Mesh Refinement and High-order Reconstruction for Finite Element Methods on Unstructured Meshes

A Dissertation Presented

by

Xinglin Zhao

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

December 2016

Stony Brook University

The Graduate School

Xinglin Zhao

We, the dissertation committe for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Xiangmin Jiao
Associate Professor, Department of Applied Mathematics and Statistics

James Glimm
Professor, Department of Applied Mathematics and Statistics

Roman Samulyak
Professor, Department of Applied Mathematics and Statistics

Vijay S. Mahadevan
Assistant Computational Scientist, Argonne National Laboratory

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Mesh Refinement and High-order Reconstruction for Finite Element Methods on Unstructured Meshes

by

Xinglin Zhao

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2016

In large scale simulations of complex partial differential equations (PDE's) using finite element methods (FEM), mesh generation, remeshing and linear solver are the most vital steps to obtain accurate solutions. All these areas have been explored quite extensively. We seek to develop an integrated framework for these steps. During simulation it is often desirable to start with a relatively coarse mesh and then refine the mesh accordingly (since in most cases the criteria for mesh resolution is not known a priori). The mesh hierarchy generated from mesh refinement could be utilized by efficient linear solvers like geometric multigrid methods (GMG), which can theoretically deliver optimal time complexity. Thus, it would be advantageous to use hierarchical mesh refinement to achieve high-order of accuracy and computational efficiency.

One effective approach is to refine the mesh uniformly. Successive uniform refinement can not only increase the accuracy of solution but also generate a natural hierarchy which could be further used by GMG. We develop parallel uniform refinement-based algorithms to generate multi-degree, multi-dimensional and multi-level meshes from coarse unstructured meshes, based on the array-based half-facet (AHF) data structure. We demonstrate its applicability to a multigrid finite element solver and the capability is developed under the parallel mesh framework “Mesh Oriented dAtaBase” (MOAB).

Meanwhile, we make effort to extend this framework to adaptive mesh refinement (AMR) which delivers solution more efficiently by increasing the computational effort near interesting features of the solutions. AMR has gradually become a vital step in large-scale numerical simulations. We develop a data structure called Hierarchical AHF to support both refinement and coarsening effectively.

A key aspect of the refinement algorithm is the positioning of the new vertices on curved boundaries. Using linear point projection scheme for the new vertices compromises the accuracy of the geometry and in turn that of the finite element solver. To address this issue, we develop a discrete geometry module in MOAB that provides high-order point projection schemes. To improve the robustness of this method on coarse mesh, we propose two extensions: first, we introduce a Hermite-style weighted-least squares formulation, to take account of both point locations as well as surface normals in the surface reconstruction; second we introduce a new blending technique to ensure G^0 continuity along sharp ridges and corners, while assuring high-order accuracy.

Acknowledgements

First, I would like to thank my advisor Dr. Jiao for all his valuable guidance and support throughout my study. I have learned a lot from his unconventional way of thinking and solving a problem, as well as gain from his vast knowledge in software development and experience in scientific writing. It has been a great pleasure to work with him and I express my gratitude for all his help.

I would like to thank collaborators Vijay S. Mahadevan, Navamita Ray and Iulian Grindeanu of Argonne National Laboratory for providing the oppurtunity to work on interesting problems, guidance as well as financial support.

I would like to thank all the present and past members of my research group, including Rebecca Conley, and Cao Lu for some of experimental result, and Tristan Delaney, Aditi Ghai, Hongxu Liu, Xuebin Wang, Oliver Yang for the helpful discussions. Thank you for all the help and encouragement. It has been great to work with all of you.

I would like to thank my mentors and friends for their support and encouragement: Prof. Shiqing Zhang, Prof. Bin Liu, Guiyou Jian, Wendong Ma, Yijun Yang, Wenxing Zhang, Hui Zhao, Tongyuan Zhao, Xu Zhou and too many others to name.

Lastly, I would like to thank my parents for their endless love and support. I dedicate this thesis to you.

I acknowledge the sponsorship of part of my research from Argonne National Laboratory under Contract DE-AC02-06CH11357 for the SciDAC program funded by the Office of Science, Advanced Scientific Computing Research of the U.S. Department of Energy, as well as from DoD-ARO under contract #W911NF1310249.

Contents

Contents **vi**

List of Figures **viii**

List of Tables **xi**

1 Introduction **1**

2 Background and Related Work **7**
- 2.1 Array-based Half-Facet (AHF) Data Structure . . . 8
- 2.2 Mesh Oriented datABase (MOAB) . . . 9
- 2.3 Adaptive Mesh Refinement for Numerical PDEs . . . 12
- 2.4 Weighted Least Squares and High-order Surface Reconstruction . . 18

3 Hierarchical Uniform Refinement for Unstructured Meshes **22**
- 3.1 Multi-degree Refinement Templates . . . 22
- 3.2 Multi-dimensional Mesh Data Structure . . . 27
- 3.3 Multi-level Mesh Storage . . . 28
- 3.4 Parallel Hierarchical Mesh Generation . . . 32
  - 3.4.1 Parallel Communication to Resolve Shared Entities . . . 33
  - 3.4.2 Refinement Algorithm . . . 38

4 Conformal and Non-conformal Adaptive Mesh Refinement **41**
- 4.1 Adaptive Mesh Refinement and Hierarchical AHF . . . 42
  - 4.1.1 Hierarchical Structure . . . 42
  - 4.1.2 Hierarchical AHF . . . 43
- 4.2 Construction and Modification of Hierarchical AHF . . . 45
  - 4.2.1 Construction of Hierarchical AHF . . . 46
  - 4.2.2 Mesh Adaptation . . . 50
- 4.3 Adaptive Finite Element Method . . . 51
  - 4.3.1 Continuity Constraints . . . 51

4.3.2 Hanging Nodes in FEM 52
4.4 Rules for Refinement 57
4.4.1 Introduction 57
4.4.2 Refine Strategy 59
4.4.3 Condition Number of Π in h-refinement 59

5 High-order Reconstruction 66
5.1 Parallel WALF for Curved Boundary Reconstruction 66
5.1.1 WALF: Weighted Averaging of Local Fittings 67
5.1.2 High-order Surface Reconstruction in MOAB 69
5.2 Hermite-style High-order Surface Reconstruction 72
5.3 Accuracy and Stability of Hermite-style Fittings 81
5.4 High-order Curve Reconstruction 83
5.5 Feature-aware Reconstruction 87
5.5.1 Ridge-aware Curve Reconstruction 88
5.5.2 Feature-aware Surface Reconstruction 91

6 Numerical Results and Applications 98
6.1 Numerical Results of Uniform Mesh Refinement 98
6.1.1 Mesh Quality Under Uniform Mesh Refinement 98
6.1.2 Computational Efficiency 100
6.1.3 Demonstration of UMR Capabilities in Discrete Solvers . . 104
6.2 Numerical Results of Adaptive Mesh Refinement 105
6.2.1 Adaptive Finite Element Method (AFEM) 105
6.2.2 Comparison with Pointer-based Data Structure 108
6.2.3 Mesh Adaptation 111
6.3 Numerical Results of High-order Reconstruction 113
6.3.1 Geometry and PDE 113
6.3.2 Parallel Surface Reconstruction in MOAB 115
6.3.3 Accuracy of Hermite-style High-order Reconstructions . . . 118

