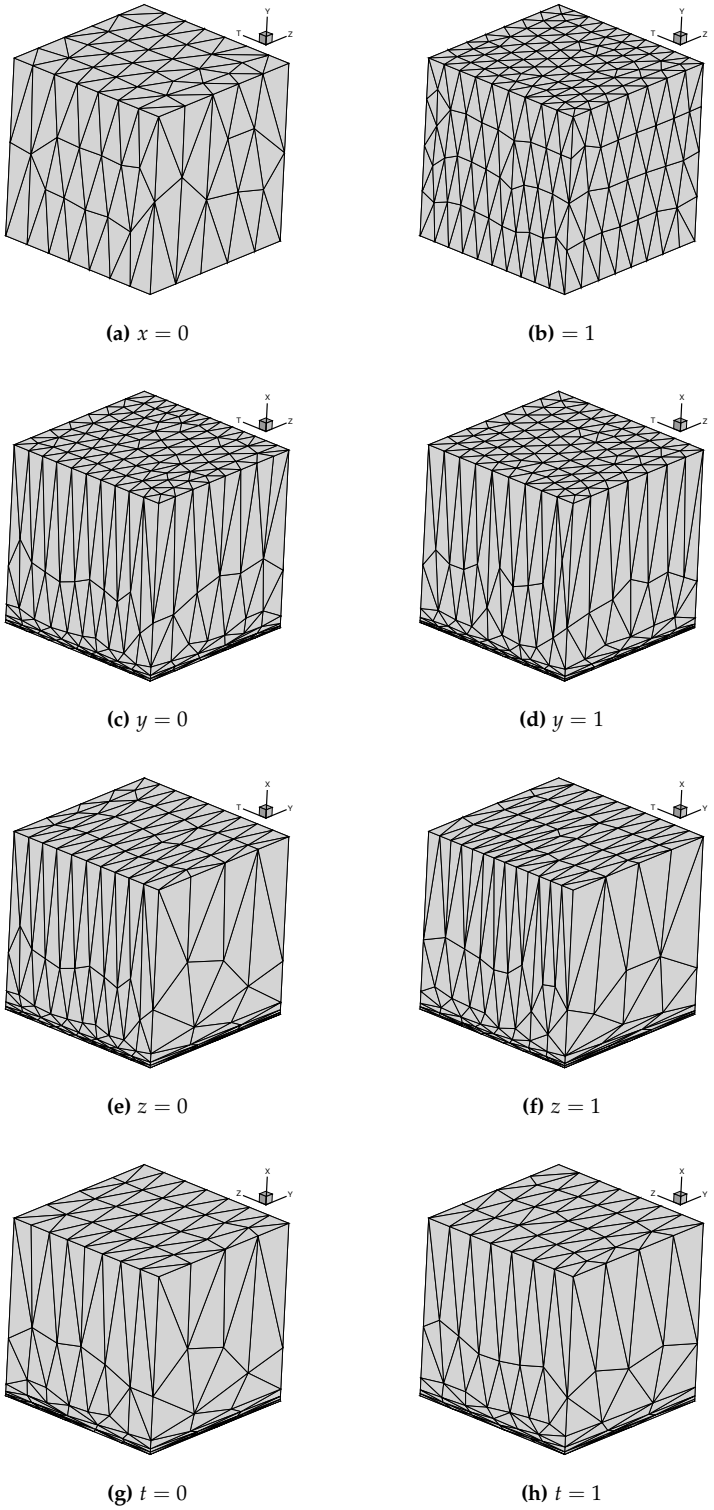


(e) $z = 0$ **(f)** $z = 1$



(g) $t = 0$ **(h)** $t = 1$

Figure 4.3: Bounding cube discretizations extracted from the final optimized pentatopal mesh at 512k DOF for the L^2 error control boundary layer case ($p = 1$).



(a) $x = 0$ **(b)** $= 1$

(c) $y = 0$ **(d)** $y = 1$

(e) $z = 0$ **(f)** $z = 1$

(g) $t = 0$ **(h)** $t = 1$

Figure 4.4: Bounding cube discretizations extracted from the final optimized pentatopal mesh at 512k DOF for the L^2 error control boundary layer case ($p = 2$).

$p=1$	$h_{x,0}$	$h^*_{x,0}$	k_{h_x}	$a_{y,0}$	k_{a_y}	$a_{z,0}$	k_{a_z}	$a_{t,0}$	k_{a_t}
Analytic	$h^*_{x,0}$	-	4.38	5.00	-5.00	2.50	-5.00	1.67	-5.00
64k	0.0835	0.0877	5.67	4.62	-7.45	2.49	-7.90	1.67	-7.08
128k	0.0785	0.0737	3.00	4.21	-5.06	2.06	-3.48	1.40	-3.51
256k	0.0643	0.0620	3.00	4.10	-3.84	2.13	-3.46	1.44	-3.25
512k	0.0517	0.0521	3.67	4.32	-4.19	2.25	-4.17	1.53	-4.28

$p=2$	$h_{x,0}$	$h^*_{x,0}$	k_{h_x}	$a_{y,0}$	k_{a_y}	$a_{z,0}$	k_{a_z}	$a_{t,0}$	k_{a_t}
Analytic	$h^*_{x,0}$	-	3.00	5.00	-3.33	2.50	-3.33	1.67	-3.33
64k	0.0906	0.1202	6.43	5.98	-7.37	3.02	-8.35	2.13	-8.81
128k	0.0899	0.1011	3.63	4.71	-4.73	2.43	-4.71	1.72	-5.28
256k	0.0762	0.0850	2.93	4.83	-3.03	2.48	-3.72	1.60	-3.29
512k	0.0651	0.0715	2.33	4.73	-3.33	2.36	-2.33	1.61	-2.72

Table 4.3: Mesh size and aspect ratio regression coefficients obtained from the $p = 1$ (top) and $p = 2$ (bottom) optimized meshes for the L^2 error control boundary layer case with $\epsilon = 0.1$. Analytic values for the regression coefficients are listed in the top rows.

Expanding spherical wave

Now, consider a function modeling the expansion of a spherical wave, similar to the Tesseract Wave case of Chapter 3:

$$u(\mathbf{x},t) = k_0 \exp(-\alpha t) \exp\left(-k_1 (r(t) - ||\mathbf{x}||)^2\right), \quad \mathbf{x} \in \mathbb{R}^3,\ t \in [0,1] \tag{4.6}$$

with $r(t) = r_0 + v_s t$, $\alpha = 1$, $k_0 = 1$, $k_1 = 200$, $v_s = 0.7$, $r_0 = 0.4$. The strength of the wave, initially k_0, decays exponentially in time at a rate of α. The parameter k_1 controls the width of wave strength δ about the increasing radius $r(t)$. In particular, since Equation 4.6 is a normal distribution; roughly 99.7% of u will lie within three standard deviations of the nominal wave radius $r(t)$. This thickness can be approximated as

$$\delta \approx 3\sigma = \frac{3}{\sqrt{2k_1}}. \tag{4.7}$$

For this problem, $\delta \approx 0.15$ and a total of 2δ should be visibly refined about the expanding sphere by the adaptation algorithm.

Since Equation 4.6 is a function of only r and t, we can perform the adaptation sequence in two dimensions, resulting in a 1000-DOF optimized mesh shown in Figure 4.5(a) and the solution on this mesh in Figure 4.5(b). The expected resolution of the wave is obtained and we should expect the same behaviour in the four-dimensional setting.

Performing the same adaptation sequence as in Algorithm 4.1 in the four-dimensional setting yields the convergence of the DOF and L^2 error over all target DOFs shown in Figure 4.6. The DOF overshoot at the final adaptation is much less than in the boundary layer case, ranging from 14-20% and metric conformity is in fact much better than in the latter case. Furthermore, Table 4.5 demonstrates that 97-99% of the edges are in the quasi-unit range and at least 48% of the pentatopes have a quality greater than 0.8. Despite good metric

◁ **Remember Figure 1.2(b)?**

This problem is actually identical to the motivating example in the introduction, whereby the feature has been rotated about the temporal axis.

◁ **Why 2δ?**

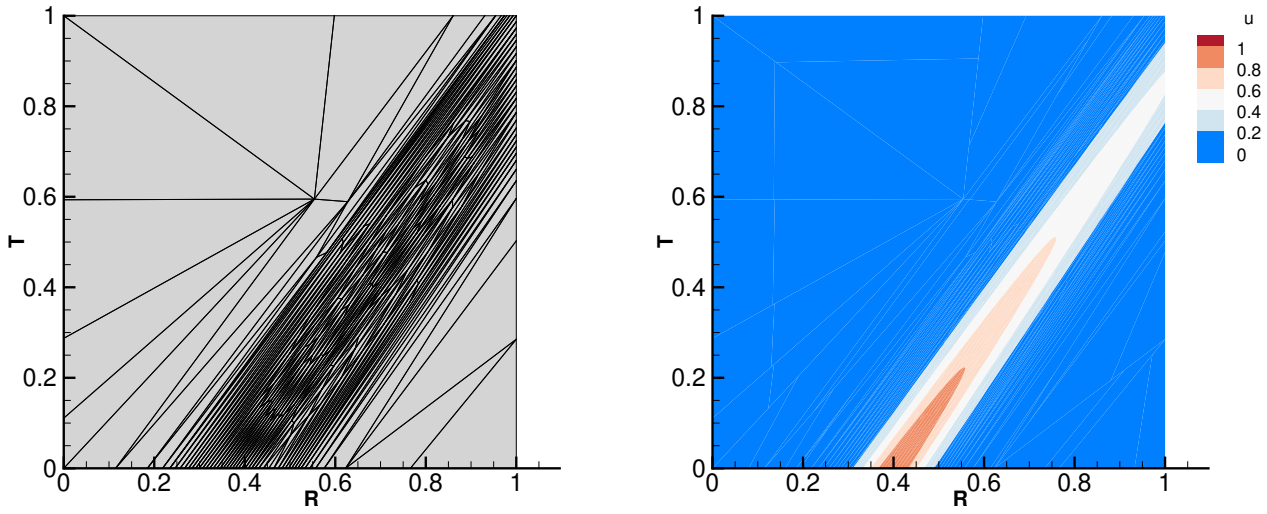
99.7% of the function in Equation 4.6 is contained within ($\pm$) three standard deviations (σ) of the expanding sphere radius. Therefore, 3σ should be refined on either side of this radius.

conformity, the convergence of the L^2 error over the adaptation iterations (Figure 4.6(b)) shows a much noisier behaviour. Further work may include improving the implied metric calculation to obtain better metrics from one adaptation sequence to the next. Also notice that, although the error increases after its initial drop at the beginning of the adaptation sequence, this behaviour is balanced by the decrease in DOF (Figure 4.6(a)) as the metric optimization algorithm persists in trying to match the requested target DOF. All adaptation sequences begin with a large DOF overshoot and, as the adaptation proceeds, the mesh is coarsened resulting in a closer match with the target DOF. For example, the $p = 1$ adaptation sequence with a target of 512k DOF hovers at above 20% overshoot until roughly the 60^{th} iteration at which point the L^2 error increases as well.

The $p = 1$ and $p = 2$ optimized meshes obtained for the target DOF request of 512k are shown in Figures 4.7 and 4.8, respectively. The expected three-dimensional cones are seen at constant x, y and z hyperplanes whereas the initial sphere (with radius $r_0 = 0.4$) and final sphere (with radius $r_f = 1.1$) are correctly obtained at constant t hyperplanes. Note that the expected width ($2\delta \approx 0.3$) of the solution around the expanding wave is seen since the width of the mesh resolution is approximately one third in each spatial direction. Furthermore, the stretching in the direction of the wave is evident; on average, two elements are needed in the temporal direction. The maximum aspect ratios for all meshes are tabulated in Table 4.4. These were computed as the maximum aspect ratio (over all pentatopes) where the maximum aspect ratio of a pentatope is computed from the eigenvalues of its implied metric as $a_{\max} = \sqrt{\lambda_{\max}/\lambda_{\min}}$. The $p = 1$ mesh optimized for 512k DOF exhibits the highest anisotropy, with the maximum aspect ratio greater than 10^3.

Order / DOF	64k	128k	256k	512k
p = 1	2.07e+02	5.07e+02	8.32e+02	2.12e+03
p = 2	6.53e+01	1.26e+02	2.60e+02	5.88e+02

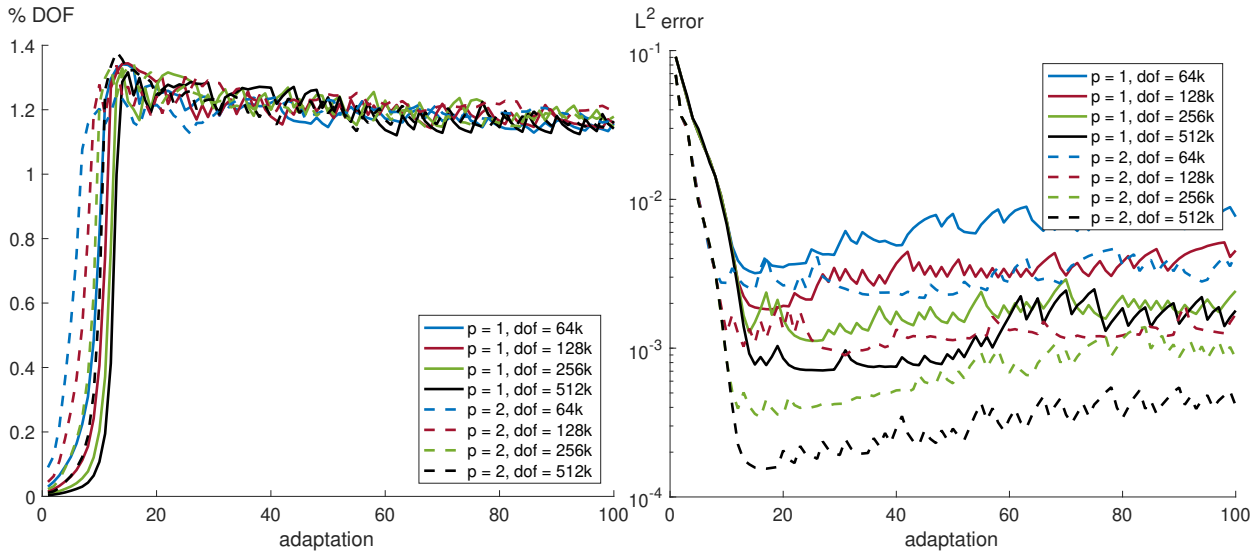
Table 4.4: Maximum aspect ratios of the pentatopal meshes optimized at various target DOF requests for the spherical wave L^2 error control case ($p = 1$ and $p = 2$).



(a) Mesh

(b) Solution, u

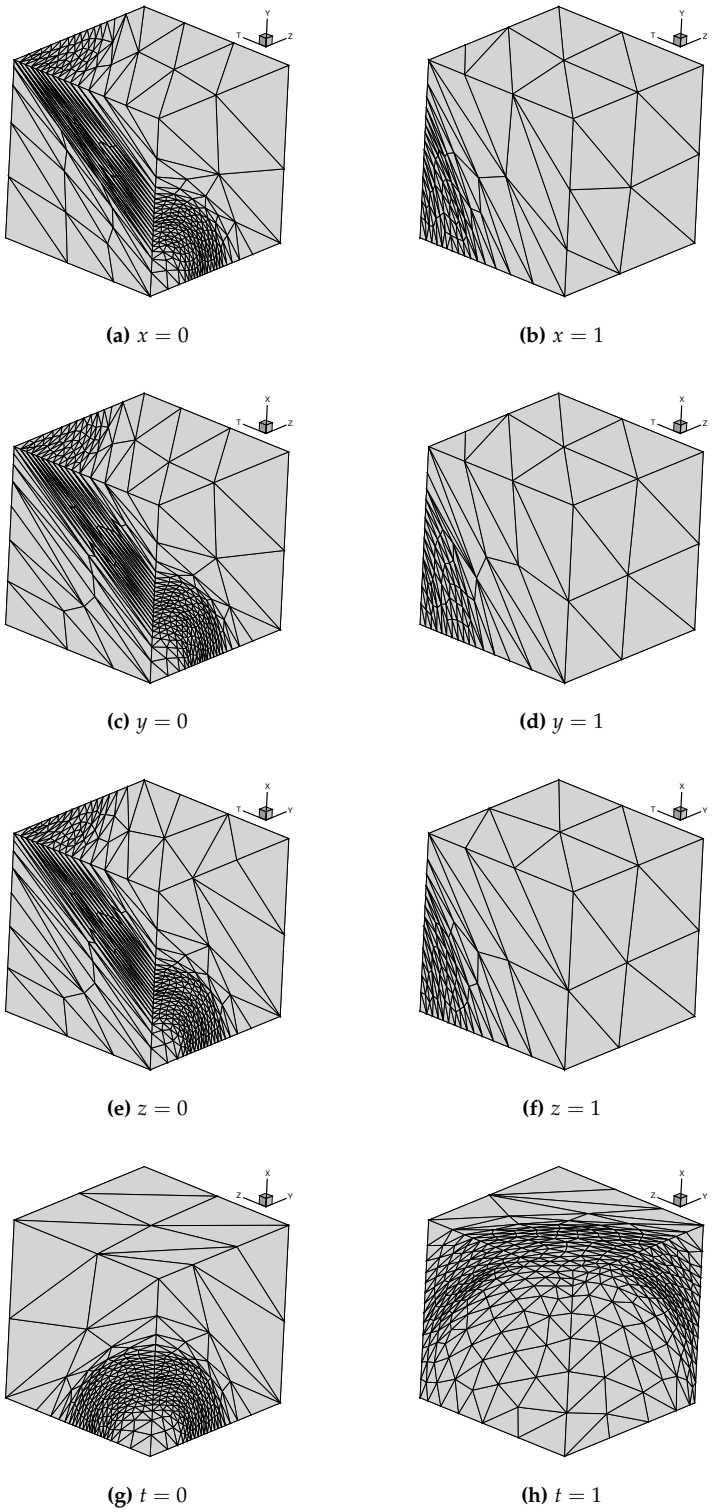
Figure 4.5: 100ok-DOF ($p = 1$) optimized mesh and solution u when adapting to the L^2 error in Equation 4.6 in two dimensions (r and t).



(a) DOF

(b) L^2 error

Figure 4.6: Convergence of the DOF (as a fraction of the target) and the L^2 error in the solution for the spherical wave L^2 error control case.



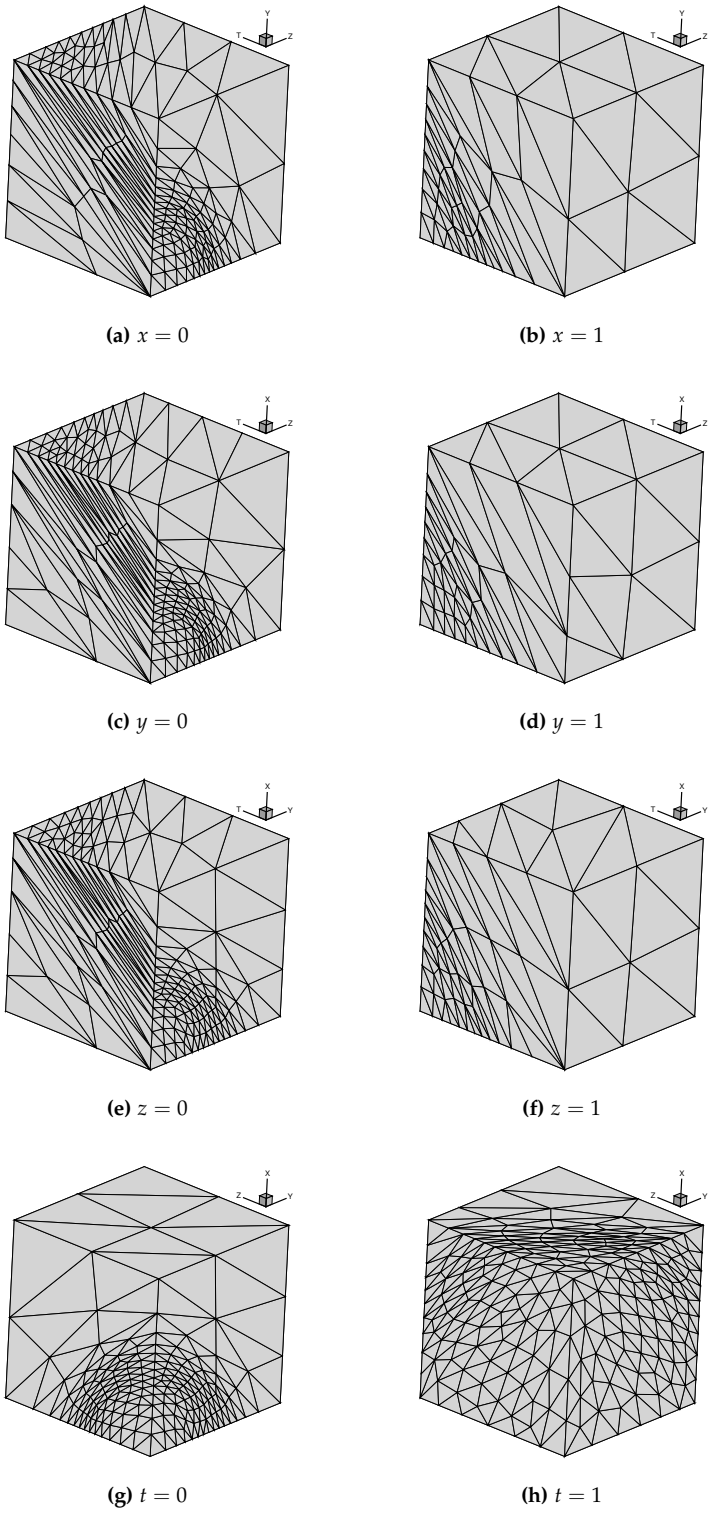
(a) $x = 0$ **(b)** $x = 1$

(c) $y = 0$ **(d)** $y = 1$

(e) $z = 0$ **(f)** $z = 1$

(g) $t = 0$ **(h)** $t = 1$

Figure 4.7: Bounding cube discretizations extracted from the final optimized pentatopal mesh at 512k DOF for the L^2 error control spherical wave case ($p = 1$).



(a) $x = 0$ **(b)** $x = 1$

(c) $y = 0$ **(d)** $y = 1$

(e) $z = 0$ **(f)** $z = 1$

(g) $t = 0$ **(h)** $t = 1$

Figure 4.8: Bounding cube discretizations extracted from the final optimized pentatopal mesh at 512k DOF for the L^2 error control spherical wave case ($p = 2$).