7 Conclusion and Further Directions 121

Bibliography 125

List of Figures

2.1 (a) An example of the AHF maps stored for a non-manifold triangular mesh. (b) Typical AHF for mixed-dimensional meshes is composed of half-vertex (black, for explicit edges only), half-edge (blue, for explicit faces only), and half-face (red) data structures. . . 10

3.1 The refinement templates defined for degree 2 and 3 refinements over a reference triangle and quadrilateral entity. The local vertex order of the reference entities follow MOAB conventions [108]. The new vertices to be introduced during refinement are assigned local ids w.r.t. to the reference. Similarly, the children entities are assigned local order. The AHF maps are created for the refined entities in terms of their local ids. 25

3.2 Left: A tetrahedron is divided into congruent smaller tetrahedra and octahedrons. Middle: A reference octahedron. Right: Each octahedron can be further subdivided into four tetrahedra in three possible ways depending on which diagonal is chosen. 25

3.3 Updating the AHF maps for the refined mesh takes place in two stages. First, maps for the children of each triangle are updated. After, both the triangles have been refined, the maps are now updated to connect the children entities sharing a parent facet. 28

3.4 The data layout of meshes at each level. For example, starting with the first quad of the sphere mesh, after each level of refinement, the children are stored in a pre-decided order as defined in the refinement templates. This local order uniquely identifies each child of the parent. 31

3.5 A shared face between two processors. After one level of refinement the new vertices and faces have to be matched. 36

3.6 For a shared edge, there can be two options: either the remote edge has the same orientation or opposite orientation. Depending on the orientation, the new vertices and child edges can be appropriately matched. 37

3.7 The flowchart for generation of mesh hierarchy with length num_levels. 40

4.1 Hierarchical array-based half-facet data structure for a multi-level mesh. 44
4.2 Example of Hierarchical AHF under refinement. 46
4.3 Red-Green Refinement 51
4.4 hanging nodes 52
4.5 Irregularity rules 61

5.1 vertex-based polynomial fittings 67
5.2 2-D illustration of weighted averaging of local fitting 69
5.3 Mesh set for reconstruction 70
5.4 Work flow of local fitting 71
5.5 Ridges and Features 88
5.6 Ridge-aware Reconstruction 89
5.7 The lack of continuity across feature edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$ 92

6.1 Left: A tetrahedral mesh. Right: Distribution of mesh quality for an initial coarse mesh and various degree of refinements. 99
6.2 Time taken to generate hierarchies in serial for different degree of refinements. 100
6.3 The initial mesh on each processor and creation of the whole assemblies from the unit. 102
6.4 Weak scaling studies for the mesh hierarchy generation algorithm using the RGG tool with two levels of uniform refinement on Blues. The left and right figure shows the time in seconds and efficiency as the number of processors increases with fixed problem size. . . . 102
6.5 An example torus mesh with eight partitions used for the scalability study of mesh hierarchy generation. 103
6.6 Mesh convergence of a 2D Poisson and 3D Diffusion-Reaction problem. 106
6.7 Comparison of convergence of multigrid and standard black-box preconditioners. 106
6.8 AFEM: Mesh adaptation for re-entrant corner problem 107
6.9 AFEM: Numerical results for re-entrant corner problem 107
6.10 Example triangular mesh during adaptive mesh refinement. 111
6.11 AMR for 2D triangular mesh: number of active cells 112
6.12 AMR for 3D tetrahedra mesh: number of active cells 112
6.13 Example AMR in 3D: cross-sections of tetrahedral mesh. 112

6.14 Convergence studies for Poisson equation with linear interpolated boundary, WALF reconstructed boundaries and exact boundary . . . 114
6.15 Geometry and PDE: geometric error affects the accuracy of PDE solution 115
6.16 Convergence studies for point projection using high-order surface reconstruction using a sphere and torus mesh in serial and parallel. . 117
6.17 Curve Reconstruction 119
6.18 Convergence study of FAH-WALF 120
6.19 Comparison with WALF 120

List of Tables

3.1 The degrees of refinement currently supported and the corresponding number of children. 26
3.2 An example for the degree 3 template of a triangle reference entity. Children entity connectivity and the template $v2hf$ and $sibhfs$ maps. 26
3.3 Estimates for new entities that will be created after a refinement of degree p. These estimates are for a 3D mesh with explicit curves and surfaces. Here $|\cdot|$ denotes the number of entities in a particular entity set. E^p and F^p are the total number of edges and faces in the 3D mesh at the previous level, respectively. 32

5.1 Comparison of average stencil sizes for WALF and FAH-WALF. . . 81

6.1 The times in seconds for the refinement, resolution of shared entities using the merge and optimized algorithms as the number of processors increases. 103
6.2 Comparison of storage requirements in kilobytes of Hierarchical AHF and libMesh for a mesh in various stages of refinement on 32-bit and 64-bit architectures. On 64-bit architecture, one may store Hierarchical AHF with 32-bit integers or 64-bit integers. 110
6.3 Mesh sizes of the sequence of sphere and torus meshes used for convergence studies. 116
6.4 Numbers of ghost layers for WALF in parallel. 118

Chapter 1

Introduction

In the numerical solution of complex partial differential equations (PDE's) using finite element methods for unstructured meshes, the two most computationally intensive steps are mesh generation and linear solvers. An initial coarse mesh representing the computational domain might not be of sufficient resolution to get meaningful results out of the discretizations for physical scales that might be present. As a result, the capability to refine a mesh is an essential part of any simulation process. Additionally, it is well known that multi-level methods such as geometric multigrid methods (GMG) can theoretically deliver optimal time complexity for solving sparse linear systems from PDE discretizations. Thus, it would be advantageous to use nested multi-level (i.e., hierarchical) meshes to achieve high-degree of verifiable accuracy and computational efficiency, especially in the context of large-scale parallel computing, as both the number of processors and the mesh resolution increase.

Uniform mesh refinement (UMR) provides a simple and efficient way to generate such hierarchies via successive refinement of the mesh at a previous level. It also provides a natural hierarchy via parent and child type of relationship between entities of meshes at different levels that enables inter-level queries. This hierarchy also supports computation of projection operators of GMG between levels. While UMR is a relatively simple process, it is by no means trivial especially in a parallel setting. Notable challenges include maintenance of mesh quality, multi-level and multi-degree refinement, and data structure and software design.

Meanwhile, to obtain accuracy some regions need to be refined to reduce discretization errors while other regions require finer models to approximate. Adaptive mesh refinement allows more efficient numerical simulations by increasing the computational effort near interesting features of the solutions [10, 11, 14]. AMR has gradually become a vital step in large-scale numerical simulations since it optimizes the relationship between accuracy and computational effort. One aspect of the refinement strategy is whether it requires the refined mesh to be conformal or not. A mesh is said to be *conformal* if the pairwise intersection of any two entities is either a lower-dimensional entity or is empty. Otherwise, a mesh is non-conformal. The conformal requirement will make no change to the underlying data structure for the mesh and the formulation of numerical algorithms. A considerable amount of work has been done in this area [15, 17, 20, 97]. However, to preserve conformity, some procedures need to be applied which would probably deliver a finer mesh than needed, or even potentially affect the overall mesh quality which is crucial for the linear system in FEM [104]. This drives the research on refinement strategies allowing non-conformal meshes, i.e. hanging nodes, in [15, 67, 73, 104]. Non-conformal refinement will incur extra work in the PDE solver part, but it will be much easier for the unification of *hp*-adaptivity for finite element method [15, 16]. Here *hp*-adaptivity means that both the mesh size *h* and the degree *p* of the approximating piecewise polynomials are adapted.