$p=1$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.57	1.88	1.08	97.72 %	0.28	0.80	53.31 %	14.83k	15.84 %
128k	0.55	1.90	1.08	97.92 %	0.23	0.79	51.85 %	29.64k	15.79 %
256k	0.58	1.93	1.09	97.72 %	0.24	0.79	48.11 %	58.72k	14.69 %
512k	0.54	2.08	1.09	97.57 %	0.11	0.79	51.76 %	116.86k	14.12 %

$p=2$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.62	1.76	1.08	98.71 %	0.35	0.79	50.15 %	4.87k	14.12 %
128k	0.60	1.60	1.07	99.31 %	0.31	0.79	51.99 %	10.21k	19.61 %
256k	0.55	1.86	1.08	97.47 %	0.29	0.79	49.59 %	20.12k	17.87 %
512k	0.56	1.96	1.08	97.40 %	0.20	0.79	49.63 %	39.76k	16.47 %

Table 4.5: Metric conformity statistics at the final adaptation iteration for the L^2 spherical wave error control case with both $p = 1$ (top) and $p = 2$ (bottom) discretizations.

Convergence of the L^2 error

We close this section by studying the convergence of the output functional, $\mathcal{J}(u)$, of Equation 4.2 with mesh refinement. The L^2 error in the solution and approximate mesh size, $h \approx \sqrt[4]{\text{DOF}}$, from the last five adaptation iterations are plotted in circles and these values from the last two target DOF requests are fit in $\log h - \log \mathcal{E}$ ($\mathcal{E}$ here being the exact L^2 error) space to estimate the rate of convergence. The boundary layer function of Equation 4.3 is four-dimensional and we expect an asymptotic convergence rate of approximately h^{p+1}[3,86,87]. This is certainly observed in the rates obtained for both $p = 1$ and $p = 2$ as seen in Figure 4.9(a) though the rates are slightly higher than expected. This may be due to the fact that the meshes are still in the pre-asymptotic range and would likely approach rates of $\mathcal{E} \sim h^{p+1}$ should finer mesh resolutions be studied. Unfortunately, the computational resources needed to run more expensive simulations were not available at the time of this work.

For the spherical wave case, the fitted convergence rates are even more accelerated than h^{p+1} which may be due to the fact that the function is effectively two-dimensional in an $r - t$ coordinate system. Thus the convergence rate should approach $h^{2(p+1)}$, though, again the results obtained from the simulations exhibit pre-asymptotic behaviour.

To verify that the algorithm can indeed reach the asymptotic range with sufficient mesh resolution, consider a very simple four-dimensional function:

$$u(x, y, z, t) = \exp(-\lambda t) \sin \pi x \sin \pi y \sin \pi z, \quad x, y, z, t \in [0, 1]. \quad (4.8)$$

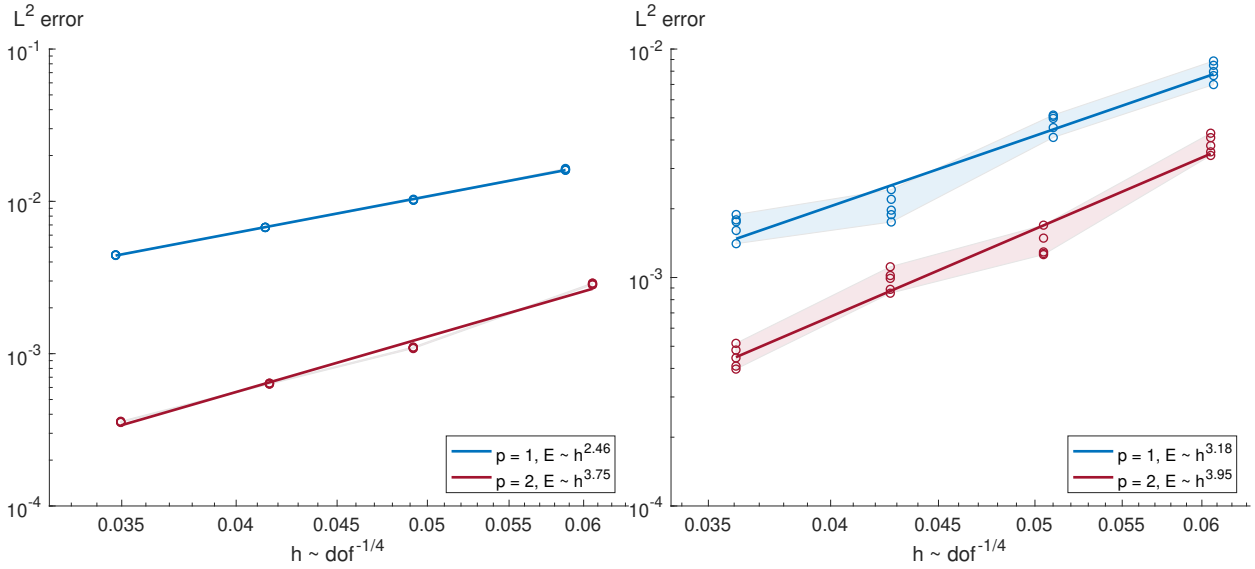
With $\lambda = 5$, the initial three-dimensional sinusoidal function decays quickly in the temporal direction and the function should require smaller elements near the $t = 0$ boundary with larger ones near the $t = 1$ boundary. Again, for both $p = 1$ and $p = 2$ solution orders, Algorithm 4.1 was used to generate optimal meshes for 64k, 128k and 256k

3. Yano, *An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes.* 2012

86. Houston *et al.*, *Adaptivity and A Posteriori Error Estimation for DG Methods on Anisotropic Meshes.* 2006

87. Cao, *An Interpolation Error Estimate on Anisotropic Meshes in R^n and Optimal Metrics for Mesh Refinement.* 2007

target DOF requests which should be sufficient to reach the asymptotic convergence regime. Figure 4.10 demonstrates that, indeed, the L^2 error in the solution decays at a rate of h^{p+1} which is a verification that the four-dimensional adaptation algorithm is working correctly and suggests the previous cases require more expensive simulations to reach the asymptotic range.



(a) Boundary layer

(b) Spherical wave

Figure 4.9: Convergence of the L^2 error with mesh refinement for the boundary layer and spherical wave cases ($p = 1$ and $p = 2$).

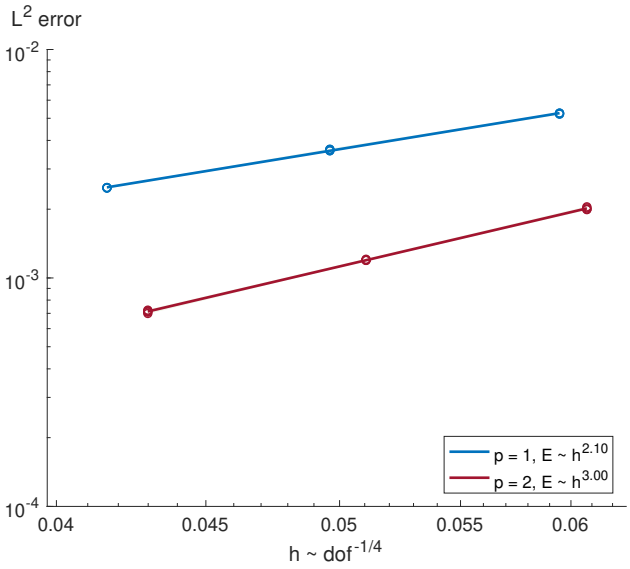


Figure 4.10: Convergence of the L^2 error with mesh refinement for the sinusoidal decay case ($p = 1$ and $p = 2$) verifies the h^{p+1} expected convergence rate.

4.3 Scalar advection-diffusion

We now demonstrate the mesh adaptation algorithm applied to the scalar advection-diffusion equation. The method of manufactured solutions (MMS) is used to compute the source term in Equation 2.24 by substituting a prescribed analytic solution into the left-hand side of the PDE, resulting in

$$s(u, \mathbf{x}, t) = \frac{\partial u}{\partial t} + \mathbf{c} \cdot \nabla u - \nu \nabla^2 u. \tag{4.9}$$

Dirichlet boundary conditions with the exact solution are applied on the spatial boundaries and the initial condition is set to $u(\mathbf{x}, 0)$.

The output of interest is the integral of the squared-solution over the domain, resulting in the analytic and discrete versions below:

$$\mathcal{J}(u) = \int_{\Omega} u^2 \, \mathrm{d}\mathbf{x}, \quad \mathcal{J}(u_h) = \int_{\mathcal{M}} u_h^2 \, \mathrm{d}\mathbf{x}. \tag{4.10}$$

Adapting to the output in Equation 4.10 should resolve the entire solution and the L^2 error in the solution should exhibit an asymptotic convergence rate of h^{p+1} as the mesh is refined.

The major difference between the L^2 error control cases of Section 4.2 and the current studies is that the solution is obtained by solving the linear system of equations that arise when discretizing the scalar advection-diffusion PDE and that the dual-weighted residual error estimation technique is used to drive the adaptive process.

The overall procedure used here is outlined in Algorithm 4.2. The algorithm is nearly identical to the one used to drive the L^2 error control cases with the exception that Line 4 employs the solution to the PDE and Line 5 employs dual-weighted residual techniques to model the local error during the MOESS algorithm in contrast to using the analytic error of the L^2 error control cases. In the applications that follow, the refinement factor will again be set to $h_{\text{ref}} = 1.6$ and 100 adaptation iterations will be used to find the optimal mesh for target DOF requests of 64k, 128k, 256k and 512k.

adaptAdvectionDiffusion

input: $\mathcal{M}_0$, c_t, p, u, h_{ref}, `maxIter`
output: $\mathcal{M}^*$

$\mathcal{M} \leftarrow \mathcal{M}_0$
for `iter = 1,...,maxIter`
  $u_{h,p} \leftarrow$ solve Equation 2.24 on current mesh $\mathcal{M}$
  $\mathbf{m} \leftarrow$ **moess**$(\mathcal{M}, u_{h,p}, h_{\text{ref}})$ using Equation 2.39
  $\mathcal{M} \leftarrow$ **adapt**$(\mathcal{M}, \mathbf{m})$ using Algorithm 3.6
$\mathcal{M}^* \leftarrow \mathcal{M}$

Algorithm 4.2: Adaptation algorithm to compute the optimal mesh to resolve the solution to a $3d + t$ advection-diffusion PDE with a target computational cost c_t, polynomial order p of the discrete solution. The algorithm starts from an initial mesh $\mathcal{M}_0$ and performs `maxIter` adaptation iterations to produce the optimal mesh $\mathcal{M}^*$.

Boundary layer

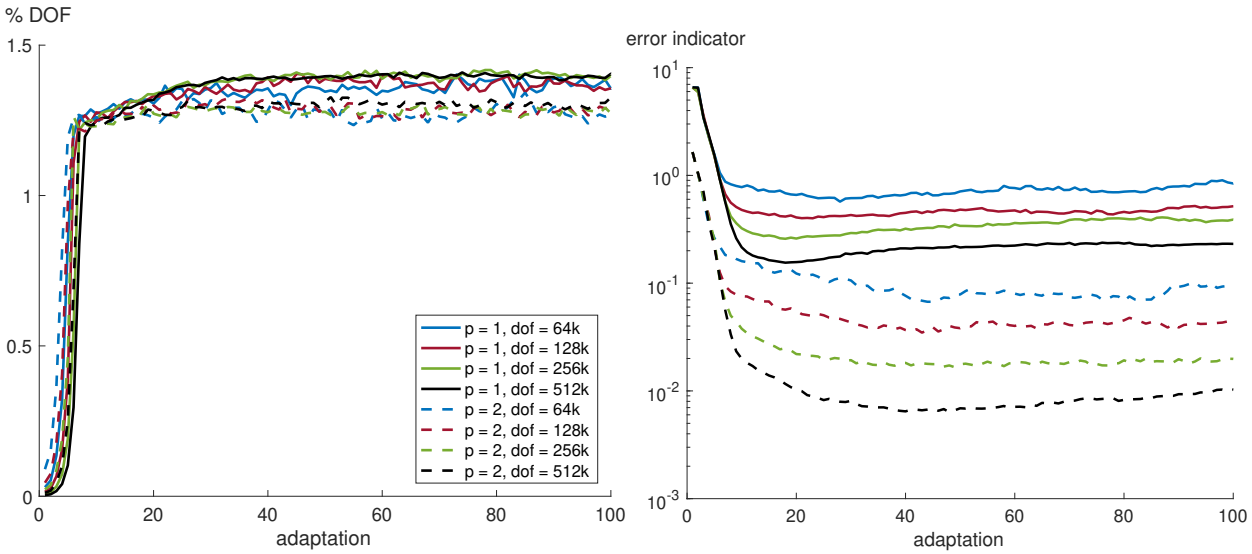
Here, we will study the same function as in Section 4.2. The convective velocity is $\mathbf{c} = (0.5, 0.5, 0.5)^t$ and the viscosity is $\nu = 1$. Figure 4.11 shows that the error indicator successfully decreases at the onset of the adaptation sequence. However, as the adaptations progress, there is a noticeable increase in the error which subsequently appears to level off. Furthermore, the DOF counts are overshot in comparison to the requested values. In contrast to the L^2 error control cases (particularly the spherical wave case) the DOF overshoot is not corrected over the course of the adaptations and remains quite high.

Table 4.6 quantifies the metric conformity statistics along with the fractional overshoot in the expected number of pentatopes (which is the same as the overshoot in the DOF for the current discontinuous Galerkin discretization). Though the fraction of edges within the quasi-unit range is acceptable (between 91-93%), the number of pentatopes of quality greater than 0.8 remains in the the 20-25% range and is highest (30%) for the 64k DOF $p = 2$ discretization. The overshoot in number of pentatopes (equivalently, in the DOF) is quite large, ranging from 29-43%. The mesh appears incapable of producing high quality pentatopes conforming to the input metric field which is possibly caused by a poor incoming metric field.

Nonetheless, the $p = 1$ and $p = 2$ optimized meshes at 512k DOF obtained for this case are shown in Figures 4.12 and 4.13 and exhibit the expected refinement in the boundary layer near the $x = 0$ wall.

The convergence of the global error estimate and, hence, output error over the course of the adaptations (see Figure 4.14) further exhibits the noisy behaviour and makes it difficult to assess whether these quantities exhibit the expected asymptotic convergence rates of h^{2p} [89].

89. Carson *et al.*, *Analysis of Output-Based Error Estimation for Finite Element Methods.* 2017



(a) % DOF **(b)** Error indicator

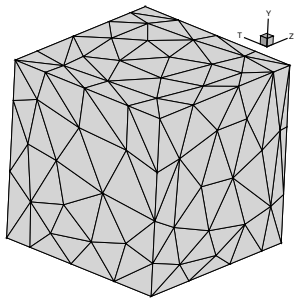
Figure 4.11: DOF fraction and error estimate versus adaptation iteration for the boundary layer advection-diffusion case.

$p = 1$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.48	2.25	1.11	91.34 %	0.16	0.70	21.66 %	17.32k	35.35 %
128k	0.50	2.37	1.10	91.93 %	0.13	0.71	23.81 %	34.80k	35.93 %
256k	0.40	2.91	1.10	91.11 %	0.08	0.69	21.04 %	73.01k	42.60 %
512k	0.42	3.06	1.10	91.23 %	0.07	0.70	23.64 %	143.05k	39.70 %

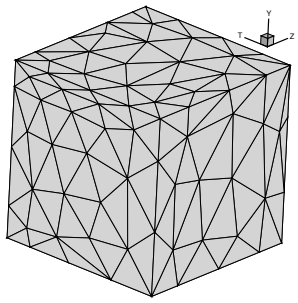
$p = 2$	$\ell_{\min}$	$\ell_{\max}$	ℓ_{avg}	$\%\ell_{\text{unit}}$	$q_{\min}$	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.56	1.94	1.10	92.73 %	0.25	0.73	29.50 %	5.52k	29.42 %
128k	0.42	2.21	1.11	92.08 %	0.19	0.72	24.61 %	11.03k	29.26 %
256k	0.51	2.22	1.10	93.40 %	0.23	0.72	26.54 %	21.96k	28.66 %
512k	0.36	2.39	1.10	92.63 %	0.11	0.72	26.88 %	44.60k	30.65 %

Table 4.6: Metric conformity statistics at the final adaptation iteration for the advection-diffusion boundary layer case with both $p = 1$ (top) and $p = 2$ (bottom) discretizations.

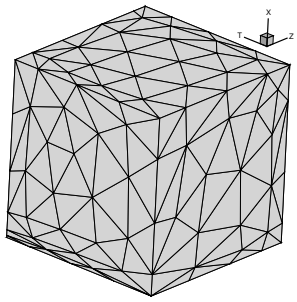
We can, however, assess the convergence of the L^2 error as well as the error indicator with mesh refinement. Figure 4.15 shows the convergence of these quantities for the range of DOF requests studied here, whereby the rates were estimated by performing a linear fit of the error-DOF values from the last two mesh refinement levels, with the last five data points from the adaptation iterations. The error indicator appears to converge faster than the expected h^{2p} rate and the rate for the L^2 error is near the expected h^2 rate for the $p = 1$ discretization and slower than the expected h^3 rate for $p = 2$. It is likely that the DOF range studied here is still pre-asymptotic. It would be worthy to investigate alternative discretizations, such as the continuous Galerkin



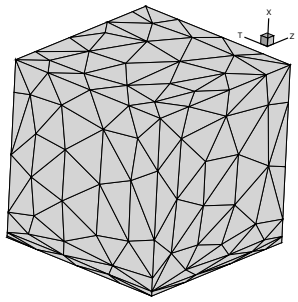
(a) $x = 0$



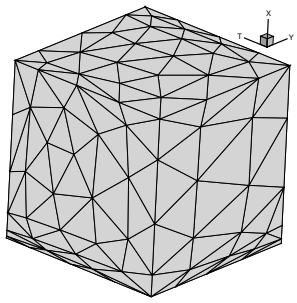
(b) $x = 1$



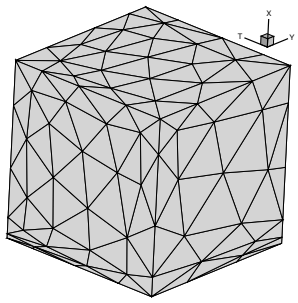
(c) $y = 0$



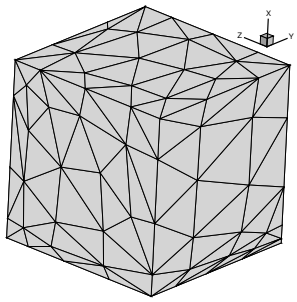
(d) $y = 1$



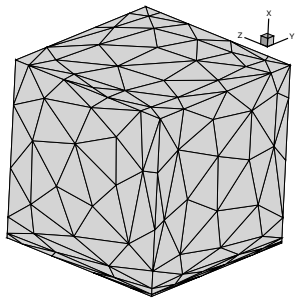
(e) $z = 0$



(f) $z = 1$

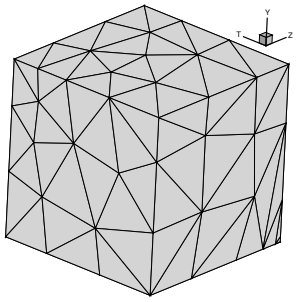


(g) $t = 0$

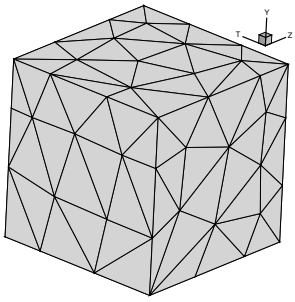


(h) $t = 1$

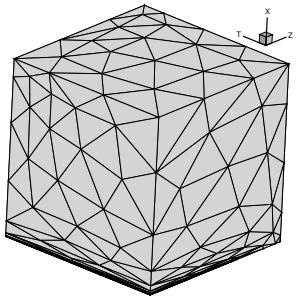
Figure 4.12: Optimized meshes at 512k DOF for the $p = 1$ advection-diffusion boundary layer case.



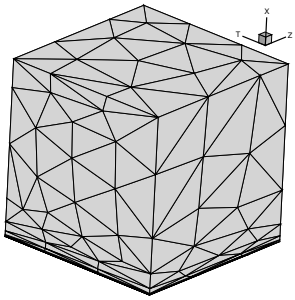
(a) $x = 0$



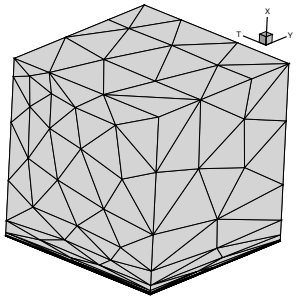
(b) $x = 1$



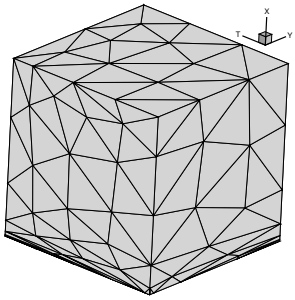
(c) $y = 0$



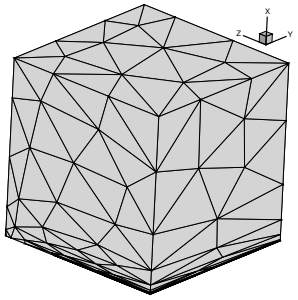
(d) $y = 1$



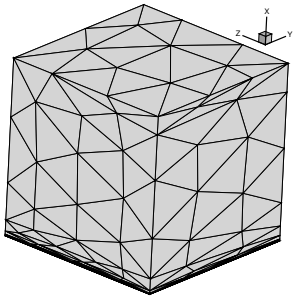
(e) $z = 0$



(f) $z = 1$



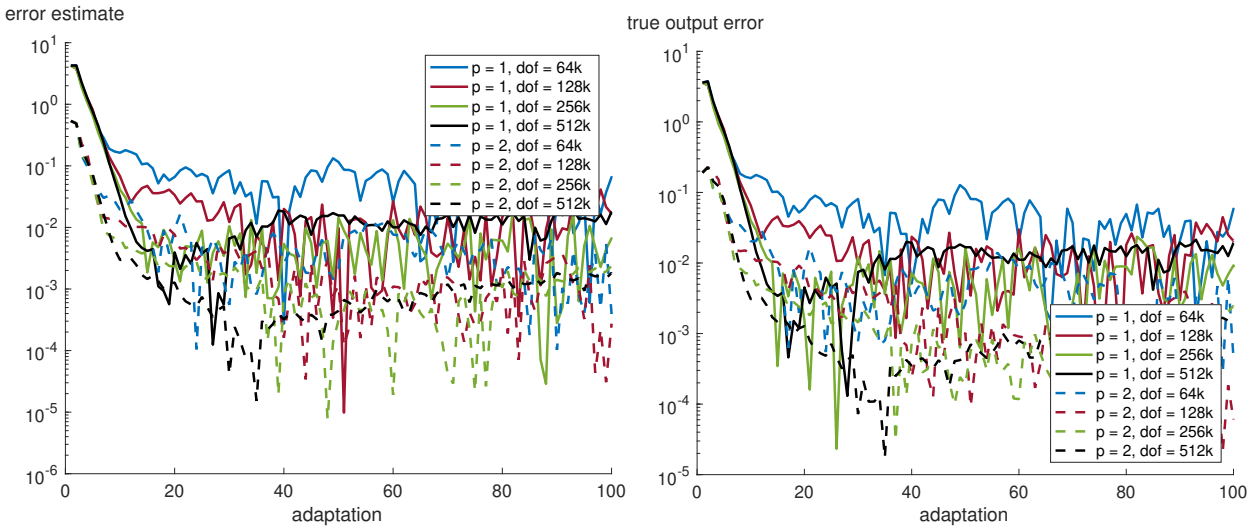
(g) $t = 0$



(h) $t = 1$

Figure 4.13: Optimized meshes at 512k DOF for the $p = 2$ advection-diffusion boundary layer case.