FEM originated as a second-order accurate method, but for several decades, researchers have been exploring high-order variants. In this dissertation, we use the term "high-order" to refer to any method of third or greater order accuracy. Some high-order methods include isoparametric FEM [42], hp-FEM [37], discontinuous Galerkin method [33], spectral element method [26], and isogeometric analysis [56]. The element quality requirements for high-order methods are even more stringent. Despite the fact that these methods can reach exponential convergence under appropriate conditions, high-order methods have remained largely confined to academic study and has yet to make much of an impact in industry [116]. This is due to many reasons, not the least of which is that high-order schemes are generally less robust [120] and generating good quality meshes for high-order methods is still not fully resolved [90].

Geometry plays an important role in solving PDEs. The use of curved elements becomes mandatory for using high-order approximations [80]. Even for low-order methods, when one tries to achieve high accuracy via mesh refinement, the triangulation must be refined along the curved boundary [34, 77]; otherwise the geometric error will counteract the effect of mesh refinement. High-order reconstruction of the boundary geometry is effective in preserving the accuracy of solution under mesh refinement. The numerical results indicate that geometric error matters in solving PDE.

This dissertation presents an integrated framework for these problems. First, we develop parallel uniform refinement-based algorithms to generate multi-degree, multi-dimensional and multi-level meshes from coarse unstructured meshes [94]. The generated mesh hierarchies can be used for a variety of purposes, such as convergence studies, multilevel methods, generating large meshes in parallel to overcome IO bottlenecks, etc. While the multi-degree refinement allows achieving uniformly greater resolution faster, the multi-dimensional refinement preserves the hierarchy over explicit lower dimensional entities such as curves in surfaces, or surfaces embedded in volumes.

Second, we develop an array-based mesh data structure to support adaptive mesh refinement and derefinement[1]. It generalizes AHF [40], which provides efficient mesh queries and modification. The array-based mesh data structure has many advantages in the context of numerical simulations, in terms of more compact memory footprint, better interoperability with simulation codes, better efficiency on modern computer architectures with deep memory hierarchy, and relative simplicity and higher efficiency for parallel implementations. However, it is more challenging to support adaptive mesh refinement with array-based mesh data structures, which require dynamic creation and deletion of entities [122].

A key aspect of the refinement algorithm is the positioning of the new vertices as entities are refined. A commonly used strategy is linear point projection. However, using linear point projection for the new vertices compromises the accuracy of the geometry and in turn that of the finite element solver. To address this issue, we take advantage of high-order boundary reconstruction strategy called *WALF,* as described in [60, 61]. We also propose two extensions of the WALF framework. First, we introduce a Hermite-style weighted-least squares formulation, to take both point locations and surface normals as input for reconstruction. A key advantage of this approach is that it allows much more compact stencils for local fittings. As a result,

[1]We use the term “derefinement” instead of “coarsening” because the algorithm would only undo the refinement selectively, and it would not coarsen beyond the original mesh.

it can achieve higher accuracy, especially for relatively coarse input meshes. We prove the consistency and stability of the proposed method, and describe robust numerical method for solving it. Second, we also introduce Hermite-style high-order reconstruction of ridge curves. In addition, we introduce a new blending technique to ensure G^0 continuity along sharp ridges and corners, while assuring high-order accuracy.

The reminder of the dissertation is organized as follows. In Chapter 2, we present some background information and recent development of related data structures and methods. In Chapter 3, we introduce our work on parallel hierarchical uniform refinement which was developed in collaboration with Dr. Navamita Ray, Dr. Iulian Grindeanu, Dr. Vijay Mahadevan from Argonne National Laboratory. In addition we also developed a high-order surface reconstruction module under MOAB to handle curved boundaries. Chapter 4 describes our effort on conformal and non-conformal adaptive mesh refinement on unstructured meshes. In Chapter 5, we discuss the problem of high-order surface reconstruction. We develop a robust reconstruction method, which delivers high-order accuracy when sharp features present. This was developed in collaboration with Dr. Navamita Ray. In Chapter 6, we present the results of some numerical experiments with mesh refinement and high-order surface reconstruction. Finally, Chapter 7 concludes the paper with a discussion.

The main contributions of this thesis are as follows. First, we developed parallel hierarchical uniform mesh refinement under the array-based unstructured mesh framework "Mesh Oriented dAtaBase" a.k.a MOAB [107]). In this work, we developed a template-based refinement strategy for subdividing each entity into smaller entities to support multi-degree refinement patterns. To support hierarchy generation and efficient mesh traversals, we extended the array-based half-facet (AHF) data structure [40]. In addition, we developed efficient parallel communication strategies to resolve shared entities along partition boundaries after refinement. The developed mesh hierarchy generation supports 1D (edges), 2D (triangles, quadrilaterals), and 3D (tetrahedral, hexahedral) meshes and mixed-dimensional meshes.

Second, we introduced a simple data model for adaptive mesh refinement with hierarchical structure. Our data model is easy to implement and is efficient in both memory and computational cost. The data structure facilitates both straightforward refinement and derefinement operations, and also allows both conformal and non-conformal meshes. In addition, a generic adaptive mesh refinement (AMR) framework on top of Hierarchical AHF is developed and a prototype is implemented for both 2D triangular and 3D tetrahedral meshes.

Third, to handle complex geometries, we developed a discrete geometry module in MOAB, which provides high-order surface projection. The discrete geometry module provides high-order boundary reconstruction strategies, based on WALF. Besides, we introduced the *FAH-WALF* method, Feature-Aware Hermite-style WALF, which extended the WALF method [61]. By design, FAH-WALF is useful when an accurate, instead of "exact" geometry is needed, and when accessing the CAD software may be inconvenient. In particular, it is especially attractive for high-order finite element methods, mesh refinement, mesh smoothing, mesh adaptivity, both in serial and parallel, especially for the solutions of PDEs. An important advantage of the FAH-WALF methodology, compared to WALF, is that we can use high-order reconstruction, such as fifth- or sixth-order reconstructions, instead of only second or third-order reconstructions, on a relatively coarse mesh, while ensuring the accuracy and stability of the method.

Chapter 2

Background and Related Work

Mesh data structures are fundamental to meshing algorithms and mesh-based numerical methods. The underlying data structure strongly influences the overall performance of the algorithms or simulations, since it is used to perform all the mesh-based combinatorial operations and as a result has been investigated since the inception of mesh generation and computational geometry. We review some terminology before describing our data structures and mesh frameworks. Adaptive methods for numerical PDEs have been an active research area since the late 1970s [10, 11] and are widely used in practice nowadays, to balance accuracy and computational efficiency. We briefly review the mesh adaptation methodology for numerical PDEs.

In addition, we briefly review the weighted least squares approximation on unstructured surface meshes using point based local polynomial fittings, the criteria for selecting the neighborhood, the weighting strategy, and the numerical linear algebra techniques for solving them robustly. We also review the WALF approach for blending the local fittings into a G^0 surface.