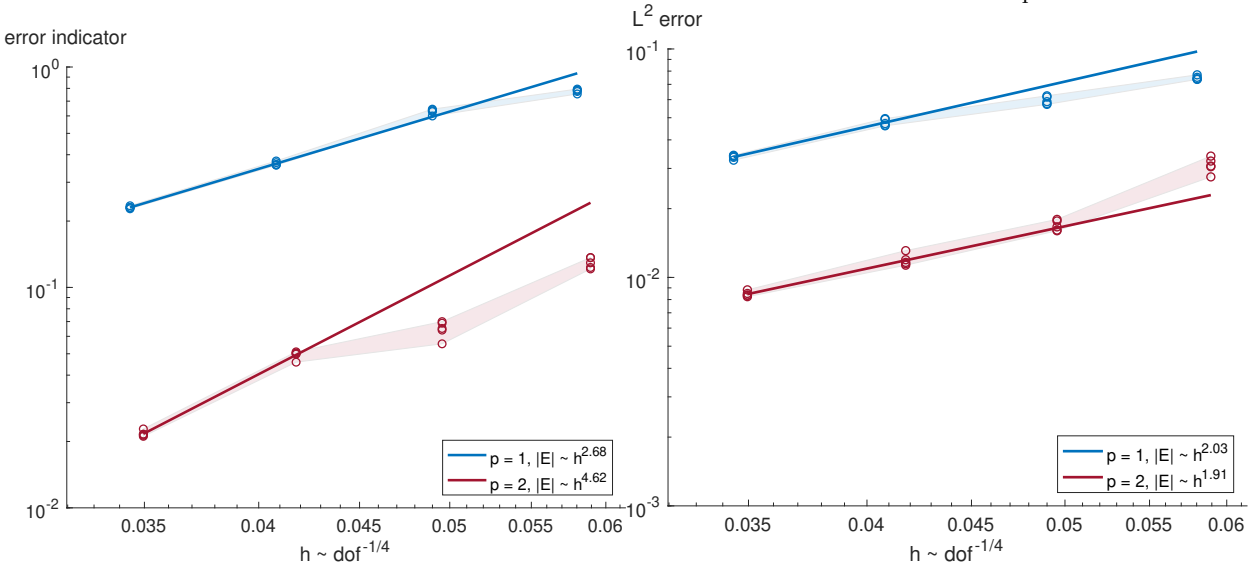
method, which has the advantage of requiring less DOF to reach a certain error level. For the same DOF, however, the continuous Galerkin discretization would produce larger meshes (with more pentatopes) which further motivates the development of the meshing software in a parallel setting.



(a) Error estimate

(b) Output error

Figure 4.14: Error estimate and output error versus adaptation iteration for the



(a) Error indicator

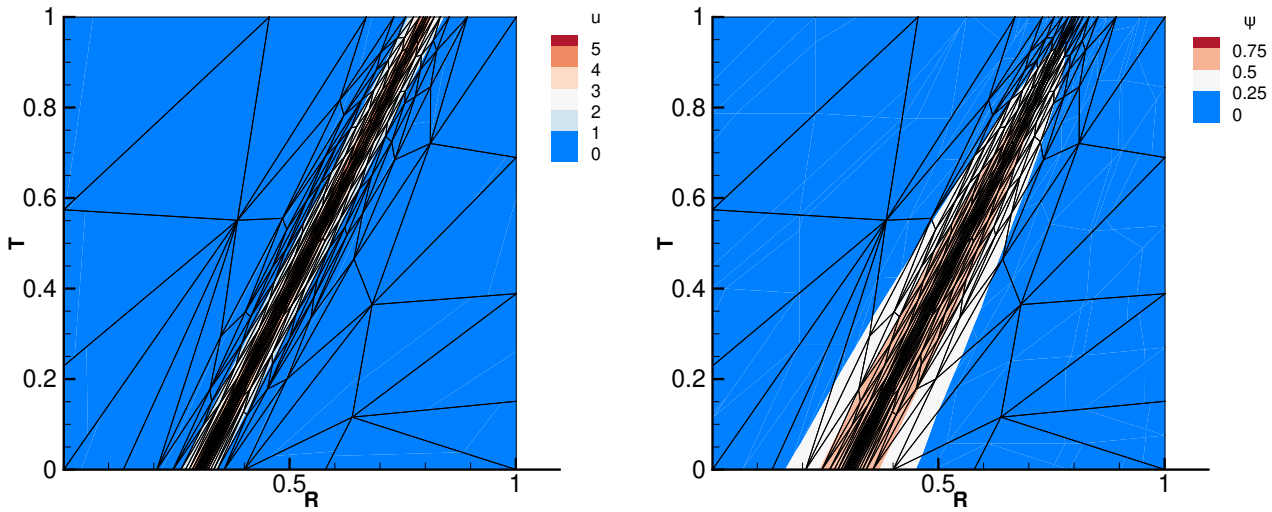
(b) L^2 error

Figure 4.15: Convergence of the error indicator and L^2 solution error with mesh refinement for the boundary layer advection-diffusion case ($p = 1$ and $p = 2$).

Expanding spherical wave

Now, let us revisit the expanding spherical wave case introduced earlier. As in the boundary layer case, we will use the method of man-

ufactured solutions and compute the source term by substituting u for the analytic solution of Equation 4.6. The parameters have been slightly modified from the original L^2 error control case. In particular, the temporal decay rate is $\alpha = 0.1$, the initial wave strength is $k_0 = 5$ and the spatial decay rate is $k_1 = 1000$. The sphere begins at a radius of $r_0 = 0.3$ and propagates with a constant velocity of $v = 0.5$, thus the final sphere (with radius $r_f = 0.8$) should be entirely contained within the tesseract domain. When solving the PDE of Equation 2.24, the convective velocity is $\mathbf{c} = 0.5\mathbf{e}_r$ (where $\mathbf{e}_r$ is the unit vector in the radial direction) and the viscosity is $\nu = 0.01$.



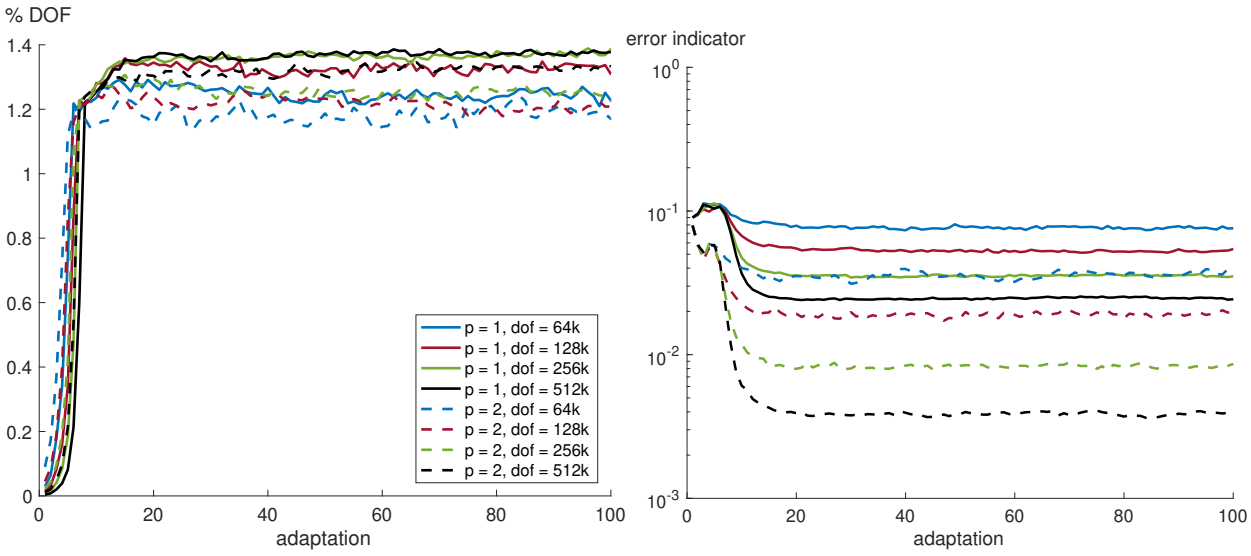
(a) Solution, u

(b) Adjoint, ψ

Figure 4.16: Solution u (left) and adjoint (right) obtained on a $p = 1$ 1000-DOF optimized mesh for the solution of the advection-diffusion equation with MMS (using Equation 4.6) in a $1d + t$ spherical-temporal coordinate system.

For both $p = 1$ and $p = 2$ discretizations, 100 adaptations were run for target DOF requests of 64k, 128k, 256k and 512k. The convergence of the DOF (as a fraction of the request) and error indicator displayed in Figure 4.17 shows that the DOF overshoot ranges from roughly 18-38% and that the error indicator, in contrast to the boundary layer case, exhibits a steadier convergence as the adaptations progress.

Metric conformity at the final adaptation iteration is better for this case, likely resulting from the smoother behaviour of the error over the adaptation iterations, thus producing more realizable metrics for the mesher. The fraction of edges in the quasi-unit range is excellent, lying between 91-98% and the fraction of pentatopes with a quality of at least 0.8 is between 21-28% for the $p = 1$ case and is as high as 39% for the $p = 2$ case.



(a) DOF

(b) Error indicator

Figure 4.17: DOF fraction and error indicator versus adaptation iteration for the spherical wave advection-diffusion case.

$p = 1$	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.52	2.03	1.09	94.83 %	0.26	0.73	28.40 %	16.02k	25.17 %
128k	0.40	2.13	1.10	92.83 %	0.12	0.70	20.28 %	34.42k	34.46 %
256k	0.45	2.28	1.10	91.75 %	0.07	0.70	19.73 %	70.29k	37.29 %
512k	0.39	2.60	1.10	91.39 %	0.05	0.70	20.84 %	141.62k	38.30 %

$p = 2$	ℓ_{min}	ℓ_{max}	ℓ_{avg}	$\%\ell_{\text{unit}}$	q_{min}	q_{avg}	$\%q_{\text{unit}}$	# simplices	% overshoot
64k	0.58	1.79	1.07	97.72 %	0.24	0.76	39.47 %	5.04k	18.10 %
128k	0.54	1.83	1.08	96.14 %	0.29	0.75	36.03 %	10.25k	20.11 %
256k	0.51	1.98	1.08	95.32 %	0.17	0.73	29.12 %	21.71k	27.21 %
512k	0.49	2.18	1.10	93.11 %	0.13	0.71	22.90 %	45.11k	32.17 %

Table 4.7: Metric conformity statistics at the final adaptation iteration for the spherical wave advection-diffusion case with both $p = 1$ (top) and $p = 2$ (bottom) discretizations.

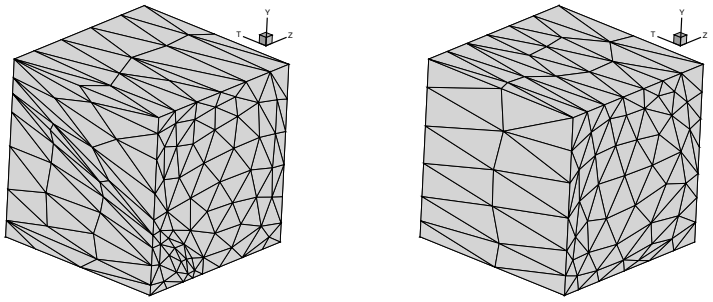
The $p = 1$ and $p = 2$ meshes shown in Figure 4.18 and 4.19 exhibit the expected resolution of the sphere at the temporal slices and the cone along the spatial slices. Note that since the wave only achieves a final radius of $r_f = 0.8$, there is little resolution in the $x = 1$, $y = 1$ or $z = 1$ bounding cubes. Also observe that the width of the wave is roughly one tenth the length of the domain in each direction which is expected given the parameters used to define Equation 4.6. With $k_1 = 1000$ and, again, exploiting the fact that 99.7% of the solution will be contained within three standard deviations of the sphere radius suggests that the wave has a width of $\delta = 3/\sqrt{2k_1} \approx 0.07$. Furthermore, only two elements are needed in the temporal direction.

The error indicator in Figure 4.20(b) converges at a rate of $h^{2.05}$ for the $p = 1$ discretization and $h^{4.02}$ for the $p = 2$ discretization, which is close to the expected h^{2p} rate[89]. However, the convergence of the output in Figure 4.20(a) suggests the solution is still underresolved. The analytic output value for this case was estimated by performing a $1d + t$ (hence, two-dimensional) adaptation sequence in spherical-temporal coordinates in $r \in [0, 1]$ and $t \in [0, 1]$. In three dimensions, the maximum radial coordinate is $r_{\max} = \sqrt{3}$, therefore solving in $1d + t$ in a domain $r \times t = [0, 1] \times [0, 1]$ would not provide an accurate estimate of the output. However, with the current parameters, the function in Equation 4.6 decays very quickly away from the expanding sphere. Therefore, the contribution of the output in Equation 4.10 for $r > 1$ is very small. To quantify this, the adaptation sequence was performed in the $r - t$ domain of $[0, 1] \times [0, 1]$ and also in $[0, \sqrt{3}] \times [0, 1]$. For reference, the primal and adjoint solutions obtained with a $p = 1$ 1000-DOF optimized mesh for the $1d + t$ case are shown in Figure 4.16. After fifty adaptation iterations for $c_t = 20,000$ with $p = 3$, both output values converged to the same result of $\mathcal{J}_{1d+t}(u) \approx 0.28336189$ with an error estimate on the order of 10^{-9}. The resulting output for our problem is then

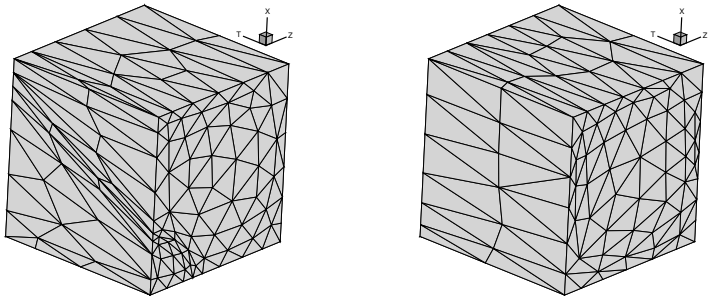
$$\mathcal{J}_{3d+t}(u) = \mathcal{J}_{1d+t}(u) \int_0^{\pi/2} \sin\phi \, \mathrm{d}\phi \int_0^{\pi/2} \mathrm{d}\theta \approx 0.445103816 \tag{4.11}$$

since we are only solving for one eighth of the expanding sphere. The $p = 1$ discretization achieves an output value of 0.355 at 706k DOF whereas the value obtained at 680k DOF for $p = 2$ is 0.369, indicating we are still quite far from resolving this function. Either a higher DOF count should be used, in which a parallel implementation of the metric optimization would be beneficial, or a continuous Galerkin (cG) discretization can be investigated, which would once again benefit from a parallel implementation of the meshing algorithm since the number of pentatopes would grow rapidly with even a moderate DOF count for a cG discretization.

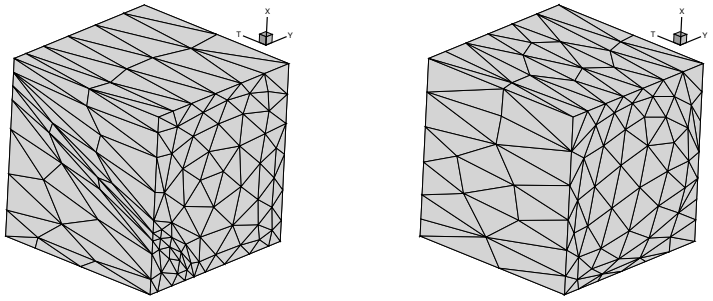
89. Carson *et al.*, *Analysis of Output-Based Error Estimation for Finite Element Methods.* 2017



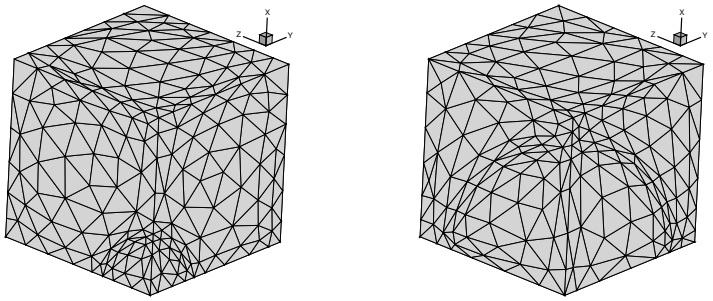
(a) $x = 0$ **(b)** $x = 1$



(c) $y = 0$ **(d)** $y = 1$

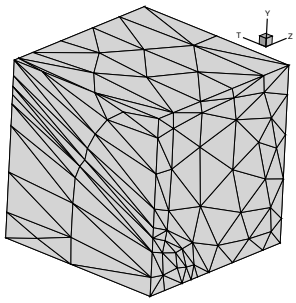


(e) $z = 0$ **(f)** $z = 1$

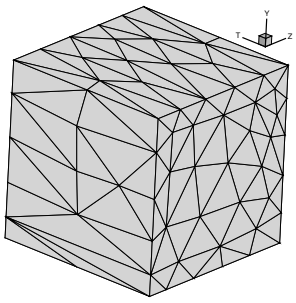


(g) $t = 0$ **(h)** $t = 1$

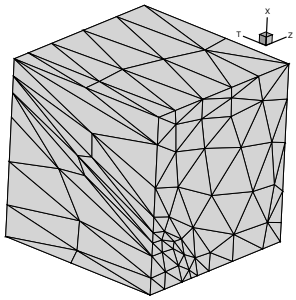
Figure 4.18: Optimized meshes at 512k DOF for the $p = 1$ expanding spherical wave case.



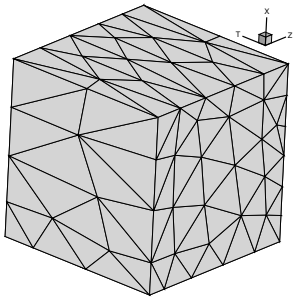
(a) $x = 0$



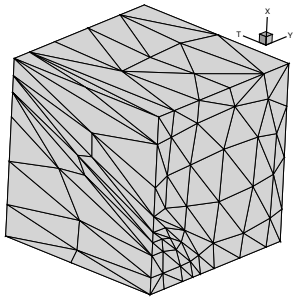
(b) $x = 1$



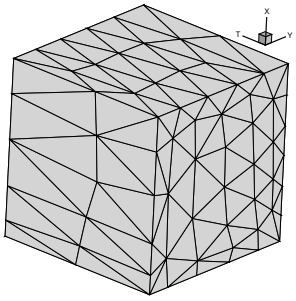
(c) $y = 0$



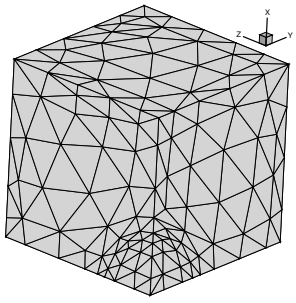
(d) $y = 1$



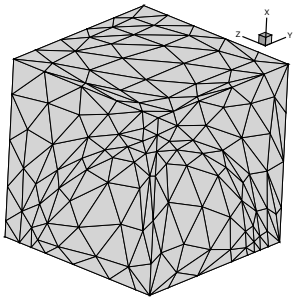
(e) $z = 0$



(f) $z = 1$

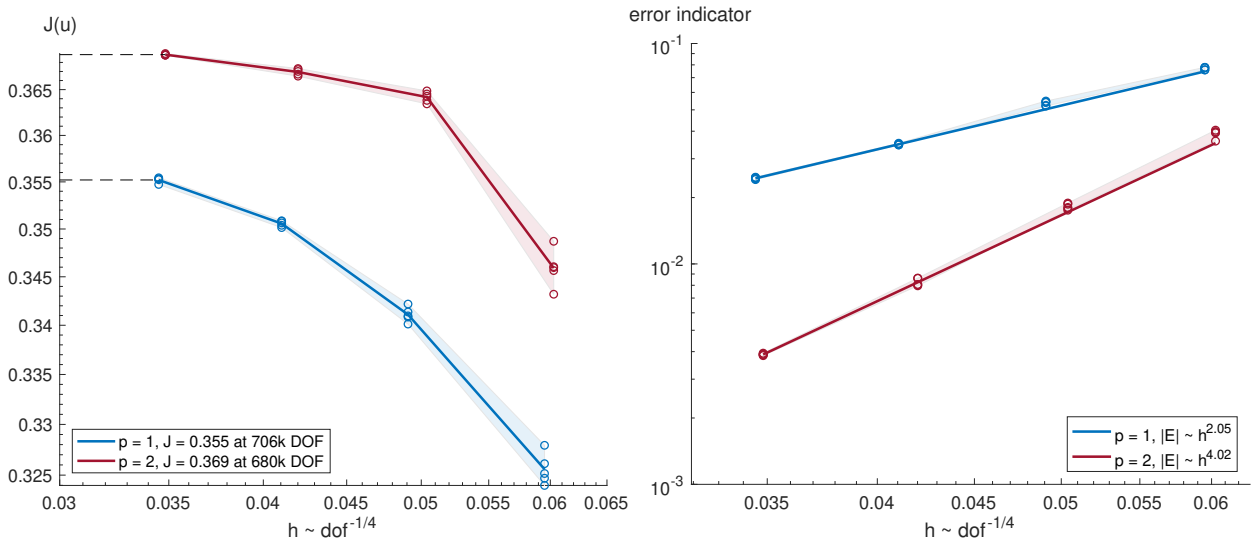


(g) $t = 0$



(h) $t = 1$

Figure 4.19: Optimized meshes at 512k DOF for the $p = 2$ expanding spherical wave case.