2.1 Array-based Half-Facet (AHF) Data Structure

There are a number of mesh data structures such as entity-based, boundary representations, corner table, radial-edge, winged, half- edge/face, incidence graphs, etc., that are used for mesh representation and queries. The two data structures that are relevant in our context are the half-edge and half-face data structures. The half-edge data structure is for 2D and surface meshes. It uses edge as the core object where the edge within each face is called a directed or half-edge. Typical implementations (e.g., CGAL [44, 66], OpenMesh [23] and Surface_Mesh [103]) store mappings from each half-edge to its opposite half-edge, its previous and next half-edge within its face, its vertices, its incident face, as well as the mapping from each vertex and each face to an incident half-edge. More compact representations, such as [7], can be obtained by storing only the mapping between opposite half-edge, optionally the mapping from each vertex to an incident half-edge, along with the element connectivity. The half-edge concept was generalized to half-faces (as in [7] and [72]) for volume meshes where half-faces refer to the oriented faces within a cell. These basic half-edge and half-face data structures are simple and are restricted to oriented, manifold meshes (with or without boundary) in 2-D and 3-D, respectively.

In [40], an efficient, compact and general array-based half-facet (AHF) mesh data structure with support for mixed-dimensional meshes, which may be non-manifold and/or non-oriented was proposed. The core object of AHF is *half-facet* as defined previously and is represented as an *implicit entity.* The concept of *sibling half-facets* unifies the half-vertex, half-edge, and half-face data structures for 1-D, 2-D, and 3-D meshes, which may be manifold or non-manifold with boundary. The

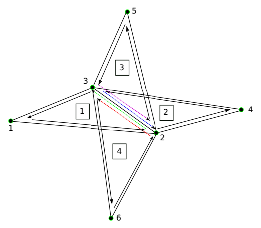
AHF data structure consists of two key maps:

- sibling half-facets (*sibhfs*): The mapping between the sibling half-facets using a cyclic linked list.
- vertex to half-facet (*v2hf*): The map of each vertex to its incident half-facet. This map provides an anchor for each vertex to its locality in the mesh.

An example of the AHF maps for a non-manifold mesh is illustrated in Figure 2.1(a). For $d \geq 2$, AHF provides a compact representation, since the intermediate dimensional entities are not stored but referenced implicitly. This data model stores the above two maps for each dimension in a modular and self-contained manner and hence supports mixed-dimensional meshes, which may be composed of sub-meshes of 1-D, 2-D, and 3-D. Figure 2.1(b) shows a diagram of a typical half-facet data structure, where the half-vertices and half-edges are only required for explicit edges and faces in the mesh, respectively. In addition, this data model can also be used for meshes with high-order elements (such as six-node triangles or 10-node tetrahedra), where the mid-edge, mid-face or mid-cell nodes do not affect the definition and identification of the half-facets. We use the AHF as the underlying mesh data structure for developing the refinement algorithms.

2.2 Mesh Oriented datABase (MOAB)

To be of any practical use to numerical simulation workflows, the mesh framework needs to support a wide range of functionalities such as efficient local mesh traversals for matrix assemblies, boundary extraction for boundary conditions, efficiently support adjacency and connectivity queries, representation of mesh data, etc., in



element	connectivity		
1	1	2	3
2	2	4	3
3	2	5	3
4	2	3	6

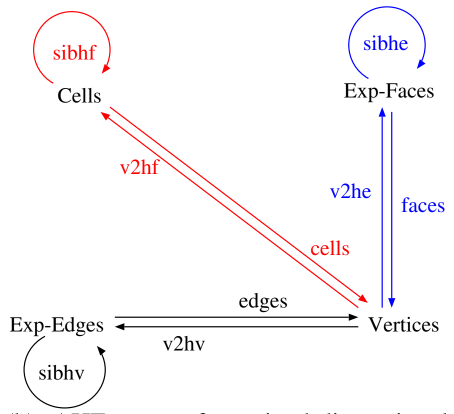
sibling half-edges

element	sibhes		
1	$\langle 0,0\rangle$	$\langle 2,2\rangle$	$\langle 0,0\rangle$
2	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 3,2\rangle$
3	$\langle 0,0\rangle$	$\langle 0,0\rangle$	$\langle 4,0\rangle$
4	$\langle 1,1\rangle$	$\langle 0,0\rangle$	$\langle 0,0\rangle$

vertex to half-edges

vertices	v2he
1	$\langle 1,0\rangle$
2	$\langle 3,0\rangle$
3	$\langle 4,1\rangle$
4	$\langle 2,1\rangle$
5	$\langle 3,1\rangle$
6	$\langle 4,2\rangle$

(a) AHF maps for a non-manifold mesh.



(b) AHF maps for mixed-dimensional meshes.

Figure 2.1: (a) An example of the AHF maps stored for a non-manifold triangular mesh. (b) Typical AHF for mixed-dimensional meshes is composed of half-vertex (black, for explicit edges only), half-edge (blue, for explicit faces only), and half-face (red) data structures.

a parallel setting. Over the past years, a number of such frameworks have been developed e.g., FMDB [101], MSTK [49], libMesh [68], etc, with various mesh representations addressing application specific needs. In such implementations, the entities are represented as “objects” explicitly, and pointers (or handles) are used to refer to these explicit objects. In our work, we choose to use array-based, pointer-free implementations for a number of reasons. First, using arrays can lead to faster memory access, fewer cache misses and hence better efficiency. Secondly, in an array-based implementation, we can treat intermediate dimensional entities (such as half-facets) as implicit entities, and reference them without forming explicit objects. This can lead to significant savings in storage, especially on computers with 64-bit pointers. In addition, array-based implementations also offer better interoperability across application codes, different programming languages, and different hardware platforms (such as between GPUs and CPUs).

Several array-based mesh libraries have been developed, including STK [41], PUMI [57], and MOAB [107]. MOAB, the Mesh Oriented datABase is a mesh data representation designed to support a range of mesh related operations, such as memory-efficient mesh representation, mesh querying, and representation of application specific data. The AHF data structure was implemented in MOAB to support query-intensive algorithms, especially those involving non-vertex/higher-dimensional adjacency queries and was found to be extremely efficient (in some cases over two orders of magnitude improvement [40, 93]) over the native data structures. MOAB is also a parallel framework and supports spatial domain-decomposed view of a parallel mesh where each subdomain is assigned to a processor, lower-dimensional entities on interfaces between subdomains are shared between processors, and ghost entities can be exchanged with neighboring processors. We develop our hierarchical mesh generation algorithms on top of MOAB's parallel framework. In Section 3, we describe the three key components of the proposed hierarchical mesh generation algorithm.

2.3 Adaptive Mesh Refinement for Numerical PDEs

Adaptive mesh refinement applies strategies to optimize the relation between accuracy and computational effort. In particular, adaptive finite element method (AFEM) based on the local mesh refinement has the following loops of form:

$$\textbf{SOLVE} \rightarrow \textbf{ESTIMATE} \rightarrow \textbf{MARK} \rightarrow \textbf{REFINE} \tag{2.3.1}$$

to iteratively improve the accuracy of the numerical approximation. For an introduction to the theory of AFEM, see Nochetto et al. [88].