(a) Output

(b) Error indicator

Figure 4.20: Convergence of the output and error indicator with mesh refinement for the spherical wave advection-diffusion case ($p = 1$ and $p = 2$).

4.4 *Perspectives*

This chapter demonstrated the first four-dimensional adaptive numerical simulations. By adapting to the exact L^2 error between a prescribed function and its discrete representation in a polynomial basis, the correct mesh size and aspect ratio distributions were obtained for a function with a rapid variation near one of the boundaries of the domain. Similarly, the L^2 error control of a four-dimensional function simulating the expansion of a spherical wave was effective in resolving the strength of the wave as it propagated in time. In particular, the meshes exhibited a significant amount of clustering within 99.7% of the wave radius and metric conformity was very good for this case. The ability of the algorithm to reach this regime was demonstrated for a benign four-dimensional function.

Furthermore, the algorithm effectively refined the same functions when substituting the exact L^2 error with error estimates obtained from the solution of the advection-diffusion equation with the method of manufactured solutions. In the case of the expanding spherical wave, the error converged steadily throughout the adaptation iterations but the computational resources available were insufficient to reach the asymptotic regime. The boundary layer case exhibited a rapid drop in error followed by a slight increase which steadied off as the adaptations proceeded. The metrics obtained from the optimization in Section 2.5 can be improved in the future. Good metric conformity, as in the case of the expanding spherical wave, is indicative of much more realizable metrics obtained from the metric optimization

Property	BL (L^2)	SW (L^2)	BL (PDE)	SW (PDE)
# interior vertices, v_i	4.0k	3.0k	5.0k	4.0k
mean valency, $\bar{\nu}_{v_i}$	115.45	114.61	118.94	112.90
# boundary vertices, v_b	4.0k	5.0k	3.0k	4.0k
mean valency, $\bar{\nu}_{v_b}$	56.27	51.94	59.65	60.22
# edges, e	102.0k	93.0k	99.0k	98.0k
mean valency, $\bar{\nu}_e$	13.43	12.62	14.48	13.85
# triangles, t	312.0k	275.0k	314.0k	305.0k
mean valency, $\bar{\nu}_t$	4.38	4.24	4.56	4.46
# tetrahedra, f	355.0k	308.0k	366.0k	351.0k
mean valency, $\bar{\nu}_f$	1.93	1.90	1.96	1.94

Table 4.8: Valency statistics for the $p = 1$ 512k DOF-optimized meshes in this chapter for the boundary layer (BL) and spherical wave (SW) cases for either the L^2 error control or advection-diffusion (PDE) problems.

Order	$p = 1$		$p = 2$		$p = 3$		$p = 4$	
Case	dG	cG	dG	cG	dG	cG	dG	cG
Boundary Layer (L^2)	683.2k	8.2k	2.0M	110.0k	4.8M	523.5k	9.6M	1.2M
Spherical Wave (L^2)	584.3k	8.0k	1.8M	100.6k	4.1M	468.6k	8.2M	1.1M
Boundary Layer (PDE)	715.2k	7.3k	2.1M	106.2k	5.0M	518.9k	10.0M	1.2M
Spherical Wave (PDE)	679.9k	7.7k	2.0M	105.9k	4.8M	509.1k	9.5M	1.2M

Table 4.9: Cost of the discontinuous (dG) and continuous (cG) discretizations with various polynomial orders p for the meshes optimized at $p = 1$ 512k DOF in this chapter.

procedure. A more steady convergence of the error indicator over the adaptation iterations was seen for this case.

Parallel implementations of both the metric optimization procedure as well as the meshing tool would allow larger and more complex problems to be studied. An investigation into less computationally expensive discretizations, such as the continuous Galerkin method, are also very attractive. This is motivated by the valency statistics of the largest meshes (optimized for $p = 1$ 512k dG DOF) in this chapter (see Table 4.8). Furthermore, the costs of the cG discretization is dramatically lower than the dG one, as shown in Table 4.9.

Chapter 5

Conclusions

Upward, not Northward[90].
— Edwin A. Abbott

5.1 Summary

Contributions

The work in this thesis began with a quest for a four-dimensional meshing algorithm to support mesh-adaptive numerical simulations for the solution of unsteady three-dimensional problems. As such, we demonstrated the first four-dimensional anisotropic meshing capability, founded on a method for performing local mesh modification operators within a dimension-independent cavity framework. First, the design of this mesh modification operator was discussed so as to ensure (1) a valid mesh and (2) a valid geometry discretization are both maintained throughout the adaptation process. Our capability was then demonstrated on benchmark three-dimensional problems. Driven by a gap in the literature in describing and justifying the scheduling of local operators, we then studied how the adaptation components influences metric conformity as well as the target number of simplices. Next, four-dimensional benchmark cases were developed and used to demonstrate the first anisotropic four-dimensional meshing capability.

We then used the meshing tool within the entire adaptive framework to find the optimal mesh for representing a function of four variables. The meshes produced by the algorithm showed the expected sizes and aspect ratios. Finally, we demonstrated the first four-dimensional adaptive numerical simulations, driven by the solution to the advection-diffusion equation, on two types of problems.

Conclusions

Chapter 3 demonstrated the importance of vertex smoothing, limiting the creation of short edges (when inserting vertices) and of dividing the algorithm into two stages with different target split lengths. In particular, metric conformity in terms of quasi-unit edge lengths and simplices was improved when these components were enabled. Furthermore, a factor was introduced to control the metric volume upon inserting a vertex in four dimensions. We determined this factor was important for matching the expected number of pentatopes. Some statistics of the valencies in the produced meshes indicate that the ball of a vertex approaches the geometry of a 120-cell. Information from theses meshes also allowed us to compare the costs of the continuous and discontinuous Galerkin finite element discretizations for the same meshes.

In Chapter 4, the expected mesh size and aspect ratio distributions were met when adapting to the exact L^2 error of a boundary layer-type function. This is a good verification that the various components, ranging from the error sampling and model synthesis in MOESS, to the mesher itself, were working correctly in the setting of an adaptive numerical simulation. Furthermore, an expanding spherical wave was modeled and exhibited the expected mesh refinement. The adaptive algorithm produced meshes with highly stretched elements along the path of the propagating feature in the spacetime domain.

The adaptive advection-diffusion cases revealed that the mesher was still effectively capturing the solution features for both a boundary layer-type function as well as that of an expanding spherical wave though more computational resources are needed for larger simulations. Similar to the L^2 error control setting, the algorithm, when applied to the expanding spherical wave case, determined roughly two elements were needed in the temporal direction.

5.2 *Future work*

Based on the above conclusions, we recommend the following areas of future work.

Improving the metrics from MOESS

The metrics produced by MOESS can be improved in a variety of ways. First, the addition of a volume- or quality-based term to the objective function of Equation 2.23 may be useful. The lack of the aforementioned term in the implied metric calculation is a limitation of the current work. Furthermore, the metric optimization itself can be improved by producing more realizable metrics. In particular, metric

requests may not necessarily align with the boundaries of the domain and can cause issues with metric conformity near these boundaries. It would be useful to either account for this alignment within the metric optimization statement itself, or perform a post-processing of the metric produced by the optimization.

Conversely, many meshing technologies such as `bamg`[29], `EPIC`[50] and `fefloa`[66] pre-process the input metric by either re-aligning it with the domain boundaries or smoothing the components throughout the domain. Whether MOESS post-processes or the mesher pre-processes the target metric, an important contribution would be the investigation of an efficient tool that would eliminate the issues caused by poorly optimized metrics.

29. Hecht, *BAMG: Bidimensional Anisotropic Mesh Generator*. 1998

50. Michal *et al.*, *Anisotropic Mesh Adaptation through Edge Primitive Operations*. 2012

66. Loseille, *Metric-Orthogonal Anisotropic Mesh Generation*. 2014

Improving the heuristic nature of the mesh adaptation algorithm

As noted in Chapter 3, our algorithm for producing a metric-conforming mesh is heuristic. It would be worthwhile to find a single measure of metric-conformity and pose the metric-conforming mesh adaptation problem as a variational statement. The density of vertices, their coordinates, and the mesh topology would all be optimized simultaneously. This single objective should be sensitive to (1) changes in element sizes so as to create a uniform mesh under the metric and (2) the creation of degenerate elements.

This idea is reminiscent of our approach using isometric embeddings and restricted Voronoi diagrams whereby we initially laid out (and fixed) the number of vertices throughout the domain. Next, the vertex coordinates (and mesh topology) were simultaneously optimized by minimizing the centroidal Voronoi tessellation energy. Ultimately, the process is still divided into two stages: (1) optimizing vertex densities and (2) optimizing the topology and coordinates. Due to the discrete nature of the mesh optimization problem, both stages are still important, though the second stage would be much more rigorous than the algorithm presented in Chapter 3.

Parallelization

To explore larger problem sizes, the meshing tool should be extended to perform the adaptation in parallel, primarily in a distributed memory environment. Preliminary work has been done in three dimensions by Digonnet[82] and Loseille[56], whereby mesh partitions are adapted by keeping the boundaries of these partitions fixed. Once the volume partitions have been adapted, the boundaries of the partitions can then be adapted. This should be done such that the mesher works directly with the partitions used by the solver.

82. Digonnet *et al.*, *Massively Parallel Anisotropic Mesh Adaptation*. 2017

56. Loseille *et al.*, *Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes*. 2017

Treatment for complex geometries

Most of the problems studied in this work, with the exception of the simple three-dimensional geometries of Chapter 3, were straight-sided. First, the mesher should be applied to more complex three-dimensional domains; a first step would be to assess the meshing capability on domains described by the Geometry and Mesh Generation Workshop (GMGW)[91]. The aforementioned problem of aligning the metric with the geometry will be important. Furthermore, the issue of interpolating metrics for vertices that lie outside the background mesh will also be important.

91. Chawner *et al.*, *2nd AIAA Geometry and Mesh Generation Workshop*. 2019

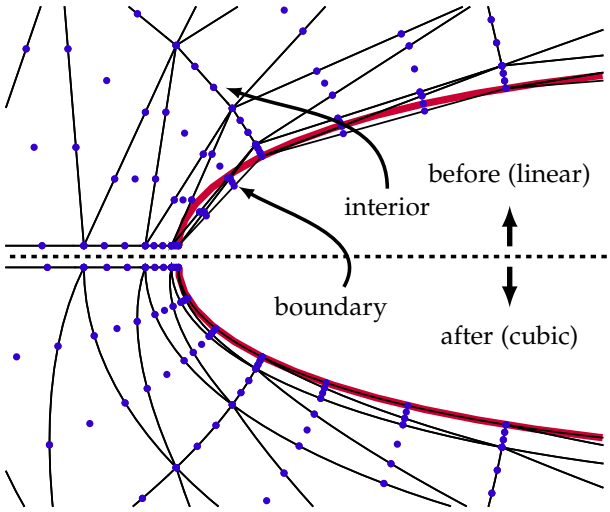


Figure 5.1: Construction of a curvilinear mesh from a straight-sided one.

In order to support high-order adaptive numerical simulations about complex geometries, the faces on the geometry will need to be curved[92]. In the anisotropic setting, the interior mesh will also need to be curved to avoid producing inverted mesh elements (see Figure 5.1).

The state-of-the art in producing curvilinear meshes is either to (1) solve an analogous physics-based problem to compute the location of the interior curvilinear vertices following the influence of the boundary displacements or (2) solve an optimization problem to simultaneously displace both boundary and interior vertices of the curvilinear mesh. Both linear and nonlinear techniques have been investigated in the former[93]. The latter optimization-based techniques have been investigated by the work of Persson[94] and Roca, Girones, Gargallo-Peiró and Sarrate[95–101]. Unfortunately, this technique optimizes a quality measure that is sensitive to the location of the quadrature points within the element and may not be able to guarantee positivity of the Jacobian determinant throughout the curvilinear element. Furthermore, Toulorge et al.[102] have investigated optimization-based ap-

92. Bassi *et al.*, *High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations*. 1997

93. Persson *et al.*, *Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics*. 2009

94. Fortunato *et al.*, *High-order Unstructured Curved Mesh Generation using the Winslow Equations*. 2016

95. Roca *et al.*, *Defining Quality Measures for High-Order Planar Triangles and Curved Mesh Generation*. 2011

96. Ruiz-Gironés *et al.*, *High-Order Mesh Curving by Distortion Minimization with Boundary Nodes Free to Slide on a 3D CAD Representation*. 2015

97. Ruiz-Gironés *et al.*, *Defining an L2-Disparity Measure to Check and Improve the Geometric Accuracy of Non-Interpolating Curved High-Order Meshes*. 2015

98. Ruiz-Gironés *et al.*, *Generation of Curved High-order Meshes with Optimal Quality and Geometric Accuracy*. 2016

99. Ruiz-Gironés *et al.*, *An Augmented Lagrangian Formulation to Impose Boundary Conditions for Distortion-Based Mesh Moving and Curving*. 2017

100. Gargallo-Peiró *et al.*, *A Surface Mesh Smoothing and Untangling Method Independent of the CAD Parameterization*. 2014

101. Gargallo-Peiró *et al.*, *Defining Quality Measures for Validation and Generation of High-Order Tetrahedral Meshes*. 2014

102. Toulorge *et al.*, *Optimizing the Geometrical Accuracy of Curvilinear Meshes*. 2016

proaches that work with a more robust identification of invalid elements[103,104].

More recently, Feuillet has extended the local operator approach to generate quadratic meshes[105] which has also been proposed in the recent work of Coupez[106]. Their results are restricted to the quadratic case and it would be interesting to extend the framework to higher-order curvilinear meshes.

Another interesting area of research would be the investigation of high-order mesh and solution visualization, building off the recent work of Loseille and Feuillet[107].

Finally, the mesh adaptation algorithm naturally handles time-varying three-dimensional geometries and we remark upon some possible methods for constructing these geometry descriptions in Appendix A. The biggest challenge in realizing a simulation within a time-varying domain is the generation of the initial mesh. Here, the initial mesh was easily obtained from the Kuhn-Freudenthal triangulation, however, more complicated geometries would require a geometry-conforming triangulation algorithm. Starting from tetrahedralizations of the domain at various instances in time, either an advancing-front technique or a constrained Delaunay method could provide the initial four-dimensional mesh.

103. Johnen *et al.*, *Geometrical Validity of Curvilinear Finite Elements*. 2013

104. Johnen *et al.*, *Efficient Computation of the Minimum of Shape Quality Measures on Curvilinear Finite Elements*. 2016

105. Feuillet *et al.*, *P2 Mesh Optimization Operators*. 2018

106. Coupez, *On a Basis Framework for High Order Anisotropic Mesh Adaptation*. 2017

107. Loseille *et al.*, *Vizir: High-order Mesh and Solution Visualization using OpenGL 4.0 Graphic Pipeline*. 2018

Applying the four-dimensional framework to other partial differential equations

The focus of this work was to demonstrate the first four-dimensional adaptive numerical simulations for the solution of PDEs. As such, we focused on simple PDEs, particularly, the advection-diffusion equation. An interesting area of future work would be to apply the framework to the solution of the unsteady Navier-Stokes equations.

Investigating other discretizations

Here, we used the discontinuous Galerkin method which is very computationally expensive in the four-dimensional setting. It would be useful to explore alternative discretizations, such as the continuous Galerkin method, so as to achieve a lower solution error with fewer degrees of freedom. Table 5.1 provides the estimated cost per pentatope of the discontinuous and continuous Galerkin methods which were obtained using the valency statistics of the four-dimensional meshes in the previous chapters. However, for the same DOF with the continuous Galerkin discretization, the cost of the mesher increases which further motivates the development of a parallel meshing capability.

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
dG	5	15	35	70	126
cG	0.04	0.71	3.38	10.54	25.71

Table 5.1: Estimated cost per pentatope for the discontinuous (dG) and continuous (cG) Galerkin methods with various polynomial orders.

Mesh adaptation for higher-dimensional parameter spaces

The dimension-independent meshing capability described in this work also opens the possibility of computing optimal meshes for high-dimensional parameter spaces. In particular, the work of Langenhove[108] demonstrated an adaptive framework for controlling the error introduced by systems with a parametric stochastic component which is particularly useful when the output of interest exhibits low regularity in the parameter space. The aforementioned work was a demonstration for two- and three-dimensional parameter spaces, but higher-dimensional ones are certainly possible with our dimension-independent mesher.

108. Langenhove *et al.*, *Goal-Oriented Error Control of Stochastic System Approximations using Metric-Based Anisotropic Adaptations.* 2018

◁ **The extension to $5d$ is trivial.**

Appendix A

Geometry and visualization in four dimensions

4*d*? I can barely visualize 3*d*!
— *David L. Darmofal*

A.1 Background

All the four-dimensional problems studied in this work are contained within the unit tesseract $\mathbf{x} \in [0,1]^4 \in \mathbb{R}^4$. As seen in Chapter 3, this geometry description is a fundamental input to our algorithm. As such, it is important to describe how this tesseract geometry is constructed, not only because it is the geometry we study, but because the developed algorithm can be applied to derive more complicated straight-sided geometries for future four-dimensional (or higher) applications.

Our geometry framework is based on the *Electronic Geometry Aircraft Design System* (EGADS) framework of Haimes[109–111] in which Nodes, Edges and Faces form the fundamental topological structures in a geometry description. These topological entities are additionally associated with geometric ones which, for simple geometries, are easy to describe. Thus, our task is to construct the topology hierarchy for the boundary of the tesseract which will consist of a set of Nodes, Edges, Faces and Volumes. Ultimately, this hierarchy should consist of the following *unique* entities: 16 Nodes, 32 Edges, 24 Faces and 8 Volumes.

Though we are only interested in four-dimensional applications in this work, most of the algorithms we develop are independent of the dimension of the mesh. As such, we describe our method should the

109. Haimes *et al.*, *On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design*. 2012

110. Haimes *et al.*, *The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry*. 2013

111. Haimes *et al.*, *EGADSlite: A Lightweight Geometry Kernel for HPC*. 2018

interested reader find a use in higher dimensions.

Furthermore, this chapter introduces a dimension-independent method for computing the intersection between a polytope and a half-space. This has important applications in the calculation of the restricted Voronoi diagram[47] and we introduce it here since it also bears use in the visualization of four-dimensional meshes. In two dimensions, the Sutherland-Hodgman re-entrant clipping algorithm[112] is commonly used to clip polygons, which is fundamental to the vertex-processing stage of standard graphics pipelines. In three dimensions, Lévy discusses a method for clipping polyhedra[113]. We have not, however, come across a dimension-independent algorithm for clipping polytopes, thus motivating the work in the following sections.

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

112. Sutherland *et al.*, *Reentrant Polygon Clipping*. 1974

113. Lévy, *Robustness and Efficiency of Geometric Programs: The Predicate Construction Kit*. 2016

A.2 A simple result from polytope theory

In order to proceed with the construction of the geometry hierarchy, we need a simple result which is founded on the following definition[114].

114. Henk *et al.*, *Basic Properties of Convex Polytopes*. 2004

Definition 2 (*simple polytope*)**.** *A n-polytope $\mathcal{P}$ is said to be* simple *if every vertex is incident to exactly n facets. A property of simple polytopes is that their dual polytopes are simplices.*

For a tesseract, every vertex is incident to four cubes (or Volumes), thus a tesseract is a simple polytope. The following result forms the foundation of our methods for (1) constructing the geometry of an n-dimensional polytope, and (2) clipping n-polytopes for either visualization purposes or to compute Voronoi diagrams.

Lemma 3. *The edges $\mathcal{E}(\mathcal{P})$ of a simple n-polytope $\mathcal{P}$ can be derived purely from the vertex-facet-incidence matrix of $\mathcal{P}$.*

As Henk suggests[114], vertices of general (even convex) 5-polytopes may be non-adjacent despite sharing many common facets. However, for *simple* polytopes, the dual of $\mathcal{P}$ is a simplex from which the entire set of facets is trivially constructed. Furthermore, the hierarchy of the facets of a polytope can be obtained from the corresponding facet hierarchy of its dual[114], which contains the edges. This result is obviously trivial when finding the edges of a n-polytope with $n \leq 4$ but, as mentioned earlier, we leave the higher dimensional applications for future work.

The edges $\mathcal{E}(\mathcal{P})$ can then be identified from the following relation on $\mathcal{V}(\mathcal{P})$:

$$\mathcal{E}(\mathcal{P}) = \left\{ e = (v_0, v_1) \;\middle|\; |\mathbf{F}(v_0) \cap \mathbf{F}(v_1)| = n - 1 \right\} \tag{A.1}$$

where $\mathbf{F}(\cdot) : \mathbb{N} \to \mathbb{Z}^n$ returns the n-facets adjacent to a particular

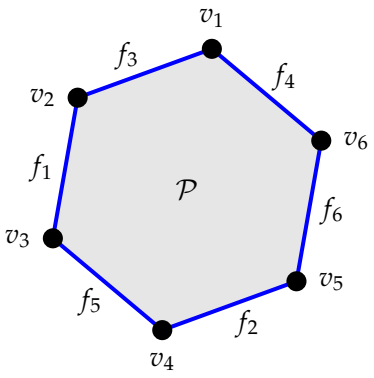


Figure A.1: Vertices and facets (in blue) of an example polygon $\mathcal{P}$. The vertex-facet incidence relations are as follows: $\mathbf{F}(v_1) = (3,4)$, $\mathbf{F}(v_2) = (1,3)$, $\mathbf{F}(v_3) = (1,5)$, $\mathbf{F}(v_4) = (2,5)$, $\mathbf{F}(v_5) = (2,6)$, $\mathbf{F}(v_6) = (4,6)$.

Recall that a relation on the set $\mathcal{V}$ is a subset of $\mathcal{V} \times \mathcal{V}$.

vertex from the vertex-facet incidence matrix. Observe that facets are labeled with integers.

A.3 Constructing the tesseract geometry

The tesseract geometry can be constructed by initializing the vertex-facet incidence matrix, $\mathbf{F}$. Before doing so, it is important to note that we label bisectors with *integers*. For a bisector (facet) B, we will refer to its integer label as $b(B)$ but will often write b for brevity. Each 3-dimensional facet of the tesseract corresponds to a cube and our first task is to build the full set of facets $\mathcal{B}$, which are described by

$$\mathcal{B} = \left\{ B \;\middle|\; b(B) \in \mp[1,2,3,4],\ \mathbf{n}_B^t \mathbf{e}_b = \text{sign}(b)\delta_{bj},\ \forall\, j = 1,\ldots,4 \right\}. \tag{A.2}$$

◁ **In words:**

A bisector B with label $b(B)$ has its normal vector $\mathbf{n}_B$ in the direction of $b\mathbf{e}_b$.

where $\mathbf{e}_b$ is the Cartesian unit vector parallel to the b-axis.