The step **ESTIMATE** needs *a posteriori* error estimator without prior knowledge of the exact solution. In numerical solution of PDEs, often the overall accuracy of numerical approximation is deteriorated by local singularities such as, e.g, singularities from boundary layers or sharp shock waves. One remedy is to refine regions where the solution is less regular. Moreover, *a posteriori* estimates of solution accuracy would be desirable when *a priori* error is not available. These arise the need of an error indicator, which could be extracted from the numerical solutions. In particular, the development of adaptive finite mesh refinement, h-version and hp-version, requires reliable and computable *a posteriori* error estimator. Since I. Babuska and W. Rheinboldt's pioneering work [10–12] on explicit error estimation in late 1970s, lots of effort has been devoted to the derivation of effective error estimator. Often estimators with local natures are preferred since they could indicate the need of local mesh refinement or increase of polynomial degree. Kelly [64], Ladeveze and Leguillon [74] used the idea of solving element by element complementary problems together with the important concept of constructing equilibrated boundary data to obtain error estimates. Demkowicz [36], Bank and Weiser [19] developed *a posteriori* error estimation based on the element residual method. Other methods based on interpolation or extrapolation could be found in [2, 123]. Surveys of the literature on *a posteriori* error estimation and analysis could be found in [3, 4, 100, 115].

Given the error estimation, certain marking strategy should be applied to collect elements in the mesh for further refinement or for increasing polynomial degree. The step **Mark** is important for the convergence and optimality of AFEM, as noted by Stevenson [106]: “any marking strategy that reduces the energy error relative to the current value must contain a substantial bulk of $\mathscr{E}_{\mathscr{T}}(U, \mathscr{T})$, and so it can be related to Dörfler Marking”. The most popular marking strategies are Maximum Strategy, Equidistribution Strategy [43], Dörfler’s Strategy [39]. Morin [87] gave a sufficient and essentially necessary condition on marking for the convergence of the finite element solutions to the exact one. This condition is not only satisfied by Dörfler’s strategy, but also by the maximum strategy and the equidistribution strategy.

The **REFINE** step refines all marked elements of a given initial triangulation. In 2D, during the mid of the 1980s Rivara introduced an effective mesh refinement algorithm based on longest edge bisection [96]. Mitchell developed a recursive algorithm for the newest vertex bisection [84]. Bank adopted regular refinement with selected temporary bisections [8], which was used in software package PLTMG

[17]. In the beginning of 1990s, Rivara and Levin extended the longest edge refinement algorithm to tetrahedra meshes [97]. However, it is not known that if this algorithm would degrade the mesh quality. At the same time, Bänsch generalized the newest vertex bisection method to 3D [20], which preserves the mesh quality under refinement. Similar approaches are developed by Liu and Joe [79], Arnold et al. [9]. Moreover, Kossaczky [71] derived a recursive variant of Bänsch's algorithm, with bisection rule for tetrahedra using local order of vertices and element type. This concept is convenient for implementation and generalization to any space dimension. Besides, anisotropic adaptive mesh refinement could be more desirable for physical problems exhibit anisotropic phenomena; see [32, 78] and the references therein. For a more completed discussion of mesh refinement, see [63, 88].

The **SOLVE** step is critically important for the optimal efficiency of AFEM. For a quasi-uniform mesh with hierarchy, V-cycle multigrid and BPX-preconditioner [24] can obtain desired accuracy with a number of operations proportional to degree of freedom. For a multiple-level method on an adaptively refined mesh, performing smoothing only on the newly added nodes would be crucial for optimal complexity. Such local multiple level methods include multiple level adaptive technique [25], Rivara's local multigrid method on adaptive triangular girds [95], and Bank's HB multigrid [18] on the red-green regular-refined mesh. In particular for graded bisection meshes, see [31, 121], which have linear complexity.

Convergence of the loops of AFEM is well studied theoretically for Poisson equation and its variants. The pioneering work started from Dörfler [39], which introduced the Dörfler's marking Strategy and proved the strict energy error reduction with a fineness assumption on the initial mesh. Then it was followed by Morin et al. [85, 86] who proved convergence without restriction on initial mesh.

Mekchay and Nochetto [82] proved a contraction property of AFEM for general elliptic PDEs. For more recent result, see [87, 102]. Binev, Dahmen and DeVore [22] (requiring coarsening), Stevenson [106] (no requirement of coarsening)and Cascon [29] gave important optimal error decay result, or the optimal cardinality of AFEM. Results of the convergence on adaptive nonconforming and mixed FEM could be found in [27, 28, 30]. For other equations, such as Stoke equation, due to the lack of Galerkin-orthogonality or quasi-orthogonality for such saddle problem, techniques used for typical AFEM analysis could not be applied; see discussions in [21, 55, 69, 70].

The convergence rate of the p version of the finite element method is exponential when the solution has no singularities. When singularities are present, the performance of p version FEM depends on the mesh. “Performance will be better when the mesh has been graded near singularity” [46]. This suggests that hp-adaptivity would be the optimal strategy. The results in [52, 53] suggest that the convergence rate is exponential if proper hp-adaptivity is applied, but the results rely on generating correct initial mesh, which is only known for model problems. Techniques of optimal mesh “refinement” strategies are still underdeveloped. Most of the adaptive finite element algorithms simply subdivide the elements where *a posteriori* error is large, or an h-refinement is applied. As it is well known, if the exact solution is sufficiently smooth, a p-refinement typically is much more effective in improving accuracy. Automatic hp-adaptivity has been an active research area in the high order finite element technology. Demkowicz and his coauthors developed fully automatic hp-adaptivity for elliptic problems and demonstrated optimal convergence rate predicted by the theory of hp method [35, 37]. Šolín et al. designed an automatic adaptivity algorithm based on arbitrary-level hanging nodes and local element projections [104]. However, as mentioned in [16], “there is no simple and widely accepted *a posteriori* indicator applicable to an already computed solution that would tell us whether we should refine any given cell of a finite element mesh or increase the polynomial degree of the shape functions defined on it.” More discussion in [54] and the reference cited therein.

As mentioned in [37], most commercial implementations of hp-FEM rely on a-$priori$ information about corner and edge singularities, or boundary layers, and they generate an initial mesh like the geometrically graded meshes of Babuska, or Shiskin type mesh for handling the boundary layers. Once the mesh is known, uniform p-refinement are used, since generally for linear elliptic boundary-value problem with piecewise analytic coefficients, on a domain with a piecewise analytic boundary surface, the solution will be analytic everywhere except in the neighborhood of singularities in the data where h-refinement could be applied. If the nature of the singularity is known *a priori*, these techniques could be effective, and optimal meshes may deliver exponential rates of convergence. Generating mesh aware of singularities and boundary layers is crucial.

If the regularity results are not available, it is nearly impossible to get an optimal mesh. Then p-refinement may lead to meshes that deliver results worse than h-adaptive method; see [37] and references there in. Various algorithms have been developed with the aim to estimate the optimal choice of hp-refinement. The Texas Three Step strategy [89] performs h and p-refinement alternately, but it does not lead to optimal results. The method in [5] monitors the local h-convergence rates to choose h or p refinement. The authors in [92] introduced the idea of obtaining optimal mesh via minimizing a local projection error for a reference solution. For related work, we refer to [13, 37, 98]. Some methods estimate the smoothness of the solutions. For example, Mavriplis [81] proposed the method of calculating the decay rate of Legendre expansion coefficients of the solution to determine whether the solution is locally smooth or not. On the other hand, local Sobolev regularity index could be approximated and further used as indicator for the choice of h and p-refinement [6]. The authors in [54] extended this idea and also gave a nice survey of other methods and more recent development. For simple representative problems, some of these strategies illustrate exponential rates of convergence [6, 54].