Initially, the 16 Nodes of the tesseract are $\{\mathbf{x} \mid \mathbf{x} \in (\pm 1, \pm 1, \pm 1, \pm 1)\}$ because this makes the identification of the facets $\mathbf{F}(v)$ on each Node v easier. Specifically, this identification is performed by matching the b-th coordinate (recall b is the label of a bisector B) of a given Node with the coordinates of the bisectors in Equation A.2. The Node coordinates are then shifted so the tesseract is in $[0,1]^4$. The resulting vertex-facet incidence matrix is provided in Table A.1.

Node, v	Coordinates, $\mathbf{x}(v)$	Facets, $\mathbf{F}(v)$
1	(1,1,1,1)	1, 2, 3, 4
2	(0,1,1,1)	-1, 2, 3, 4
3	(0,0,1,1)	-1,-2, 3, 4
4	(0,0,0,1)	-1,-2,-3, 4
5	(0,1,0,1)	-1, 2,-3, 4
6	(0,1,1,0)	-1, 2, 3,-4
7	(0,0,1,0)	-1,-2, 3,-4
8	(0,1,0,0)	-1, 2,-3,-4
9	(1,0,0,1)	1,-2, 3, 4
10	(1,0,0,1)	1,-2,-3, 4
11	(1,1,0,0)	1, 2,-3,-4
12	(1,0,1,0)	1,-2, 3,-4
13	(1,0,0,0)	1,-2,-3,-4
14	(1,1,1,0)	1, 2, 3,-4
15	(1,1,0,1)	1, 2,-3, 4
16	(0,0,0,0)	-1,-2,-3,-4

Table A.1: Tesseract Node coordinates and the set of facets $\mathbf{F}(v)$ on each Node where the integer labels correspond to Equation A.2.

The Edges are then identified using Equation A.1 along with the vertex-facet incidence matrix $\mathbf{F}$. To construct the Faces, we need to determine the four (globally unique) Edge children of each Face. Similarly, each Volume entity must reference six (globally unique) Face

children. The procedure for doing so is described in Algorithm A.1. We first loop through the eight bisectors and construct the VRep of the bounding cubes (Volumes) – this is the easy part. With the VRep of each cube, we now obtain the HRep of this cube. The number of bisectors in the HRep should be seven since there are six bounding squares plus the common bisector defining the cube. Next we loop through each of bisector in this HRep (skipping the common one) and construct the VRep of each square. We can then retrieve a previously created square with this VRep, or create a new one. In order to create a square, we need to find the four Edge children which is easily done by looking up the indices of the Edges created earlier. Upon either creating these squares or retrieving them, we now have six squares which can be added as children of the current cube. This procedure is better described in Algorithm A.1.

◁ **An Edge or an edge?**

We make the distinction between a geometry **Edge** and a mesh **edge**. Capital letters will be used throughout this work to represent geometry entities whereas lowercase ones will represent mesh entities.

buildTesseract

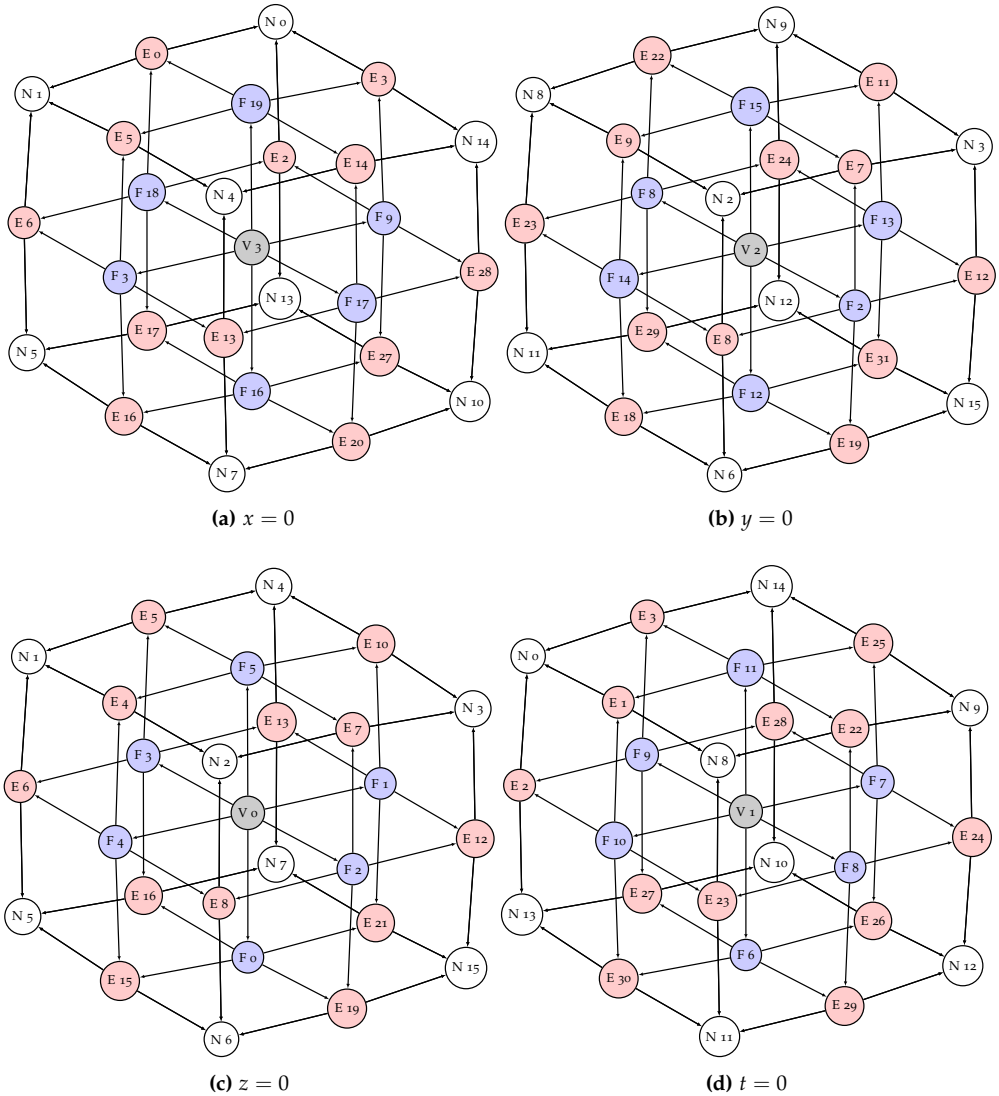
input: tesseract VRep ($\mathcal{P}$), vertex-facet incidence matrix (**F**)
output: tesseract geometry hierarchy, $\mathcal{G}$

```
1  C ← ∅ ▷ Volumes (cubes) to fill
2  for b ∈ B
3      c_b ← VRep(P, b)
4      h_b ← HRep(c_b) ▷ there should be 7
5      S ← ∅ ▷ Faces (squares) to fill
6      for f ∈ h_b ▷ loop through Faces of cube c_b
7          if f = b continue
8          s_f ← VRep(c_b, f)
9          e_f ← HRep(s_f) ▷ there should be 6
10         E ← ∅ ▷ Edges of the Face to fill
11         for e ∈ e_f
12             if e = b or e = f continue
13             E ← E ∪ makeTopology(P(e))
           end
14         S ← S ∪ makeTopology(E)
       end
15     C ← C ∪ makeTopology(F)
   end
16 G ← makeTopology(C)
```

Algorithm A.1: Unique identification of topology hierarchy for the tesseract. The function *makeTopology* is assumed to construct a topological object with a provided set of lower dimensional children. Note that these lower dimensional children may be retrieved if they already exist in the geometry hierarchy. This avoids duplication which would cause issues when traversing the geometry hierarchy during the mesh adaptation procedure.

The procedure in Algorithm A.1 only requires the vertex-facet incidence matrix along with the VRep of the top-level geometry to infer the remaining topology hierarchy. Therefore, it can be used to derive more complicated four-dimensional geometries in the future.

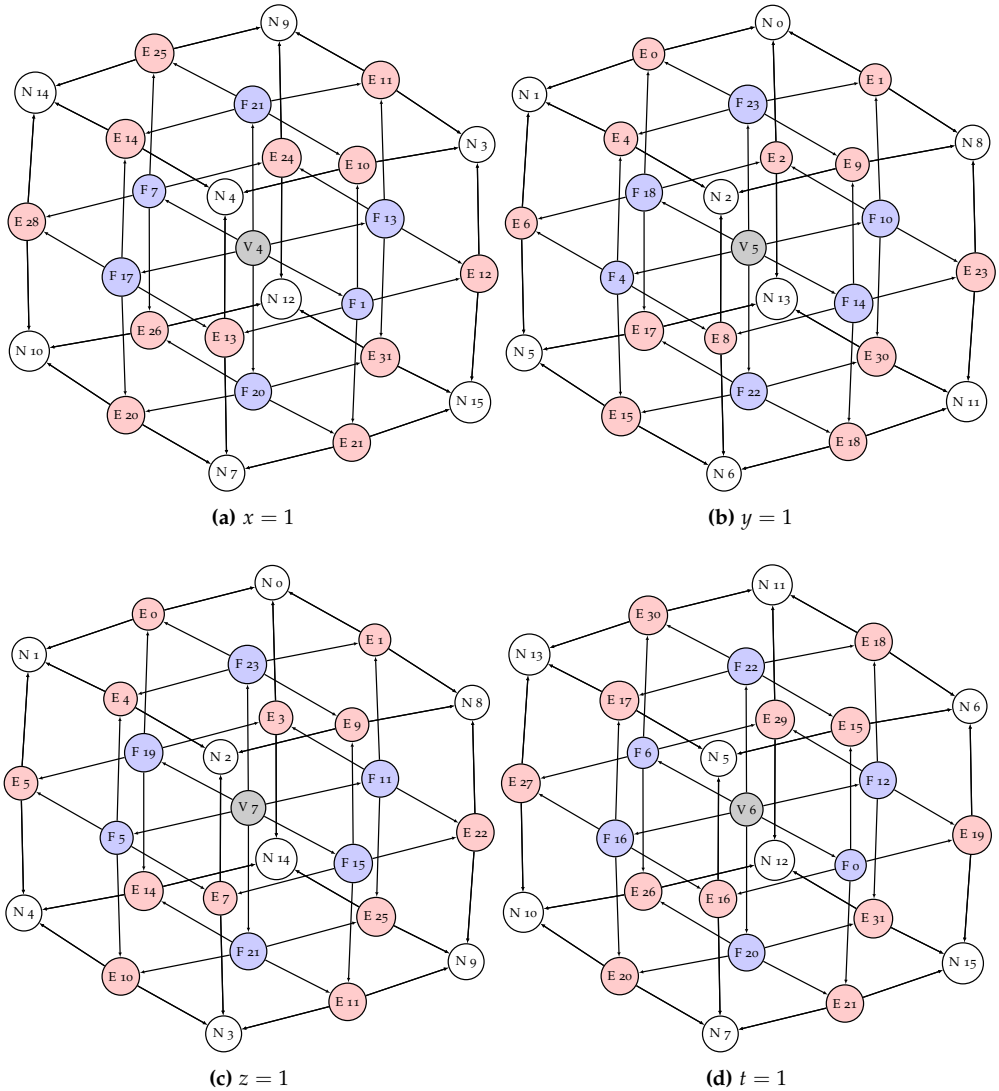
The output of Algorithm A.1 can be represented as a collection of directed graphs for the topologies of each of the eight bounding cubes. These are provided in Figures A.2 and A.3 should the interested reader wish to reproduce the geometry without implementing the aforementioned algorithm.



(a) $x = 0$ **(b)** $y = 0$

(c) $z = 0$ **(d)** $t = 0$

Figure A.2: Representation of the eight bounding cubes of the tesseract geometry as a directed graph with unique geometry entities at the $x = 0$, $y = 0$, $z = 0$ and $t = 0$ hyperplanes. The notation is as follows: V (in gray) refers to a bounding cube (Volume), F (in blue) refers to a square (Face), E refers to a geometry Edge (in red) and N refers to a Node (in white).



(a) $x = 1$ **(b)** $y = 1$

(c) $z = 1$ **(d)** $t = 1$

Figure A.3: Representation of the eight bounding cubes of the tesseract geometry as a directed graph with unique geometry entities at the $x = 1$, $y = 1$, $z = 1$ and $t = 1$ hyperplanes. The notation is as follows: V (in gray) refers to a bounding cube (Volume), F (in blue) refers to a square (Face), E refers to a geometry Edge (in red) and N refers to a Node (in white).

A.4 Visualizing a slice of a four-dimensional mesh

In order to visualize n-dimensional meshes, we require that our polytopal or simplicial meshes can be intersected with a $(n-1)$-dimensional clipping plane such that the intersected polytopes can also be visualized. The following result ensures this and also allows for the calculation of restricted Voronoi diagrams in a dimension-independent manner[47].

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

Proposition 2. *A simple n-polytope $\mathcal{P} \subset \mathbb{R}^n$ intersected with a halfspace $\mathcal{H}^+$ produces a simple polytope.*

◁ **Notation alert:**

$\mathcal{H}$ is used to refer to a hyperplane and the notation $\mathcal{H}^+$ (conversely, $\mathcal{H}^-$) is used to represent the halfspaces on either side of $\mathcal{H}$.

Proof. It suffices to show that every vertex of the new polytope will be incident to exactly n facets. One way of describing the halfspace $\mathcal{H}^+$ is by using a unique point $\mathbf{x}_0$ and normal to the dividing hyperplane $\mathbf{n}$. Since $\mathcal{P}$ is simple, we can determine its edges $\mathcal{E}(\mathcal{P})$. Every vertex of the original polytope $v \in \mathcal{V}(\mathcal{P})$ can then be classified as to whether it is in the halfspace as

$$\mathcal{V}^+(\mathcal{P}) = \left\{ v \in \mathcal{V}(\mathcal{P}) \;\middle|\; (\mathbf{x}(v) - \mathbf{x}_0) \cdot \mathbf{n} > 0 \right\}. \tag{A.3}$$

The vertices created from the intersection are then computed by intersecting each edge with the dividing hyperplane (note we can filter which edges are intersected by finding edges with one vertex in $\mathcal{V}^+(\mathcal{P})$ and one that is not). Denote these intersection vertices as $\mathcal{S}$, therefore, the new vertices of the polytope $\mathcal{Q}$ are $\mathcal{V}(\mathcal{Q}) = \mathcal{V}^+(\mathcal{P}) \cup \mathcal{S}$. There are two cases to consider. First, the vertices $\mathcal{V}^+(\mathcal{P})$ are clearly adjacent to n facets since they were not affected by the intersection. Second, the vertices $\mathcal{S}$ were each created from the intersection of an edge with the hyperplane. Since edges are adjacent to $n-1$ facets (Equation A.1) and the dividing hyperplane, itself, defines a facet of $\mathcal{Q}$, then each intersection vertex is adjacent to n facets. □

The basic method for identifying the visible slices of a n-dimensional mesh first consists of identifying which polytopes (or simplices) of this mesh are cut by the input slice, represented as a $(n-1)$-dimensional hyperplane with point $\mathbf{x}_0$ and normal $\mathbf{n}$. Next, each cut polytope $\mathcal{P}$ must be intersected with the plane. This can be achieved by identifying the set of edges defining a relation on the vertices of the polytope which lie on opposite sides of the hyperplane:

$$\mathcal{E}_s = \left\{ e = (v_0, v_1) \;\middle|\; |((\mathbf{x}(v_0) - \mathbf{x_0}) \cdot \mathbf{n}) \cdot ((\mathbf{x}(v_1) - \mathbf{x_0}) \cdot \mathbf{n}) < 0 \right\} \tag{A.4}$$

where $\mathcal{E}_s$ is, again, a relation on $\mathcal{V}(\mathcal{P})$. The intersection vertices are then determined by intersecting each edge $e \in \mathcal{E}_s$ with the hyperplane. These are tagged with the $n-1$ common bisectors between the two

vertices v_0 and v_1 such that the $(n-1)$-polytope resulting from the intersection can be visualized.

In four dimensions, this hyperplane is specified as a normal vector in either of the four directions (X,Y,Z,T) as well as some offset distance, thus defining the coordinates of $\mathbf{x}_0$. Our framework also has the ability to handle the six rotations about the XY, XZ, XT, YZ, YT and ZT planes but we do not find the dynamic manipulation of these rotations very intuitive from a user perspective. In fact, simply slicing four-dimensional meshes at constant X, Y, Z or T hyperplanes provides an intuitive view of the mesh for our purposes.

◁ **Did you know?**

A rotation in $4d$ is actually about a *plane* the same way a rotation in $3d$ is about an axis. Interesting...

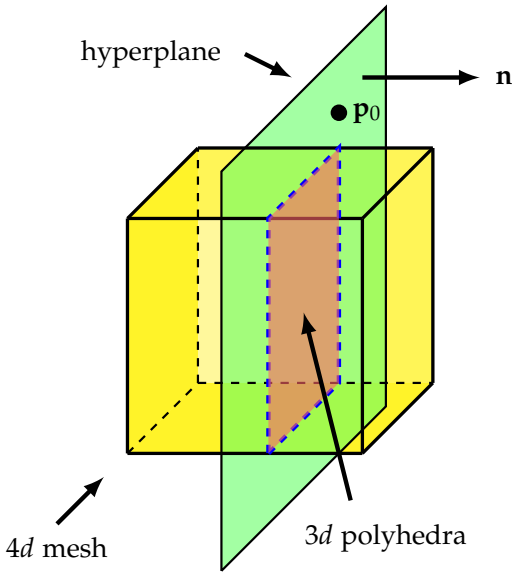


Figure A.4: Schematic of how a $4d$ mesh is sliced to produce polyhedra.

This concept is readily extendable to visualize four-dimensional solution fields. Consider the case of a vertex-valued field attached to the four-dimensional mesh $\mathcal{M}$. When $\mathcal{M}$ is intersected with $\mathcal{H}(\mathbf{n}, \mathbf{p}_0)$ on Line 6 of Algorithm A.2, a linear interpolation between the vertex values at the edge endpoints will yield the value of the field at the intersection vertex. The vertex field can then be passed as usual with the clipped polyhedral mesh to the visualization system.

Examples

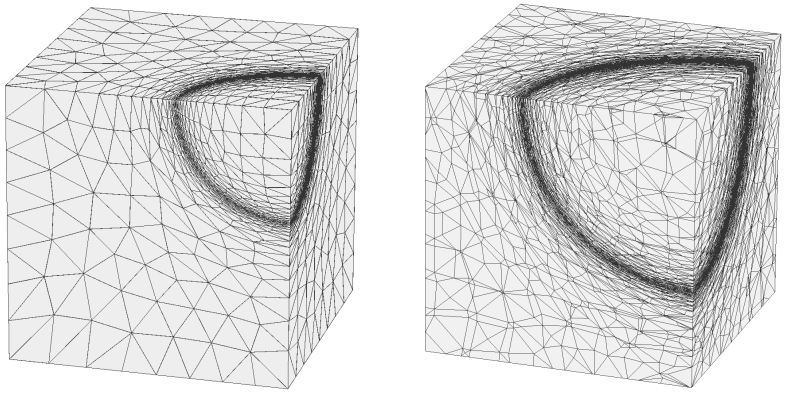
Let us provide some sample visualizations of four-dimensional meshes. The first example is a visualization of a pentatopal mesh, corresponding to the mesh of an expanding sphere. The details of the problem setup are in Chapter 3 but note that the sphere begins with a radius of $R_0 = 0.4$ and expands at constant velocity to a final radius of $R_f = 0.8$ after one second. A visualization of the polyhedral mesh obtained by

visualize4dMesh

input: $\mathcal{M} = (\mathcal{V}, \mathcal{T}) \subset \mathbb{R}^4$, $\mathbf{n}, \mathbf{p}_0 \in \mathbb{R}^4$
output: $\overline{\mathcal{M}} = (\overline{\mathcal{V}}, \overline{\mathcal{T}}) \subset \mathbb{R}^3$
1 **for** $\kappa \in \mathcal{T}$
2 $\quad \mathcal{E} \leftarrow \mathcal{E}_s(\kappa)$ from Equation A.4
3 $\quad$ **if** $\mathcal{E} = \emptyset$ **continue** ▷ κ is not clipped
4 $\quad \mathcal{P} \leftarrow \emptyset$ ▷ initialize polyhedron
5 $\quad$ **for** $e \in \mathcal{E}$
6 $\quad\quad \mathbf{x} \leftarrow e \cap \mathcal{H}(\mathbf{n}, \mathbf{p}_0)$
7 $\quad\quad \mathbf{u} \leftarrow$ **transform**$(\mathbf{x}, \mathcal{H}(\mathbf{n}, \mathbf{p}_0))$
8 $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup |\overline{\mathcal{V}}|$
9 $\quad\quad \overline{\mathcal{V}} \leftarrow \overline{\mathcal{V}} \cup \mathbf{u}$
$\quad$ **end**
10 $\quad \overline{\mathcal{T}} \leftarrow \overline{\mathcal{T}} \cup \mathcal{P}$
end

Algorithm A.2: Visualization of a $4d$ mesh by slicing with a hyperplane. Each element κ is checked for an intersection with the hyperplane $\mathcal{H}(\mathbf{n}, \mathbf{p}_0)$. The intersection points are computed on Line 6 and then transformed into three-dimensional coordinates using an orthonormal basis for the hyperplane on Line 7. These intersection points are labeled according to the current size of the three-dimensional vertices $\overline{\mathcal{V}}$ (Line 8) and added to the list of vertices on Line 9. The resulting three-dimensional polyhedron is added to the topology of the mesh $\overline{\mathcal{T}}$ on Line 10. This three-dimensional mesh can then be visualized with a typical visualization system.

clipping the full pentatopal mesh at a time of 0.6 seconds is shown in Figure A.5.



(a) $t = 0$ **(b)** $t = 0.6$

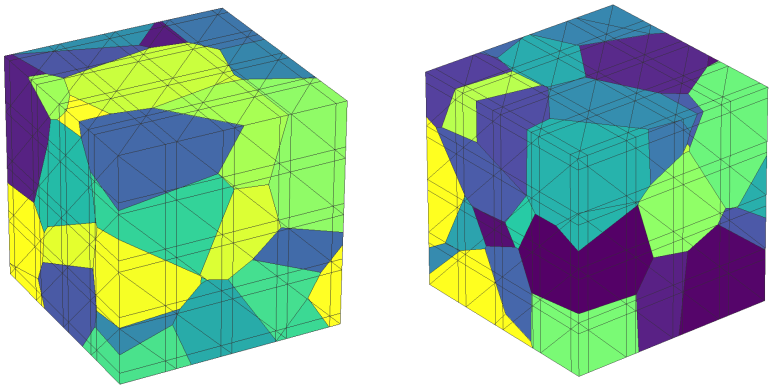
Figure A.5: Visualization of a pentatopal mesh.