A vast amount of FEM software packages have been developed; see [1] for a complete list. However, implementations of hp finite element method are hard, and a few of them are accessible, such as [15, 35, 67, 105]. deal.II [15], a C++ finite element library, supports hp-finite element in 1D, 2D (quadrilaterals) and 3D(hexahedra), and allows hanging nodes introduced in hp-refinement. Hanging nodes will be eliminated according to continuity constraints [16]. Likewise, libMesh [67] is also a C++ library for serial/parallel adaptive algorithms. libMesh supports h-refinement, coarsening (by h restitution of subelements), and h-refinement with uniformly high degree elements. Moreover, the developmental branch of libMesh now supports adaptively p refined and hp refined meshes with some element types. Hermes2D [105] supports adaptive FEM in 2d based on algorithm in [104]. According to the author's knowledge, Hermes does not have 3D hp-FEM accessible to public now but the algorithm has been developed in [73].

2.4 Weighted Least Squares and High-order Surface Reconstruction

Surface meshes and their manipulations are critical for geometric modeling, meshing, numerical simulations, and many other related problems. Some example problems that involve manipulating surface meshes include mesh generation and mesh enhancement for finite element or finite volume computations [48], mesh smoothing in ALE methods [38], and mesh adaptation in moving boundary problems [59]. In many of the applications, a continuous CAD model may not be available. Instead, only a surface mesh, typically with piecewise linear or bilinear faces, is available.

We consider the problem of reconstructing a highly accurate, continuous geometric support from a given surface mesh. We refer to this problem as *high-order surface reconstruction* (or simply *high-order reconstruction*). By "high-order," we mean the method should be able to deliver more than second order accuracy, and preferably fifth, sixth, and even higher order, compared to just first or second order accuracy of the traditional techniques. The high-order reconstruction is important in geometric modeling [118] and meshing [61] for scientific applications, computer graphics [47], etc.

In [61], four requirements were posed for high-order reconstruction:

Geometric accuracy: The reconstruction should be accurate and asymptotically convergent to the exact surface to certain order under mesh refinement.

Continuity: The reconstructed surface should be continuous to some degree (e.g., G^0, G^1, or G^2 continuous, depending on applications).

Feature preservation: The reconstruction should preserve sharp features (such as ridges and corners) in the geometry.

Stability: The reconstruction should be numerically stable and must not be oscillatory under noise.

Two methods, called *Weighted Averaging of Local Fittings* (*WALF*) and *Continuous Moving Frames* (*CMF*), were also proposed in [61], for reconstructing a feature-preserving, high-order surface from a given surface mesh. Both methods were based on weighted least squares approximations and piecewise polynomial fittings, and they could achieve third- and even higher order accuracy, while guaranteeing global G^0 continuity. In contrast, most existing methods could achieve only first- or second-order accuracy. Between WALF, and CMF, the WALF was preferred for its simplicity and higher efficiency, and it has been shown to be particularly effective to moderately fine input meshes. However, WALF also had two limitations. First, if the input mesh is relatively coarse, then the methods may be inaccurate, due to the lack of points in the stencil. While some safeguards were added against oscillations by degrading to lower-order accuracy [61], the resulting accuracy would not be optimal. Second, the methods could not guarantee G^0 continuity along sharp features (ridges and corners), so the reconstructed surface may not be "watertight," which can be problematic if one need to sample some points very close to the sharp features.

For more systematic treatment of sharp features, we amend the preceding requirements with the following additional considerations for the reconstructed surface at and near sharp features:

High-order Feature Reconstruction: The ridge curves must be reconstructed to high-order accuracy. In addition, the reconstructed surface near the sharp features must also achieve high-order accuracy.

Continuity at Features: The reconstructed surface must satisfy G^0 continuity next to sharp features.

Robustness: The technique must be robust and numerically stable. In particular, it must not involve computing intersections of high-order surfaces, which are notoriously unstable.

In this thesis, we propose two extensions of the *WALF* framework. First, we introduce a Hermite-style weighted-least squares formulation, which takes advantage of both point locations and surface normals for surface reconstruction. A key advantage of this approach is that it allows much more compact stencils. As a result, it can achieve higher accuracy, especially for relatively coarse input meshes. We prove the consistency and stability of the proposed method, and describe robust numerical method for solving it. Second, we introduce a new blending technique to ensure G^0 continuity along sharp ridges and corners, while assuring high-order accuracy.

Using both point and normal is not a new idea. It is analogous to Hermite interpolation in numerical analysis. For surface modeling, Walton [118] defined an approach to reconstruct G^1 continuous surfaces. Another example is the curved PN-triangles [117]. Other related work includes [50], in which points and normals are used together for estimating curvatures. However, to the best of our knowledge, our proposed technique is among the first that leverage both points and normals to deliver high-order accuracy in surface reconstructions. As a result, it provides a valuable alternative to the traditional CAD models, such as NURBS [45] and T-splines [99], especially for scientific applications that require accurate instead of exact geometry, such as the solutions of PDEs.

Note that another approach for surface reconstruction, which is orthogonal to high-order reconstruction, is high-degree-continuity reconstruction, such as the moving least squares (MLS) approximation for point clouds [47, 76]. The MLS is theoretically G^∞ when global weighting schemes are used. However, note that in practice, the small weights in MLS are truncated to zeros for efficiency, leading to loss of continuity in MLS. More importantly, MLS provides no guarantee the order of accuracy. Although its convergence was conjectured in [76], in practice it is found to be nonconvergent even for simple geometries such as a torus [61]. In addition, it is difficult to treat sharp features in the framework of MLS. In contrast, FAH-WALF, similar to WALF, achieves only G^0 continuity, but they ensuring high-order accuracy in the normals and curvatures along edges and at vertices, and can treat sharp features in a systematic fashion.

Chapter 3

Hierarchical Uniform Refinement for Unstructured Meshes

The uniform refinement based mesh hierarchy generation has three key design components. First, entity type and degree-specific refinement templates are defined that are used to subdivide an entity into its children. The templates are also used to update the underlying data structures for all the new children. The second key design component is the use of array-based half-facet data structures for efficient mesh traversal during refinement. Finally, the newly created meshes are stored in level-wise contiguous array to provide memory compactness.

3.1 Multi-degree Refinement Templates

The standard refinement strategy divides each d dimensional entity to $1 \rightarrow 2^d$ subentites. However, a desired mesh resolution can be reached much faster if a higher degree of refinement is used. In such cases, the length of the mesh hierarchies is also small, which might be preferable from some multigrid methods such as GMGs. Another motivation to use high degree refinement patterns is that it allows straightforward extension to high-order entities and thus support *hp*-refinements. By multi-degree refinement we mean a refinement pattern following vertex positions analogous to high-order (degree p where $p \geq 2$) Lagrangian elements. Thus, for a d dimensional entity, a degree p refinement divides the entity to $1 \rightarrow p^d$ subentites. We support prime number degrees as any higher degree refinements can be obtained by applying a sequence of such prime degree refinements. Table 3.1 lists the currently supported degrees for each dimension. We note that in uniform refinement the same modification operation of dividing an entity into a specific number of entities is applied to each entity of the mesh. Thus defining static templates w.r.t a reference entity that contains certain entity type and degree specific information could aid the refinement process. The template mainly stores the following information for each entity type and supported degree of refinement :

- *Numbering convention of new vertices*: The new vertices are assigned local indices through which they are uniquely identified within a reference entity.
- *Connectivity of new child entities*: The connectivity of the new entities using the local indices of the new vertices. Each such child is also uniquely identifiable by a local id w.r.t reference.
- *Half-facet maps for new child entities*: The local sibling half-facet (**sibhfs**) and vertex to incident half-facet (**v2hf**) maps are stored.