Algorithm A.2 also works for four-dimensional polytopal meshes. Here we present two views of a four-dimensional restricted Voronoi diagram which were computed using our algorithm for computing Voronoi diagrams in a dimension-independent manner[47]. The Voronoi diagram of Figure A.6 was computed by randomly placing 100 four-dimensional sites within a four-dimensional Kuhn-Freudenthal triangulation[88] with 4 divisions in each of the x, y, z and t directions. The resulting scenes at $t = 0.3$ and $t = 0.7$ seconds are less than intuitive but we provide them in Figure A.6 for reference. In any case, the

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

88. Kuhn, *Simplicial Approximation of Fixed Points*. 1968

parallel edges of the resulting mesh hint at the divisions in the original Kuhn-Freudenthal triangulation. The Voronoi cell associated with each polyhedron is described by the colours.



(a) $t = 0.3$ **(b)** $t = 0.7$

Figure A.6: Visualization of a $4d$ Voronoi diagram. The colours correspond to the Voronoi cells in which the clipped polyhedra reside.

A.5 Perspectives

The geometry studied here was the unit tesseract. It would be more interesting to study more complicated shapes in the future that include moving geometries. Suppose we have a set of m bodies moving in time, $b_i(t)$, $i = 1, \ldots, m$. We can view the spatiotemporal domain geometry $\Omega(t)$ as the Boolean subtraction of these movies bodies from some outer far-field geometry, $\mathcal{F}(t)$:

$$\Omega(t) = \mathcal{F}(t) - \sum_{i=1}^{m} b_i(t).$$

The advantage of this view of the spatiotemporal geometry is that we can still use a three-dimensional geometry framework, such as the one of Haimes[110], provided the framework exposes a solid Boolean operation between bodies. Otherwise, a more general, representation of the spatiotemporal geometry can be obtained with trivariate B-splines[115].

110. Haimes *et al.*, *The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry*. 2013

115. Cohen *et al.*, *Geometric Modeling with Splines*. 2001

Appendix B

Dimension-independent calculation of the restricted Voronoi diagram

B.1 Background

This chapter discusses the dimension-independent implementation we developed to calculate restricted Voronoi diagrams[47]. The original intent was to develop an anisotropic meshing algorithm based on the concept of isometric embeddings, thus requiring the generation of a uniform mesh in the embedding space. We chose the centroidal Voronoi tessellation to compute this uniform mesh, building upon the work of Lévy[35,43]. Though the work in this thesis deviated from the use of isometric embeddings, we provide the details of how we compute restricted Voronoi diagram in a dimension-independent manner. This chapter also addresses where our friend the giraffe came from.

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

◁ **This is where I came from!**

35. Lévy *et al.*, *Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration*. 2012

43. Lévy, *"Geogram: A Programming Library of Geometric Algorithms"*. 2016

B.2 Centroidal Voronoi tessellations

Given a set of m points (or *sites*), $\mathcal{U} = \{ \mathbf{u}_i \,|\, \mathbf{u}_i \in \mathbb{R}^N \}_{i=1,\dots,m}$, the *Voronoi diagram* of these sites is the polytopal mesh defined by

$$\mathrm{Vor}(\mathcal{U}) = \{ \mathrm{vor}(\mathbf{u}_i) \mid i = 1, \dots, m \} \tag{B.1}$$

where the Voronoi cells $\mathrm{vor}(\mathbf{u}_i)$ are N-polytopes:

$$\mathrm{vor}(\mathbf{u}_i) = \{ \mathbf{x} \in \mathbb{R}^N \mid ||\mathbf{x} - \mathbf{u}_i|| < ||\mathbf{x} - \mathbf{u}_j||, \forall j \neq i \}. \tag{B.2}$$

Now, given a mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T}) \subset \mathbb{R}^N$ and a set of m sites $\mathcal{U}$, we can compute the *restricted* Voronoi diagram

$$\mathrm{Vor}_{\mathcal{M}}(\mathcal{U}) = \mathrm{Vor}(\mathcal{U}) \cap \mathcal{M}, \tag{B.3}$$

which is the intersection of the Voronoi diagram with the mesh $\mathcal{M}$. Although $\mathrm{Vor}(\mathcal{U})$ is a mesh consisting of N-polytopes (possibly unbounded), $\mathrm{Vor}_{\mathcal{M}}(\mathcal{U})$ is a mesh consisting of n-polytopes, assuming the dimension of the topology of $\mathcal{M}$ is $\dim(\mathcal{T}) = n$. $\mathcal{M}$ may be simplicial or polytopal.

We can optimize the sites $\mathcal{U}$ to minimize the *centroidal Voronoi tessellation energy*

$$\mathbb{E}(\mathcal{U};\mathcal{M}) = \sum_{i=1}^{m} \int_{\mathrm{vor}_{\mathcal{M}}(\mathbf{u}_i)} ||\mathbf{x} - \mathbf{u}_i||^2 \, \mathrm{d}\mathbf{x}, \tag{B.4}$$

where $\mathrm{vor}_{\mathcal{M}}(\mathbf{u}_i)$ denotes the Voronoi cell generated by site $\mathbf{u}_i$ restricted to the mesh $\mathcal{M}$. Minimizing the CVT energy can be done in a variety of ways. Lévy and Liu[116] investigate a quasi-Newton approach, but a simpler approach is to use Lloyd relaxation[117]. This consists of iteratively placing a site at the centroid of its restricted Voronoi cell. There are deterministic and probabilistic methods for computing the centroids of the restricted Voronoi cells[118–121]. We prefer a deterministic approach, thus requiring the exact calculation of the centroids of every restricted Voronoi cell.

Every restricted Voronoi cell in Equation B.4 is the intersection of the full Voronoi cell $\mathrm{vor}(\mathbf{u}_i) \subset \mathbb{R}^N$ with the mesh $\mathcal{M}$:

$$\mathrm{vor}_{\mathcal{M}}(\mathbf{u}_i) = \mathrm{vor}(\mathbf{u}_i) \cap \mathcal{M} = \bigcup_{\boldsymbol{\kappa} \in \mathcal{M}} \mathrm{vor}(\mathbf{u}_i) \cap \boldsymbol{\kappa}. \tag{B.5}$$

Instead of computing the intersection of each Voronoi cell with the mesh, we can compute the intersection of each mesh element with the entire Voronoi diagram. Without loss of generality, consider the case of a simplicial mesh. This *restricted Voronoi simplex* (RVS) $\mathrm{Vor}_{\boldsymbol{\kappa}}(\mathcal{U})$ can be defined as

$$\mathrm{Vor}_{\boldsymbol{\kappa}}(\mathcal{U}) = \bigcup_{i=1}^{m} \boldsymbol{\kappa} \cap \mathrm{vor}(\mathbf{u}_i). \tag{B.6}$$

◁ vor **or** Vor**?**

We use the notation "vor" to denote Voronoi *cells* whereas we use "Vor" to denote the Voronoi *diagram*.

116. Liu *et al.*, *On Centroidal Voronoi Tessellation: Energy Smoothness and Fast Computation.* 2009

117. Lloyd, *Least Squares Quantization in PCM.* 1982

118. Du *et al.*, *Centroidal Voronoi Tessellations: Applications and Algorithms.* 1999

119. Du *et al.*, *Grid Generation and Optimization Based on Centroidal Voronoi Tessellations.* 2002

120. Du *et al.*, *Tetrahedral Mesh Generation and Optimization Gased on Centroidal Voronoi Tessellations.* 2003

121. Du *et al.*, *Anisotropic Centroidal Voronoi Tessellations and their Applications.* 2005

B.3 Computing restricted Voronoi simplices

The input to the calculation of a restricted Voronoi simplex is a simplex of the mesh $\boldsymbol{\kappa}$ along with the sites $\mathcal{U}$. To perform the calculation exactly, we use the dimension-independent `side` predicates of Lévy[43]. As such, we need to determine whether an intersection point is the intersection of a hyperplane with an edge, triangle, tetrahdron, pentatope, etc.

For example, consider the intersection of a triangle $\boldsymbol{\kappa} = (\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2)$ with the Voronoi cell defined by the site $\mathbf{u}_i$: $\mathrm{vor}(\mathbf{u}_i)$. This Voronoi cell can be viewed as the intersection of all the $m-1$ halfspaces defined by

43. Lévy, *"Geogram: A Programming Library of Geometric Algorithms"*. 2016

◁ **Hyperplane?**

We consider the general setting, in which the triangle can be embedded in a dimension greater than two.

every pair of sites $(\mathbf{u}_i, \mathbf{u}_j)$ $(i \neq j)$:

$$\text{vor}(\mathbf{u}_i) = \bigcap_{j=1}^{m} \mathcal{H}^+(\mathbf{u}_i, \mathbf{u}_j), \quad j \neq i \tag{B.7}$$

Instead of inefficiently performing all intersections in Equation B.7, the security radius theorem[35] gives a bound for which bisectors can be classified as *contributing* bisectors. To apply the security radius theorem, we need the vertices of the current polytope being clipped, which requires a method for intersecting a hyperplane with a polytope.

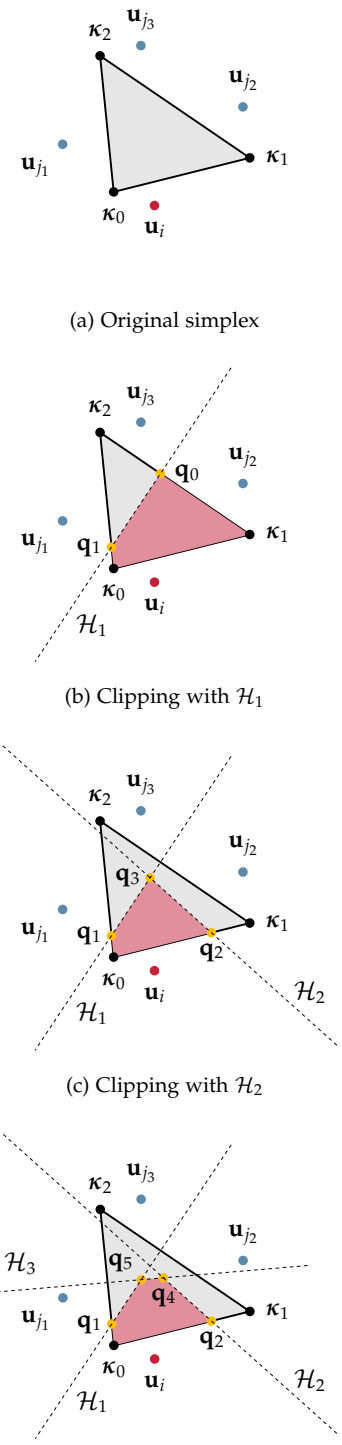
Let us begin with the closest $\mathbf{u}_j$ to $\mathbf{u}_i$ since this pair will define a contributing bisector, which we will denote as $\mathcal{H}_1$. Our goal is to compute the portion of $(\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2)$ closest to $\mathbf{u}_i$ by intersecting it with $\mathcal{H}_1$ (Figure B.1(a)). First, we classify $\boldsymbol{\kappa}_0$, $\boldsymbol{\kappa}_1$ and $\boldsymbol{\kappa}_2$ with respect to $\mathcal{H}_1$. This can be done exactly with the `side_0` predicate, revealing that both $\boldsymbol{\kappa}_0$ and $\boldsymbol{\kappa}_1$ are in the halfspace $\mathcal{H}_1^+$ but $\boldsymbol{\kappa}_2$ is in $\mathcal{H}_1^-$. Therefore, we know that the two edges $(\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2)$ and $e_1 = (\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_2)$ will be intersected with $\mathcal{H}_1$, yielding the intersection points $\mathbf{q}_0$ and $\mathbf{q}_1$, respectively.

Now, we need to clip the new polygon $\mathcal{P}_1 = \text{conv}(\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_1, \mathbf{q}_0, \mathbf{q}_1)$ with the next contributing bisector, $\mathcal{H}_2$ (Figure B.1(b)). To do so, we can classify whether the vertices of $\mathcal{P}_1$ are on $\mathcal{H}_2^+$ or $\mathcal{H}_2^-$ and then identify which edges of $\mathcal{P}_1$ will be clipped by $\mathcal{H}_2$. Using `side_0` on the vertices $\mathbf{q}_0$ or $\mathbf{q}_1$ would result in an inexact calculation since their coordinates were computed with finite-precision arithmetic via the intersection of the edges with $\mathcal{H}_1$. However, since we know that $\mathbf{q}_0$ is the intersection of the edge $(\boldsymbol{\kappa}_1, \boldsymbol{\kappa}_2)$ with $\mathcal{H}_1$, then we can use the `side_1` predicate to classify its side relative to $\mathcal{H}_2$. The same can be done for $\mathbf{q}_1$, knowing it resulted from the intersection of the edge $(\boldsymbol{\kappa}_0, \boldsymbol{\kappa}_2)$ with $\mathcal{H}_1$. The resulting intersection points are $\mathbf{q}_2$ and $\mathbf{q}_3$.

The next polygon in the sequence is determined by $\mathcal{P}_2 = \text{conv}(\boldsymbol{\kappa}_0, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_1)$ and, again, we need to classify each vertex with respect to the next bisector $\mathcal{H}_3$. The vertex $\boldsymbol{\kappa}_0$ can be classified using `side_0`; both $\mathbf{q}_1$ and $\mathbf{q}_2$ and p_1 can be classified using `side_1`. However, $\mathbf{q}_3$ is the intersection of the entire triangle $\boldsymbol{\kappa}$ with $\mathcal{H}_1$ *and* $\mathcal{H}_2$. To classify $\mathbf{q}_3$ with respect to $\mathcal{H}_3$, we need to use the `side_2` predicate.

By extracting the edges of $\mathcal{P}_2$ and intersecting each edge with $\mathcal{H}_3$ results in the Voronoi polygon $\mathcal{P}_3 = \text{conv}(\boldsymbol{\kappa}_0, \mathbf{q}_2, \mathbf{q}_4, \mathbf{q}_5, \mathbf{q}_1)$, which forms the final clipped Voronoi polytope (Figure B.1(d)).

To generalize this concept, consider a vertex q computed from the intersection of an edge $e = (e_0, e_1)$ with a bisector $\mathcal{H}$. If e_0 was created from the intersection of a i-simplex $(i \leq n)$ τ_{e_0} of $\mathcal{M}$ and e_1 was created from the intersection of a j-simplex $(j \leq n)$ τ_{e_1} of $\mathcal{M}$, then q is the intersection of the k-simplex $(k \geq \max(i, j), k \leq n)$ $\tau_q = \tau_{e_0} \cup \tau_{e_1}$. We therefore store the symbolic information as to which simplex τ (of the mesh $\mathcal{M}$) produced the intersection. Note that we also need to retain



(a) Original simplex

(b) Clipping with $\mathcal{H}_1$

(c) Clipping with $\mathcal{H}_2$

(d) Clipping with $\mathcal{H}_3$

Figure B.1: Clipping a simplex by a Voronoi cell create by the site $\mathbf{u}_i$.

the k sites that form contributing bisectors with the site $\mathbf{u}_i$.

Algorithm B.1 summarizes this clipping procedure for a single Voronoi polytope. Once the entire restricted Voronoi simplex has been computed (using Algorithm B.1 with all Voronoi cells forming a non-empty intersection with $\boldsymbol{\kappa}$), then the centroids $\mathbf{c}(\text{vor}_{\mathcal{M}}(\mathbf{u}_i))$ of the restricted Voronoi cells can be computed:

$$\mathbf{c}(\text{vor}_{\mathcal{M}}(\mathbf{u}_i)) = \frac{\sum_{\mathcal{P}} \mathbf{c}(\mathcal{P}) v(\mathcal{P})}{\sum_{\mathcal{P}} v(\mathcal{P})}, \quad \text{for } \mathcal{P} = \boldsymbol{\kappa} \cap \text{vor}_{\boldsymbol{\kappa}}(\mathbf{u}_i) \neq \emptyset, \ \forall \boldsymbol{\kappa} \in \mathcal{M} \tag{B.8}$$

where $\mathbf{c}(\mathcal{P})$ retrieves the centroid of the input polytope and $v(\mathcal{P})$ computes the volume. With Lloyd relaxation, the sites are iteratively moved to the centroids computed from Equation. B.8. Upon each iteration, we can compute the energy from Equation B.4.

35. Lévy *et al.*, *Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration*. 2012

computeRestrictedVoronoiPolytope

input: $\mathbf{u}_i$, $\boldsymbol{\kappa}$
output: $\mathcal{P} = \text{vor}(\mathbf{u}_i) \cap \boldsymbol{\kappa}$

1 $\mathcal{P} \leftarrow \kappa$
2 $j = 1$
3 **while** (security radius not reached)
4 $\quad \mathbf{u}_j = \textbf{nearestNeighbour}(\mathbf{u}_i, j)$
5 $\quad \mathcal{H} \leftarrow \mathcal{H}(\mathbf{u}_i, \mathbf{u}_j)$
6 $\quad \mathcal{Q} \leftarrow \emptyset$
7 $\quad \mathcal{E} \leftarrow \mathcal{E}(\mathcal{P})$
8 $\quad$ **for** $e = (e_0, e_1) \in \mathcal{E}$
9 $\quad\quad s_0 = \texttt{side\_I}(e_0, \mathcal{H})$ ▷ for the I-simplex τ_{e_0}
10 $\quad\quad s_1 = \texttt{side\_J}(e_1, \mathcal{H})$ ▷ for the J-simplex τ_{e_1}
11 $\quad\quad$ **if** $s_0 = s_1$ ▷ no intersection of e with $\mathcal{H}$
12 $\quad\quad\quad$ **continue**
13 $\quad\quad p \leftarrow e_0$ **or** e_1 (whichever has side s in $\mathcal{H}^+$)
14 $\quad\quad \mathbf{q} \leftarrow e \cap \mathcal{H}$ and set $q \leftarrow |\mathcal{P}|$
15 $\quad\quad \tau_q \leftarrow \tau_{e_0} \cup \tau_{e_1}$
16 $\quad\quad \mathbf{F}(q) \leftarrow \mathbf{F}(e_0) \cap \mathbf{F}(e_1) \cup b(\mathcal{H})$
17 $\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup p \cup q$
$\quad$ **end**
18 $\quad \mathcal{P} \leftarrow \mathcal{Q}$
19 $\quad j = j + 1$ ▷ proceed to next nearest neighbour
end

Algorithm B.1: Calculation of a restricted Voronoi polytope for a Voronoi cell defined by the site $\mathbf{u}_i$ clipped with a simplex $\boldsymbol{\kappa}$ in the mesh $\mathcal{M}$. Line 15 computes the symbolic information as to which k-simplex τ_q of the mesh $\mathcal{M}$ contributes to the calculation of the intersection point q. Line 16 determines which facets are attached to the new intersection vertex q. Note that this is simply the list of facets attached to the edge as well as the facet produced by the bisector $\mathcal{H}$ (labeled with $b(\mathcal{H})$ as in Appendix A). The new polytope $\mathcal{Q}$ is formed by appending whichever vertex p of the edge e is on the side $\mathcal{H}^+$ along with the new intersection vertex q.

To demonstrate the algorithm, consider the optimization of a set of

sites in a n-cube $[0,1]^n$ where the background mesh $\mathcal{M}$ is the Kuhn-Freudenthal triangulation[88] with a single division in each coordinate direction (i.e., there are 2 triangles in $2d$, 6 tetrahedra in $3d$ and 24 pentatopes in $4d$). For these cases, 40 Voronoi sites were initially seeded randomly within $[0,0.1]^n$. Furthermore, consider the mesh of a giraffe with 102,867 vertices and 124,153 triangles embedded in $\mathbb{R}^3$. For this case, 0.4% of the mesh vertices were randomly selected as the sites (this makes the Voronoi cells larger so that they more realistically represent the patches of a giraffe).

The convergence of the CVT energy over 500 iterations of Lloyd relaxation is shown in Figure B.2. The optimized RVD of the final giraffe looks much closer to the patterns on a realistic giraffe than in its initial RVD.

88. Kuhn, *Simplicial Approximation of Fixed Points*. 1968

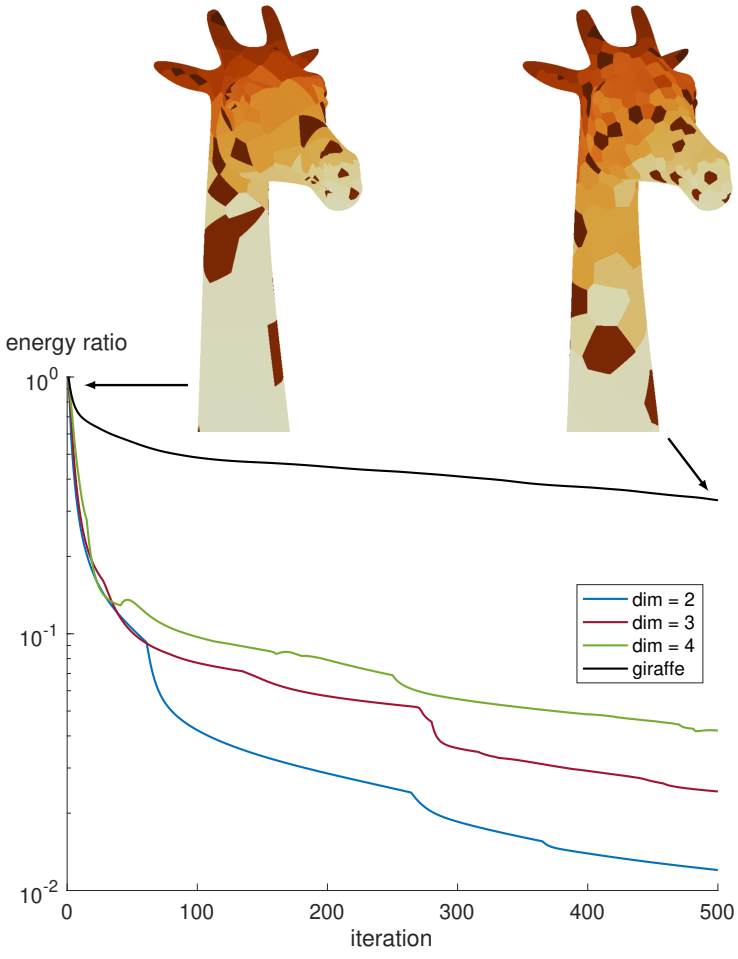


Figure B.2: Convergence of the CVT energy of Equation B.4 (normalized by the initial energy) versus iteration of Lloyd relaxation for the optimization of 40 Voronoi sites in $[0,1]^n$ (n = 2,3,4) as well as for approximately 200 sites on the mesh of a giraffe.

B.4 Perspectives

The calculation of the restricted Voronoi diagram is embarassingly parallel and currently implemented using C++11, OpenMP, PTHREAD and `EMP` (provided with `EGADS`[111]) threads. Future work involves porting this algorithm to graphics processing units (GPUs). Furthermore, the optimization of the sites would benefit from a quasi-Newton approach[116]. Finally, methods for conforming to an input description of the geometry should continue to be investigated[47].

111. Haimes *et al.*, *EGADSlite: A Lightweight Geometry Kernel for HPC*. 2018

116. Liu *et al.*, *On Centroidal Voronoi Tessellation: Energy Smoothness and Fast Computation*. 2009

47. Caplan *et al.*, *Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings*. 2017

Appendix C

Software implementation notes

> Did you write a unit test for that?
>
> — *Marshall C. Galbraith*

This chapter remarks upon some important points in both the `avro` and `SANS` implementations that went into this thesis.

C.1 `avro`

Geometric predicates

The use of exact geometric predicates in a mesh generation tool is essential to perform robust intersection or volume calculations[122]. Shewchuk provides robust geometric predicates for the exact calculation of triangle and tetrahedron volumes through the `orient2d` and `orient3d` predicates. In Chapter 3, we needed to determine whether a vertex was visible to the boundary of a cavity which is equivalent to checking whether the volume formed by every proposed insertion element is positive. Thus, in four dimensions, we need to robustly determine when the volume of a pentatope is positive. To alleviate the expertise needed in developing geometric predicates, we use the Predicate Construction Kit (PCK) of Lévy[43] which abstracts every floating-point value to an `expansion_nt` structure which tracks the round-off error incurred by floating-point operations. Algorithm C.1 lists the `orient4d` predicate we develop in the language of PCK.

It should be noted that employing the exact volume calculation in Algorithm C.1 incurs a significant computational cost in comparison to the inexact calculation (by simply calculating the determinant with floating-point arithmetic). As a result, a filter is used to determine

122. Shewchuk, *Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates*. 1996

43. Lévy, *"Geogram: A Programming Library of Geometric Algorithms"*. 2016

◁ **Thank you Bruno Lévy!**

As we can see, the implementation of this predicate is very simple in the language of PCK!

when the exact calculation should be used, and when the inexact one suffices. This filter is generated using FPG[123] using the script in `pck_filters.sh` in the `avro/test/develop/orient4d` directory (click here to view the script).

123. Edelsbrunner *et al.*, *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms*. 1990

```
double
orient4d( const double* p0 , const double* p1 , const double* p2 ,
          const double* p3 , const double* p4 )
{
  // use the filter to determine if we can use
  // the fast volume calculation
  int s = orient4d_filter(p0,p1,p2,p3,p4);
  if (s!=FPG_UNCERTAIN_VALUE)
    return orient4dfast(p0,p1,p2,p3,p4);

  // compute the vectors p1-p0, p2-p0, p3-p0, p4-p0
  const expansion& a11 = expansion_diff( p1[0] , p0[0] );
  const expansion& a12 = expansion_diff( p2[0] , p0[0] );
  const expansion& a13 = expansion_diff( p3[0] , p0[0] );
  const expansion& a14 = expansion_diff( p4[0] , p0[0] );

  const expansion& a21 = expansion_diff( p1[1] , p0[1] );
  const expansion& a22 = expansion_diff( p2[1] , p0[1] );
  const expansion& a23 = expansion_diff( p3[1] , p0[1] );
  const expansion& a24 = expansion_diff( p4[1] , p0[1] );

  const expansion& a31 = expansion_diff( p1[2] , p0[2] );
  const expansion& a32 = expansion_diff( p2[2] , p0[2] );
  const expansion& a33 = expansion_diff( p3[2] , p0[2] );
  const expansion& a34 = expansion_diff( p4[2] , p0[2] );

  const expansion& a41 = expansion_diff( p1[3] , p0[3] );
  const expansion& a42 = expansion_diff( p2[3] , p0[3] );
  const expansion& a43 = expansion_diff( p3[3] , p0[3] );
  const expansion& a44 = expansion_diff( p4[3] , p0[3] );

  // compute the determinant with exact arithmetic
  const expansion& Delta = expansion_det4x4( a11 , a12 , a13 , a14 ,
                                             a21 , a22 , a23 , a24 ,
                                             a31 , a32 , a33 , a34 ,
                                             a41 , a42 , a43 , a44 );
  if (Delta.sign()==ZERO) return 0.0;
  return Delta.value();
}
```

Algorithm C.1: Exact calculation of the volume of a pentatope in the language of the Predicate Construction Kit[43]. The inputs are the five vertex coordinates of a pentatope and the output is the signed volume of the pentatope. The algorithm returns zero when the exact volume is zero.

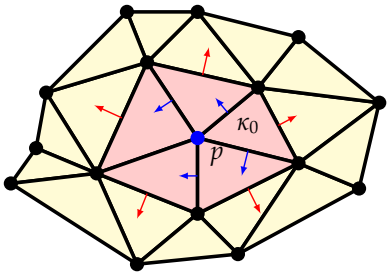


Figure C.1: Finding the ball of p by starting with κ_0 and marching through its neighbours. The blue arrows indicate marches through neighbours that contain p whereas the red arrows indicate when the recursion terminates because the neighbour does not contain p. The red triangles ultimately form the ball $b(p)$ of p.

Inverse topology

As mentioned in Chapter 3, a frequent operation in the mesh adaptation tool is the calculation of the simplices sharing a particular facet f, $\mathcal{C}(f)$ (Equation 3.2).

A costly algorithm would be to loop through all simplices and check if $f \subset \kappa$ for each simplex κ of the mesh. Instead, let us define the *inverse topology*, in which every vertex is aware of a single simplex it is attached to. Our goal is to compute the *ball* $b(p)$ of a vertex p given

a simplex κ_0 attached to p. Since we retain the neighbour relations of each simplex, we can march through the neighbours of κ_0, adding to $b(p)$ if the neighbour κ_i contains the vertex p. If so, we recursively march into the neighbours of κ_i, unless it has already been visited. See Figure C.1 for a description of this procedure.

If the facet f is simply a vertex, then we are done. Otherwise, the simplices sharing f are formed from the intersection of the ball of each vertex:

$$\mathcal{C}(f) = \bigcap_{v \in f} b(v).$$

Association between source files and algorithms

Table C.1 lists the association between some of the algorithms or equations described in this thesis with the source code in `avro`.

Function	Class/Function Name in `avro`	File	Notes
`adaptMesh` (Algorithm 3.6)	`adapt` via `SerialAdaptation`	`api/adapt.cpp`	main adaptation algorithm
Base cavity operator	`Cavity`	`mesh/local/cavity.[h,cpp]`	Equations 3.2 and 3.3
Specialized operators	`Insert/Collapse/EdgeSwap/Smooth`	`mesh/local/primitive.[h,cpp]`	also does geometry hierarchy checks
`splitEdges` (Algorithm 3.5)	`splits`	`mesh/local/unitise.hpp`	method of `SerialAdaptation`
`collapseEdges` (Algorithm 3.4)	`collapses`	`mesh/local/unitise.hpp`	method of `SerialAdaptation`
`swapEdges` (Algorithm 3.2)	`swaptimise`	`mesh/local/adaptation.cpp`	method of `SerialAdaptation`
`smoothVertices` (Algorithm 3.1)	`smooth`	`mesh/local/optimise.hpp`	method of `SerialAdaptation`

Table C.1: Association between algorithms in this thesis with the corresponding implementations in `avro`.

C.2 SANS

For a topology with dimension n, let us define a *cell* as an element of the mesh (or `XField` in `SANS`) which also has a topological dimension of n. Also define a *trace* as an element of the mesh with topological dimension $n-1$. Traces are further classified into being either *interior* or *boundary* traces, and are stored in corresponding `InteriorTraceGroups` or `BoundaryTraceGroups`. Interior trace groups have unique *left* and *right* `CellGroups` and boundary trace groups are divided according to which boundary they are associated with (for the specification of boundary conditions).

Transforming trace coordinates to cell coordinates

`SANS` requires the calculation of cell coordinates from the trace coordinates for any orientation of an incoming trace. That is, given reference trace coordinates $\mathbf{s} \in \mathbb{R}^{n-1}$ for an arbitrarily oriented trace $\mathbf{f}$, we need to determine the reference cell coordinates $\mathbf{u} \in \mathbb{R}^n$.

Let the reference coordinates of the orthogonal trace simplex be stored columnwise in a matrix $\mathbf{M}$, i.e. the i-th column of $\mathbf{M}$ is $\mathbf{p}_i$ as defined in Chapter 2). All the orientations of the reference trace

simplex are encoded in the permutations of $\mathbf{f}_0 = [0, 1, \ldots, n]$. Thus $\mathbf{f}$ is a permutation of $\mathbf{f}_0$ and we can write $\mathbf{f} = \mathbf{P}\mathbf{f}_0$ in terms of the permutation matrix $\mathbf{P}$. Then $\mathbf{u} = \mathbf{MP}\boldsymbol{\alpha}$ where $\boldsymbol{\alpha}$ are the barycentric coordinates: $\boldsymbol{\alpha} = [1 - \sum_{i=1}^{n} s_i, s_1, \ldots, s_{n-1}]^t$.

◁ **Generating script:**

In `SANS`, please see the corresponding MATLAB script `BasisFunction/Documentation/TraceToCellRefCoord.m` which generates these expressions (symbolically for $\mathbf{s}$) and also reports the orientation of each permutation.

Local split implementation

The local split implementation hinges on the `XField_Lagrange` capability which provides the ability to construct `XFields` by reading files, through mesher interfaces, or for unit tests. In particular, an `XField_Lagrange` can construct the interior and boundary trace associations by simply describing the vertices (in `SANS`, the DOF of the `XField`), as well as the cell group and boundary group topologies.

When performing a local split, we can extract a *local patch* (`XField_LocalPatch`) from a global `XField` by simply inheriting from a `XField_Lagrange` and store the DOF and cells we wish to extract. The `XField_Lagrange` class was modified to return the leftover traces after filling in the interior trace groups. These leftover traces form the boundary of a local patch and we need to classify whether these traces are in a *ghost* boundary trace group of the local patch or are on actual boundary groups of the global `XField`. Before proceeding, let us define some conventions.

Conventions for local `XField`s

1. There are always two cell groups. Cell group 0 (`RESOLVE`) consists of elements in which the solution will be recomputed. Cell group 1 (`FIXED`) consists of elements in which the solution DOF will be fixed during the local solve procedure.

2. The are at most three interior trace groups. Interior trace groups are unique in terms of which cell groups they border. Thus there may an interior trace group bordering the 0-0 cell group (`RESOLVE-RESOLVE`), 0-1 cell groups (`RESOLVE-FIXED`), or 1-1 cell groups (`FIXED-FIXED`).

3. There are any number of boundary trace groups and the number of these is determined by which Galerkin method is used (discontinuous or continuous).

4. There is always a single *ghost* boundary trace group.

Extracting a local patch The extraction of a local patch is composed of the following stages:

1. Identify which cells of the global `XField` should be added to the `RESOLVE` cell group and which should be added to the `FIXED` cell group,

2. Use `XField_Lagrange` to connect the cell groups, thus filling in the interior trace groups while listing traces (the boundary *leftovers*) which are only bordered by one cell,
3. Classify the leftover traces (call these $\mathcal{F}$) into either ghost or boundary trace groups,
4. Finalize the local `XField` by assigning cell groups and trace groups,
5. Determine the mappings of the cells and traces (both interior and boundary) to elements in the global `XField`.

The last step is necessary when projecting the solution from the global `XField` to the local one but also when computing the curvilinear coordinates (if necessary) of the `XField_LocalPatch`. In order to retrieve the mappings in this step, the global information (group and element) of each local element is stored in Step 1.

When classifying the leftover traces $\mathcal{F}$ in Step 3, we need to determine whether it is placed in a boundary trace group or in the ghost boundary trace group. First and foremost, if a trace $f \in \mathcal{F}$ maps to a boundary trace group of the global `XField`, then f is added to a boundary trace group of the local patch, regardless of the discretization. For discontinuous Galerkin discretizations, $f \in \mathcal{F}$ is added to a boundary group if the left element of the trace is in the `RESOLVE` cell group of the local patch. For continuous Galerkin discretizations, all leftover traces are added to boundary groups. For more information, see the function `XField_LocalPatch::isBoundaryTrace`.

◁ **Cell-to-trace connectivity:**

The cell-to-trace connectivity structure in the global `XField` can be used to determine which global trace is associated with each local trace

Splitting a local patch After a local patch has been extracted, it can be split by specifying which canonical edge of a cell should be split. Since local patches are extracted from a single cell (call this the main cell), this canonical edge corresponds to that of the main cell. The local split procedure consists of the following stages:

1. Create the new vertex (DOF) at the midpoint of the edge e,
2. Use the (unsplit) local patch as well as the edge information to build a set of traces $\mathcal{S}$ which contains both *unsplit* and *split* traces,
3. Compute the cells that should be split, i.e. compute $\mathcal{C} \leftarrow \mathcal{C}(e)$ for the edge e as in Equation 3.2,
4. Add cells into the `XField_Lagrange`:
   - Any $\kappa \notin \mathcal{C}$ is added to the appropriate `RESOLVE` or `FIXED` cell group,
   - Every element $\kappa \in \mathcal{C}$ is split into two new elements by finding the index of each edge endpoint in κ and setting it to the index of the newly created vertex,

◁ **Extending this framework:**

Future work may involve extracting local patches about *vertices* instead of cells, in which there is no *main* cell. In this situation, either (1) both the cell and canonical edge to be split will need to be specified, or (2) the global indices of the edge vertices need to be looked up to find the corresponding edge in the local patch.

5. Use `XField_Lagrange` to connect the cell groups, thus filling in the interior trace groups,
6. Classify the leftover traces (again, call these $\mathcal{F}$) into either ghost or boundary trace groups,
7. Determine the mappings of the cells and traces (both interior and boundary) to the global elements in the global `XField` and construct the `ElementSplitInfo` structure for each cell and trace.

The set of traces $\mathcal{S}$ is needed in order to classify the leftover boundary traces $\mathcal{F}$ in Step 6. It is also needed such that we can lookup the global trace information for Step 7. Furthermore, Step 7 requires $\mathcal{S}$ in order to determine which of the traces of the split patch are `New`, `Split` or `Unsplit` for the `ElementSplitInfo` structure of each trace.

Once all the cell and trace maps are complete, the local patch provides the option to project the DOF of the global `XField` to the split `XField_LocalPatch` if the global `XField` is curved.

◁ **`TraceSoup`:**

The set of traces $\mathcal{S}$ is referred to as a *trace soup* in `SANS`. It is simply a container for traces of the original (unsplit) local patch that are either split or unsplit during the local split process. Therefore, it does not account for the new (interior) traces that are created during a split. The `TraceSoup::lookup` method determines the `ElementSplitFlag`.

Implementation	Edge	Iso	dG	cG	Curved	Mixed	new SLOC
`XField_Local`	✓	✓	✓	✗	✓	✓(?)	3,660
`XField_ElementLocal`	✓	✗	✓(?)	✓	✓(?)	✗	1,434
`XField_LocalPatch`	✓	✗	✓	✓	✓	✗	612

Table C.2: Local split implementation capabilities. The term *Edge* refers to whether edge splits are supported and *Iso* refers to whether isotropic splits are supported. The term *Mixed* refers to whether mixed-element meshes are supported. The last column tabulates the number of new source lines of code (SLOC) that were needed for the implementation. The markings ✓(?) indicate that the method has been implemented but yet to be demonstrated in practice.

Timing results Previous implementations exist in the `XField_Local` and `XField_ElementLocal` classes which provide similar functionalities in a dimension-dependent manner. With the exception of isotropic splits (in which every edge of the main cell is split) and mixed-element meshes, the current implementation in `XField_LocalPatch` matches all desired capabilities of the local split procedure. In particular, edge splits for both discontinuous and continuous Galerkin discretizations on both straight-sided and curvilinear meshes are supported. Table C.2 compares these capabilities. Furthermore, Tables C.3 and C.4 tabulate some timing results of each implementation All timing results were obtained on an Intel i7-5930K CPU at 3.50GHz.

Implementation	$2d$ sec/split	$3d$ sec/split	$4d$ sec/split
`XField_Local`	4.6e-5	7.5e-5	n/a
`XField_ElementLocal`	2.8e-5	1.3e-4	n/a
`XField_LocalPatch`	3.5e-5	6.4e-5	1.1e-4

Table C.3: Timing results for splits with the discontinuous Galerkin discretization with previous (`XField_Local`, `XField_ElementLocal`) and current (`XField_LocalPatch`) implementations.

Implementation	$2d$ sec/split	$3d$ sec/split	$4d$ sec/split
`XField_ElementLocal`	7.8e-5	3.7e-4	n/a
`XField_LocalPatch`	9.0e-5	4.8e-4	4.9e-3

Table C.4: Timing results for splits used with the continuous Galerkin discretization with previous (`XField_ElementLocal`) and current (`XField_LocalPatch`) implementations.

Bibliography

[1] M. Yano, J. M. Modisette, and D. L. Darmofal, "The Importance of Mesh Adaptation for Higher-Order Discretizations of Aerodynamic Flows," in *20th AIAA Computational Fluid Dynamics Conference*, 2011–3852, June 2011 (cit. on p. 17).

[2] J. Slotnick, A. Khodadoust, J. Alonso, D. L. Darmofal, W. Gropp, E. Lurie, and D. J. Mavriplis, "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," NASA/CR-2014-218178, 2014 (cit. on p. 17).

[3] M. Yano, "An Optimization Framework for Adaptive Higher-Order Discretizations of Partial Differential Equations on Anisotropic Simplex Meshes," PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2012 (cit. on pp. 18, 38, 59, 78, 79, 91).

[4] S. Jayasinghe, "An Adaptive Space-Time Discontinuous Galerkin Method for Reservoir Flows," PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2018 (cit. on pp. 18, 20).

[5] S. Jayasinghe, D. L. Darmofal, N. K. Burgess, M. C. Galbraith, and S. R. Allmaras, "A Space-Time Adaptive Method for Reservoir Flows: Formulation and One-Dimensional Application," *Computational Geosciences*, Vol. 22, No. 1, pp. 107–123, February 2018 (cit. on pp. 18, 20).

[6] J. T. Oden, "A General Theory of Finite Elements II. Applications," *International Journal for Numerical Methods in Engineering*, Vol. 1, No. 3, pp. 247–259, 1969 (cit. on p. 19).

[7] J. Argyris and D. Scharpf, "Finite Elements in Time and Space," *Nuclear Engineering and Design*, Vol. 10, No. 4, pp. 456–464, 1969 (cit. on p. 19).

[8] I. Fried, "Finite-Element Analysis of Time-Dependent Phenomena," *AIAA Journal*, Vol. 7, No. 6, pp. 1170–1173, 1969 (cit. on p. 19).

[9] M. Behr, "Simplex Space-Time Meshes in Finite Element Simulations," *International Journal for Numerical Methods in Fluids*, Vol. 57, No. 9, pp. 1421–1434, July 2008 (cit. on p. 19).

[10] A. Üngör and A. Sheffer, "Tent-Pitcher: A Meshing Algorithm for Space-Time Discontinuous Galerkin Methods," in *Proceedings of the 9th International Meshing Roundtable*, 2000, pp. 111–122 (cit. on p. 19).

[11] A. D. Mont, "Adaptive Unstructured Spacetime Meshing for Four-Dimensional Spacetime Discontinuous Galerkin Finite Element Methods," Master's thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, December 2011 (cit. on p. 19).

[12] S. Thite, "Adaptive Spacetime Meshing for Discontinuous Galerkin Methods," *Computational Geometry*, Vol. 42, No. 1, pp. 20–44, 2007 (cit. on p. 19).

[13] K. J. Fidkowski and Y. Luo, "Output-Based Space-Time Mesh Adaptation for the Compressible Navier-Stokes Equations," *Journal of Computational Physics*, Vol. 230, No. 14, pp. 5753–5773, 2011 (cit. on p. 20).

[14] K. J. Fidkowski, "Output-Based Space-Time Mesh Optimization for Unsteady Flows Using Continuous-in-Time Adjoints," *Journal of Computational Physics*, Vol. 341, No. 15, pp. 258–277, July 2017 (cit. on p. 20).

[15] W. Bangerth and R. Rannacher, "Finite Element Approximation of the Acoustic Wave Equation: Error Control and Mesh Adaptation," *East-West Journal of Numerical Mathematics*, Vol. 7, No. 4, pp. 263–282, 1999 (cit. on p. 20).

[16] W. Bangerth, M. Geiger, and R. Rannacher, "Adaptive Galerkin Finite Element Methods for the Wave Equation," *Computational Methods in Applied Mathematics*, Vol. 10, No. 1, pp. 3–48, 2010 (cit. on p. 20).

[17] R. Hartmann, "Adaptive FE Methods for Conservation Equations," in *Hyperbolic Problems: Theory, Numerics, Applications: Eighth International Conference in Magdeburg*, H. Freistühler and G. Warnecke, Eds., Vol. 141, ser. International Series of Numerical Mathematics, Birkhäuser, Basel, February 2001, pp. 495–503 (cit. on p. 20).

[18] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz, "Adaptive Remeshing for Compressible Flow Computations," *Journal of Computational Physics*, Vol. 72, pp. 449–466, 1987 (cit. on p. 21).

[19] R. Löhner and P. Parikh, "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method," *International Journal for Numerical Methods in Fluids*, Vol. 8, pp. 1135–1149, January 1988 (cit. on p. 21).

[20] J. Peraire, J. Peiro, L. Formaggia, K. Morgan, and O. C. Zienkiewicz, "Finite Element Euler Computations in Three Dimensions," *International Journal for Numerical Methods in Engineering*, Vol. 26, pp. 2135–2159, October 1988 (cit. on p. 21).

[21] D. Marcum and N. Weatherill, "Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection," *AIAA Journal*, Vol. 33, pp. 1619–1625, 1995 (cit. on p. 21).

[22] R. Löhner, "Adaptive Remeshing for Transient Problems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 75, No. 1, pp. 195–214, 1989 (cit. on p. 21).

[23] J. Peraire, J. Peiró, and K. Morgan, "Adaptive Remeshing for Three-Dimensional Compressible Flow Computations," *Journal of Computational Physics*, Vol. 103, No. 2, pp. 269–285, 1992 (cit. on p. 21).