Figure 3.1 shows the templates for triangle and quadrilateral reference entities for degrees 2 and 3. The numbering convention of the reference entity follows the

MOAB canonical numbering. Apart from these three pieces of data, the template also stores some other auxiliary information to aid tracking of new vertices introduced on the entity boundary to avoid vertex duplication during refinement. An example of a degree 3 refinement template for a reference triangle is illustrated in Tabl 3.2.

Except for tetrahedron entity, the templates are static for all other entities (triangles, quads, hexes) in the sense that they do not need to refer to the physical entity in order to refine the parent entity. The refinement schemes for a tetrahedron involves division into smaller congruent tetrahedra and octahedrons which are subsequently divided into four tetrahedra. Figure 3.2 shows how refining a tetrahedron leads to congruent smaller tetrahedra and octahedra. The right side of the figure shows the three possible diagonal choices for each such octahedron to be divided into four tetrahedra. To deliver a good mesh quality, each such octahedra is divided into sub-tetrahedra by connecting the shortest diagonal [112]. Thus, for higher-degree refinements, each octahedron has to find the shortest diagonal using the physical entity. However, we note that though the smaller tetrahedra obtained by tessellating an octahedron are not congruent to the parent tetrahedron, all the octahedra are congruent to each other. Therefore, the shortest diagonal is unique and hence each octahedron would be divided using the same pattern. We use this fact to define three static templates for each degree. During refinement, each tetrahedron makes a choice of the appropriate template based on the smallest diagonal of the physical tetrahedron. Currently, the template generation is not automatic. Effort is being made to make it automatic so that higher-degrees, such as 7, 11, etc., can also be supported.

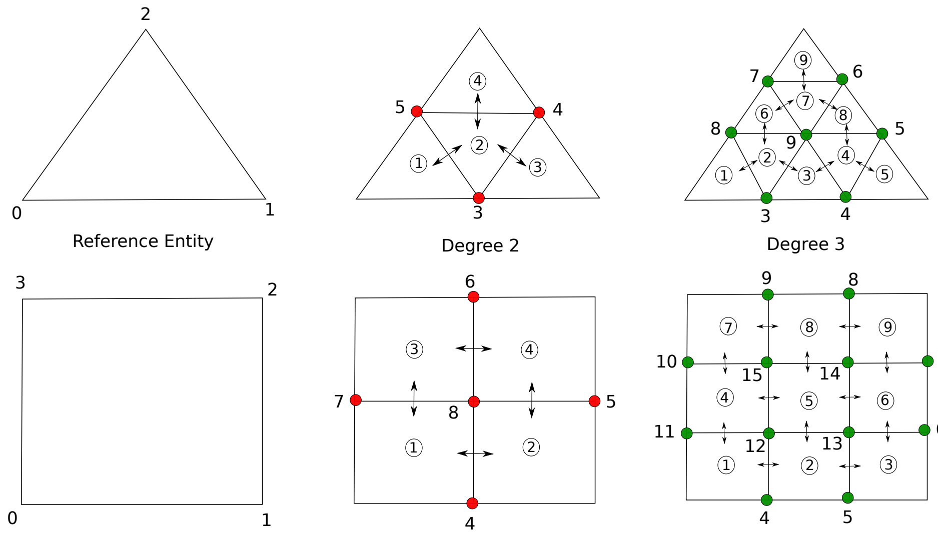


Figure 3.1: The refinement templates defined for degree 2 and 3 refinements over a reference triangle and quadrilateral entity. The local vertex order of the reference entities follow MOAB conventions [108]. The new vertices to be introduced during refinement are assigned local ids w.r.t. to the reference. Similarly, the children entities are assigned local order. The AHF maps are created for the refined entities in terms of their local ids.

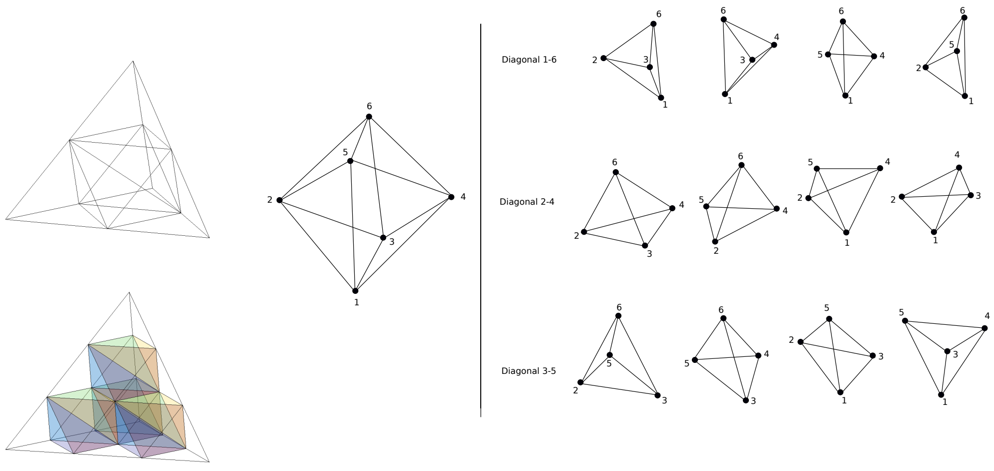


Figure 3.2: Left: A tetrahedron is divided into congruent smaller tetrahedra and octahedrons. Middle: A reference octahedron. Right: Each octahedron can be further subdivided into four tetrahedra in three possible ways depending on which diagonal is chosen.

Table 3.1: The degrees of refinement currently supported and the corresponding number of children.

Dimension	Degrees	#Children
1	2, 3, 5	2, 3, 5
2	2, 3, 5	4, 9, 25
3	2, 3	8, 27

Table 3.2: An example for the degree 3 template of a triangle reference entity. Children entity connectivity and the template $v2hf$ and $sibhfs$ maps.

Child Entity Connectivity	
1	⟨0, 3, 8⟩
2	⟨3, 9, 8⟩
3	⟨3, 4, 9⟩
4	⟨4, 5, 9⟩
5	⟨4, 1, 5⟩
6	⟨8, 9, 7⟩
7	⟨9, 6, 7⟩
8	⟨9, 5, 6⟩
9	⟨7, 6, 2⟩

sibhfs		
⟨0, 0⟩	⟨2, 2⟩	⟨0, 0⟩
⟨3, 2⟩	⟨6, 0⟩	⟨1, 1⟩
⟨0, 0⟩	⟨4, 2⟩	⟨2, 1⟩
⟨5, 2⟩	⟨8, 0⟩	⟨3, 1⟩
⟨0, 0⟩	⟨0, 0⟩	⟨4, 0⟩
⟨2, 1⟩	⟨7, 2⟩	⟨0, 0⟩
⟨8, 2⟩	⟨9, 0⟩	⟨6, 1⟩
⟨4, 1⟩	⟨0, 0⟩	⟨7, 0⟩
⟨7, 1⟩	⟨0, 0⟩	⟨0, 0⟩

v	*v2hf*
0	⟨1, 0⟩
1	⟨5, 1⟩
2	⟨9, 2⟩
3	⟨3, 0⟩
4	⟨5, 0⟩
5	⟨8, 1⟩
6	⟨9, 1⟩
7	⟨6, 2⟩
8	⟨1, 2⟩
9	⟨8, 0⟩