[24] S. Pirzadeh, "Three-Dimensional Unstructured Viscous Grids by the Advancing-Layers Method," *AIAA Journal*, Vol. 34, No. 1, pp. 43–49, 1996 (cit. on p. 21).

[25] F. Alauzet and D. Marcum, "A Closed Advancing-Layer Method with Connectivity Optimization-based Mesh Movement for Viscous Mesh Generation," *Engineering with Computers*, Vol. 31, No. 3, pp. 545–560, July 2015 (cit. on p. 21).

[26] D. F. Watson, "Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes," *The Computer Journal*, Vol. 24, No. 2, p. 167, 1981 (cit. on p. 22).

[27] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, Qhull: Quickhull Algorithm for Computing the Convex Hull, Astrophysics Source Code Library, April 2013 (cit. on p. 22).

[28] H. Borouchaki, P. George, F. Hecht, P. Laug, and E. Saltel, "Mailleur Bidimensionnel de Delaunay Gouverné par une Carte de Métriques. Partie I: Algorithmes," INRIA-Rocquencourt, France. Tech Report No. 2741, 1995 (cit. on p. 22).

[29] F. Hecht, BAMG: Bidimensional Anisotropic Mesh Generator, `http://www-rocq1.inria.fr/gamma/cdrom/www/bamg/eng.htm`, INRIA-Rocquencourt, France, 1998 (cit. on pp. 22, 24, 109).

[30] F. J. Bossen and P. S. Heckbert, "A Pliant Method for Anisotropic Mesh Generation," in *Proceedings of the 5th International Meshing Roundtable*, October 1996, pp. 63–74 (cit. on pp. 22, 50).

[31] C. Dobrzynski and P. Frey, "Anisotropic Delaunay Mesh Adaptation for Unsteady Simulations," in *Proceedings of the 17th International Meshing Roundtable*, October 2008, pp. 177–194 (cit. on p. 22).

[32] M. Rouxel-Labbé, M. Wintraecken, and J.-D. Boissonnat, "Discretized Riemannian Delaunay Triangulations," *Procedia Engineering*, Vol. 163, pp. 97–109, 2016, 25th International Meshing Roundtable (cit. on pp. 22, 25).

[33] J.-D. Boissonnat, R. Dyer, A. Ghosh, and N. Martynchuk, "An Obstruction to Delaunay Triangulations in Riemannian Manifolds," *Discrete & Computational Geometry*, Vol. 59, No. 1, pp. 226–237, January 2018 (cit. on pp. 22, 24).

[34] G. D. Cañas and S. J. Gortler, "Surface Remeshing in Arbitrary Codimensions," *The Visual Computer*, Vol. 22, No. 9, pp. 885–895, September 2006 (cit. on p. 22).

[35] B. Lévy and N. Bonneel, "Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration," in *Proceedings of the 21st International Meshing Roundtable*, 2012 (cit. on pp. 22, 123, 125).

[36] J. F. Nash, "C^1 Isometric Imbeddings," *Annals of Mathematics*, Vol. 60, No. 3, pp. 383–396, November 1954 (cit. on p. 23).

[37] ——, "The Imbedding Problem for Riemannian Manifolds," *Annals of Mathematics*, Vol. 63, No. 1, pp. 20–63, January 1956 (cit. on p. 23).

[38] V. Nivoliers, B. Lévy, and C. Geuzaine, "Anisotropic and Feature Sensitive Triangular Remeshing Using Normal Lifting," *Journal of Computational and Applied Mathematics*, Vol. 289, pp. 225–240, December 2015 (cit. on p. 23).

[39] F. Dassi, S. Perotto, H. Si, and T. Streckenbach, "A Priori Anisotropic Mesh Adaptation Driven by a Higher Dimensional Embedding," *Computer-Aided Design*, Vol. 85, pp. 111–122, 2017 (cit. on p. 23).

[40] F. Dassi, A. Mola, and H. Si, "Curvature-Adapted Remeshing of CAD Surfaces," *Procedia Engineering*, Vol. 82, pp. 253–265, 2014, 23rd International Meshing Roundtable (cit. on p. 23).

[41] F. Dassi, H. Si, S. Perotto, and T. Streckenbach, "Anisotropic Finite Element Mesh Adaptation via Higher Dimensional Embedding," *Procedia Engineering*, Vol. 124, pp. 265–277, 2015, 24th International Meshing Roundtable (cit. on p. 23).

[42] F. Dassi, P. Farrell, and H. Si, "An Anisoptropic Surface Remeshing Strategy Combining Higher Dimensional Embedding with Radial Basis Functions," *Procedia Engineering*, Vol. 163, pp. 72–83, 2016, 25th International Meshing Roundtable (cit. on p. 23).

[43] B. Lévy, "Geogram: A Programming Library of Geometric Algorithms", `http://alice.loria.fr/software/geogram/doc/html/index.html`, Villers les Nancy, France: INRIA Project ALICE, 2016 (cit. on pp. 24, 123, 124, 129, 130).

[44] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, Springer-Verlag, 1996, pp. 203–222 (cit. on p. 24).

[45] H. Si, "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator," *ACM Transactions on Mathematical Software*, Vol. 41, No. 2, 11:1–11:36, February 2015 (cit. on p. 24).

[46] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, Vol. 290, No. 5500, pp. 2319–2323, 2000 (cit. on p. 24).

[47] P. C. Caplan, R. Haimes, D. L. Darmofal, and M. C. Galbraith, "Anisotropic Geometry-Conforming d-simplicial Meshing via Isometric Embeddings," *Procedia Engineering*, Vol. 203, pp. 141–153, 2017, 26th International Meshing Roundtable (cit. on pp. 24, 27, 31, 114, 119, 121, 123, 128).

[48] P. C. Caplan, R. Haimes, and X. Roca, "Isometric Embedding of Curvilinear Meshes Defined on Riemannian Metric Spaces," in *Proceedings of the 27th International Meshing Roundtable*, 2018 (cit. on pp. 24, 27).

[49] Z. Zhong, W. Wang, B. Lévy, J. Hua, and X. Guo, "Computing a High-dimensional Euclidean Embedding from an Arbitrary Smooth Riemannian Metric," *ACM Transactions on Graphics*, Vol. 37, No. 4, 62:1–62:16, July 2018 (cit. on p. 24).

[50] T. Michal and J. Krakos, "Anisotropic Mesh Adaptation through Edge Primitive Operations," in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2012-159, January 2012 (cit. on pp. 25, 43, 59, 62, 109).

[51] M. A. Park and D. L. Darmofal, "Parallel Anisotropic Tetrahedral Adaptation," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008-917, 2008 (cit. on pp. 25, 43).

[52] M. A. Park, "Anisotropic Output-Based Adaptation with Tetrahedral Cut Cells for Compressible Flows," PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2008 (cit. on p. 25).

[53] F. Alauzet and A. Loseille, "A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics," *Computer-Aided Design*, Vol. 72, pp. 13–39, March 2016 (cit. on p. 25).

[54] A. Loseille, "Unstructured Mesh Generation and Adaptation," in *Handbook of Numerical Methods for Hyperbolic Problems*, ser. Handbook of Numerical Analysis, R. Abgrall and C.-W. Shu, Eds., Vol. 18, Elsevier, 2017, ch. 10, pp. 263–302 (cit. on p. 25).

[55] T. Coupez, "Génération de Maillage et Adaptation de Maillage par Optimisation Locale," *Revue Européenne des Éléments Finis*, Vol. 9, No. 4, pp. 403–423, 2000 (cit. on pp. 25, 31, 43, 44, 46).

[56] A. Loseille, F. Alauzet, and V. Menier, "Unique Cavity-Based Operator and Hierarchical Domain Partitioning for Fast Parallel Generation of Anisotropic Meshes," *Computer-Aided Design*, Vol. 85, pp. 53–67, 2017 (cit. on pp. 26, 43, 44, 52, 62, 65, 76, 109).

[57] C. Gruau, "Metric Generation for Anisotropic Mesh Adaptation with Numerical Applications to Material Forming Simulation," PhD thesis, École Nationale Supérieure des Mines de Paris, 2005 (cit. on pp. 26, 43, 46, 47).

[58] P. Tremblay, "2-D, 3-D and 4-D Anisotropic Mesh Adaptation for the Time-Continuous Space-Time Finite Element Method with Applications to the Incompressible Navier-Stokes Equations," PhD thesis, University of Ottawa, 2007 (cit. on p. 26).

[59] B. Grünbaum, *"Convex Polytopes"*, ser. Graduate Texts in Mathematics. Springer, 2003, Vol. 221 (cit. on p. 30).

[60] M. P. do Carmo, *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Inc., 1976 (cit. on p. 32).

[61] F. Alauzet, "Size Gradation Control of Anisotropic Meshes," *Finite Elements in Analysis and Design*, Vol. 46, No. 1-2, pp. 181–202, January 2010 (cit. on p. 33).

[62] D. Ibanez, N. Barral, J. Krakos, A. Loseille, T. Michal, and M. Park, "First Benchmark of the Unstructured Grid Adaptation Working Group," *Procedia Engineering*, Vol. 203, pp. 154–166, 2017, 26th International Meshing Roundtable (cit. on pp. 33, 59, 60, 61).

[63] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache, "Log-Euclidean Metrics for Fast and Simple Calculus on Diffusion Tensors," *Magnetic Resonance in Medicine*, Vol. 56, pp. 411–421, 2006 (cit. on p. 33).

[64] A. Loseille and F. Alauzet, "Continuous Mesh Framework Part I: Well-Posed Continuous Interpolation Error," *SIAM Journal on Numerical Analysis*, Vol. 49, No. 1, pp. 38–60, 2011 (cit. on p. 34).

[65] P. Frey and P. George, *Mesh Generation: Application to Finite Elements: Second Edition*. January 2008 (cit. on p. 34).

[66] A. Loseille, "Metric-Orthogonal Anisotropic Mesh Generation," *Procedia Engineering*, Vol. 82, pp. 403–415, 2014, 22nd International Meshing Roundtable (cit. on pp. 35, 52, 109).

[67] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, D. May, L. C. McInnes, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc Users Manual," ANL-95/11 - Revision 3.8, Argonne National Laboratory, 2017 (cit. on p. 38).

[68] P. L. Roe, "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, pp. 357–372, 1981 (cit. on p. 38).

[69] F. Bassi and S. Rebay, "GMRES Discontinuous Galerkin Solution of the Compressible Navier-Stokes Equations," in *Discontinuous Galerkin Methods: Theory, Computation and Applications*, K. Cockburn and Shu, Eds., Berlin: Springer, 2000, pp. 197–208 (cit. on p. 38).

[70] A. Stroud, *Approximate Calculation of Multiple Integrals*. Prentice-Hall Inc., 1971 (cit. on p. 38).

[71] R. Becker and R. Rannacher, "An Optimal Control Approach to A Posteriori Error Estimation in Finite Element Methods," in *Acta Numerica*, A. Iserles, Ed., Cambridge University Press, 2001 (cit. on p. 39).

[72] J. Kudo, "Robust Adaptive High-Order RANS Methods," Master's thesis, Massachusetts Institute of Technology, Computation for Design and Optimization, June 2014 (cit. on p. 41).

[73] P. George, "Gamanic3d, Adaptive Anisotropic Tetrahedral Mesh Generator," Technical Report, INRIA, 2002 (cit. on p. 43).

[74] G. Rokos, G. J. Gorman, J. Southern, and P. H. J. Kelly, "A Thread-Parallel Algorithm for Anisotropic Mesh Adaptation," arXiv:1308.2480, 2013 (cit. on p. 43).

[75] D. A. Ibanez, "Conformal Mesh Adaptation on Heterogeneous Supercomputers," PhD thesis, Rensselaer Polytechnic Institute, 2016 (cit. on pp. 43, 76).

[76] C. Dobrzynski, "MMG3D: User Guide," Technical Report RT-0422, INRIA, March 2012 (cit. on p. 43).

[77] B. M. Klingner and J. R. Shewchuk, "Aggressive Tetrahedral Mesh Improvement," in *Proceedings of the 16th International Meshing Roundtable*, October 2007, pp. 3–23 (cit. on p. 44).

[78] C. Gruau and T. Coupez, "3D Tetrahedral, Unstructured and Anisotropic Mesh Generation with Adaptation to Natural and Multidomain Metric," *Computer Methods in Applied Mechanics and Engineering*, Vol. 194, No. 48–49, pp. 4951–4976, 2005 (cit. on p. 44).

[79] P. C. Caplan, "An Adaptive Framework for High-Order, Mixed-Element Numerical Simulations," Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2014 (cit. on p. 53).

[80] A. Huerta, A. Angeloski, X. Roca, and J. Peraire, "Efficiency of High-Order Elements for Continuous and Discontinuous Galerkin Methods," *International Journal for Numerical Methods in Engineering*, Vol. 96, No. 9, pp. 529–560, 2013 (cit. on p. 53).

[81] Unstructured Grid Adaptation Working Group, UGAWG GitHub repository, `https://github.com/UGAWG`, 2019 (cit. on p. 60).

[82] H. Digonnet, T. Coupez, P. Laure, and L. Silva, "Massively Parallel Anisotropic Mesh Adaptation," *The International Journal of High Performance Computing Applications*, Vol. 33, No. 1, pp. 3–24, 2017 (cit. on pp. 76, 109).

[83] C. Tsolakis, N. Chrisochoides, M. Park, A. Loseille, and T. Michal, "Parallel Anisotropic Unstructured Grid Adaptation," in *2019 AIAA Science and Technology Forum*, 2019–1995, 2019 (cit. on p. 76).

[84] M. C. Galbraith, S. R. Allmaras, and D. L. Darmofal, "A Verification Driven Process for Rapid Development of CFD Software," in *53rd AIAA Aerospace Sciences Meeting*, 2015-0818, January 2015 (cit. on p. 77).

[85] A. Grundmann and H. M. Moller, "Invariant Integration Formulas for the n-Simplex by Combinatorial Methods," *SIAM Journal on Numerical Analysis*, Vol. 15, pp. 282–290, April 1978 (cit. on p. 77).

[86] P. Houston, E. H. Georgoulis, and E. Hall, "Adaptivity and A Posteriori Error Estimation for DG Methods on Anisotropic Meshes," *International Conference on Boundary and Interior Layers*, 2006 (cit. on pp. 78, 91).

[87] W. Cao, "An Interpolation Error Estimate on Anisotropic Meshes in R^n and Optimal Metrics for Mesh Refinement," *SIAM Journal on Numerical Analysis*, Vol. 45, No. 6, pp. 2368–2391, 2007 (cit. on pp. 78, 91).

[88] H. W. Kuhn, "Simplicial Approximation of Fixed Points," *Proceedings of the National Academy of Science*, Vol. 61, pp. 1238–1242, December 1968 (cit. on pp. 78, 121, 127).

[89] H. A. Carson, D. L. Darmofal, M. C. Galbraith, and S. R. Allmaras, "Analysis of Output-Based Error Estimation for Finite Element Methods," *Applied Numerical Mathematics*, Vol. 118, pp. 182–202, 2017 (cit. on pp. 94, 101).

[90] E. Abbott, *Flatland: A Romance of Many Dimensions*. Roberts Brothers, 1885 (cit. on p. 107).

[91] J. Chawner, J. Dannenhoffer, M. Gammon, C. Ollivier-Gooch, B. Jones, J. Masters, T. Michal, N. Taylor, H. Thornburg, and C. Woeber, 2nd AIAA Geometry and Mesh Generation Workshop, `http://www.gmgworkshop.com`, 2019 (cit. on p. 110).

[92] F. Bassi and S. Rebay, "High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations," *Journal of Computational Physics*, Vol. 138, No. 2, pp. 251–285, 1997 (cit. on p. 110).

[93] P.-O. Persson and J. Peraire, "Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics," in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, 2009-0949, 2009 (cit. on p. 110).

[94] M. Fortunato and P.-O. Persson, "High-order Unstructured Curved Mesh Generation using the Winslow Equations," *Journal of Computational Physics*, Vol. 307, pp. 1–14, 2016 (cit. on p. 110).

[95] X. Roca, A. Gargallo-Peiró, and J. Sarrate, "Defining Quality Measures for High-Order Planar Triangles and Curved Mesh Generation," in *Proceedings of the 20th International Meshing Roundtable*, Springer Berlin Heidelberg, 2011, pp. 265–383 (cit. on p. 110).

[96] E. Ruiz-Gironés, X. Roca, and J. Sarrate, "High-Order Mesh Curving by Distortion Minimization with Boundary Nodes Free to Slide on a 3D CAD Representation," *Computer-Aided Design*, Vol. 72, pp. 52–64, 2015 (cit. on p. 110).

[97] E. Ruiz-Gironés, J. Sarrate, and X. Roca, "Defining an L2-Disparity Measure to Check and Improve the Geometric Accuracy of Non-Interpolating Curved High-Order Meshes," *Procedia Engineering*, Vol. 124, pp. 122–134, 2015, 24th International Meshing Roundtable (cit. on p. 110).

[98] E. Ruiz-Gironés, J. Sarrate, and X. Roca, "Generation of Curved High-order Meshes with Optimal Quality and Geometric Accuracy," *Procedia Engineering*, Vol. 163, pp. 315–327, 2016, 25th International Meshing Roundtable (cit. on p. 110).

[99] E. Ruiz-Gironés, A. Gargallo-Peiró, J. Sarrate, and X. Roca, "An Augmented Lagrangian Formulation to Impose Boundary Conditions for Distortion-Based Mesh Moving and Curving," *Procedia Engineering*, 2017, 26th International Meshing Roundtable (cit. on p. 110).

[100] A. Gargallo-Peiró, X. Roca, and J. Sarrate, "A Surface Mesh Smoothing and Untangling Method Independent of the CAD Parameterization," *Computational Mechanics*, Vol. 53, No. 4, pp. 587–609, 2014 (cit. on p. 110).

[101] A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate, "Defining Quality Measures for Validation and Generation of High-Order Tetrahedral Meshes," in *Proceedings of the 22nd International Meshing Roundtable*, Springer, 2014, pp. 109–126 (cit. on p. 110).

[102] T. Toulorge, J. Lambrechts, and J.-F. Remacle, "Optimizing the Geometrical Accuracy of Curvilinear Meshes," *Journal of Computational Physics*, Vol. 310, pp. 361–380, 2016 (cit. on p. 110).

[103] A. Johnen, J.-F. Remacle, and C. Geuzaine, "Geometrical Validity of Curvilinear Finite Elements," *Journal of Computational Physics*, Vol. 233, pp. 359–372, 2013 (cit. on p. 111).

[104] A. Johnen, C. Geuzaine, T. Toulorge, and J.-F. Remacle, "Efficient Computation of the Minimum of Shape Quality Measures on Curvilinear Finite Elements," *Procedia Engineering*, Vol. 163, pp. 328–339, 2016, 25th International Meshing Roundtable (cit. on p. 111).

[105] R. Feuillet, A. Loseille, and F. Alauzet, "P2 Mesh Optimization Operators," in *Proceedings of the 27th International Meshing Roundtable*, 2018 (cit. on p. 111).

[106] T. Coupez, "On a Basis Framework for High Order Anisotropic Mesh Adaptation," in *Proceedings of the 26th International Meshing Roundtable*, Springer Berlin Heidelberg, 2017 (cit. on p. 111).

[107] A. Loseille and R. Feuillet, "Vizir: High-order Mesh and Solution Visualization using OpenGL 4.0 Graphic Pipeline," in *2018 AIAA Aerospace Sciences Meeting*, 2018-1174, January 2018, pp. 1–13 (cit. on p. 111).

[108] J. V. Langenhove, D. Lucor, F. Alauzet, and A. Belme, "Goal-Oriented Error Control of Stochastic System Approximations using Metric-Based Anisotropic Adaptations," *Journal of Computational Physics*, Vol. 374, pp. 384–412, 2018 (cit. on p. 112).

[109] R. Haimes and M. Drela, "On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design," in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2012–683, 2012 (cit. on p. 113).

[110] R. Haimes and J. Dannenhoffer, "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry," in *21st AIAA Computational Fluid Dynamics Conference*, 2013–3073, 2013 (cit. on pp. 113, 122).

[111] R. Haimes and J. Dannenhoffer, "EGADSlite: A Lightweight Geometry Kernel for HPC," in *AIAA Aerospace Sciences Meeting*, 2018–1401, January 2018 (cit. on pp. 113, 128).

[112] I. E. Sutherland and G. W. Hodgman, "Reentrant Polygon Clipping," *Communications of the ACM*, Vol. 17, No. 1, pp. 32–42, January 1974 (cit. on p. 114).

[113] B. Lévy, "Robustness and Efficiency of Geometric Programs: The Predicate Construction Kit," *Computer-Aided Design*, Vol. 72, pp. 3–12, 2016 (cit. on p. 114).

[114] M. Henk, J. Richter-Gebert, and G. M. Ziegler, "Basic Properties of Convex Polytopes," in *Handbook of Discrete and Computational Geometry, 2nd Ed.* 2004 (cit. on p. 114).

[115] E. Cohen, R. F. Risenfeld, and G. Elber, *Geometric Modeling with Splines*. A K Peters/CRC Press, 2001 (cit. on p. 122).

[116] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, "On Centroidal Voronoi Tessellation: Energy Smoothness and Fast Computation," *ACM Transactions on Graphics*, Vol. 28, No. 4, 101:1–101:17, September 2009 (cit. on pp. 124, 128).

[117] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transaction on Information Theory*, Vol. 28, No. 2, pp. 129–137, 1982 (cit. on p. 124).

[118] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi Tessellations: Applications and Algorithms," *SIAM Review*, Vol. 41, pp. 637–676, 1999 (cit. on p. 124).

[119] Q. Du and M. Gunzburger, "Grid Generation and Optimization Based on Centroidal Voronoi Tessellations," *Applied Mathematics and Computation*, Vol. 133, pp. 591–607, 2002 (cit. on p. 124).

[120] Q. Du and D. Wang, "Tetrahedral Mesh Generation and Optimization Gased on Centroidal Voronoi Tessellations," *International Journal for Numerical Methods in Engineering*, Vol. 56, pp. 1355–1373, 2003 (cit. on p. 124).

[121] ——, "Anisotropic Centroidal Voronoi Tessellations and their Applications," *SIAM Journal on Scientific Computing*, Vol. 26, No. 3, pp. 737–761, 2005 (cit. on p. 124).

[122] J. R. Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," *Discrete & Computational Geometry*, Vol. 18, No. 3, pp. 305–363, 1996 (cit. on p. 129).

[123] H. Edelsbrunner and E. P. Mücke, "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms," *ACM Transactions on Graphics*, Vol. 9, No. 1, pp. 66–104, 1990 (cit. on p. 130).