3.2 Multi-dimensional Mesh Data Structure

During refinement, one of the key tasks is to avoid introducing duplicate vertices for shared boundaries between entities. This requires frequent calls to adjacency routines to get entities connected through entity boundaries. Since AHF stores the half-facet maps between entities, these types of queries are the most efficient ones it can support. As a result, it is natural to use AHF as the underlying data structure. As each level of the mesh hierarchy is generated by refinement of the previous level, it is necessary to update the appropriate AHF maps for the new level so that the new mesh can be queried later. The process of updating the AHF maps for the new level is aided both by the refinement templates (defined in the previous subsection) and also by the fact that the topology of the domain does not change during refinement. Thus, children of manifold entities remain manifold for all levels, and no new non-manifold entities are introduced. The AHF maps are updated in two stages:

1. *Update maps for children of an entity*: After an entity is refined, the sibling half-facet **(sibhfs)** and vertex-to-incident half-facet (**v2hf**) maps are updated only for the child entities of the working entity by using the same from the refinement templates.

2. *Update maps between children of parent siblings*: After all the entities of the mesh have been refined, the maps are now updated to connect the child entities incident on boundaries of the parents.

Figure 3.3 illustrates the above two steps during refinement of a triangle mesh with two entities. Algorithms 1 and 2 outline the procedure for updating local and global AHF maps. For mixed-dimensional meshes, the maps for each lower dimensional

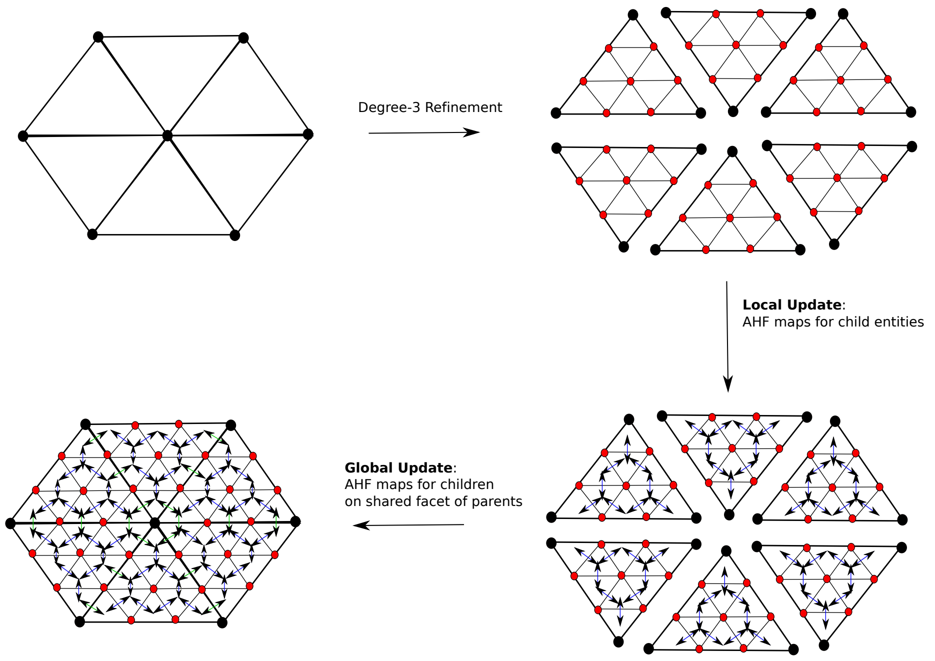


Figure 3.3: Updating the AHF maps for the refined mesh takes place in two stages. First, maps for the children of each triangle are updated. After, both the triangles have been refined, the maps are now updated to connect the children entities sharing a parent facet.

submesh are updated in a similar manner.

3.3 Multi-level Mesh Storage

An array-based mesh storage leads to increased efficiency, but it requires careful consideration to maintain it for operations involving a change in the contiguity of the memory space. By virtue of the uniform refinement of the mesh, it is possible to estimate the total number of entities that will be created for a given degree of refinement. Table 3.3 shows the estimates for new entities created after a single level of refinement for a 3D mesh with embedded curves and surfaces. Once the storage

Algorithm 1 Updating AHF maps for children of an entity

```
Require: : childEnts: ordered child entities of a parent, childVerts: ordered new
    vertex indices of childEnts, refTemplate: refinement template for entity type
    and degree
Ensure: sibhfs: update sibling half-facets for childEnts, v2hf: update vertex to
    half-facet for childVerts
 1: for each vertex $v$ in childVerts do
 2:     if v2hf($v$) = 0 then
 3:         $cid \leftarrow$ child id from refTemplate$\rightarrow$$\mathbf{v2hf}$;
 4:         $lid \leftarrow$ half-facet id from refTemplate$\rightarrow$$\mathbf{v2hf}$;
 5:         v2hf($v$) = (childEnts[$cid$], $lid$);
 6:     end if
 7: end for
 8: for each child $c$ in childEnts do
 9:     for each facet $f$ in $c$ do
10:         if sibhfs($f$ is not set then
11:             $cid \leftarrow$ sibling child id from refTemplate$\rightarrow$$\mathbf{sibhfs}$;
12:             $lid \leftarrow$ sibling half-facet id from refTemplate$\rightarrow$$\mathbf{sibhfs}$;
13:             sibhfs($c$, $f$) = (childEnts[$cid$], $lid$);
14:             sibhfs(childEnts[$cid$], $lid$) = ($c$, $f$);
15:         end if
16:     end for
17: end for
```

Algorithm 2 Updating AHF maps between children of sibling parents sharing a facet.

```
Require: : PE: entities of previous level, CE: entities of current level, PV: vertex
    indices of PE, CV: vertex indices of CE, refTemplate: refinement template for
    entity type and degree entity type and degree
Ensure: sibhfs: update sibling half-facets for CE, v2hf: update vertex to half-facet
    for duplicates of PV in CV
 1: for each vertex $v$ in PV do
 2:     $lvid$ ← local id of v in $pid$;
 3:     $pid$ ← parent entity from incident half-facet $v2hf(v)$;
 4:     $cid$ ← child id from refTemplate→$\mathbf{v2hf}$;
 5:     $lid$ ← half-facet id from refTemplate→$\mathbf{v2hf}$;
 6:     $curvid$ ← current id of v in CV;
 7:     v2hf($curvid$) = (CE[$cid$], $lid$);
 8: end for
 9: for each entity $e$ in PE do
10:     for each facet $f$ in $e$ do
11:         $childs$ ← children entities of $e$ incident on $f$;
12:         $sibeids$ ← sibling entities from $sibhfs(e, f)$;
13:         $siblids$ ← sibling facet ids from $sibhfs(e, f)$;
14:         for each sibling $seid$ in $sibeids$ do
15:             $sibchilds$ ← children entities of $seid$ incident on $siblids[seid]$;
16:             find orientation of $f$ and $siblids[seid]$;
17:             update sibhfs($childs$,$f$) with matching ($sibeids$, $siblids$);
18:         end for
19:     end for
20: end for
```

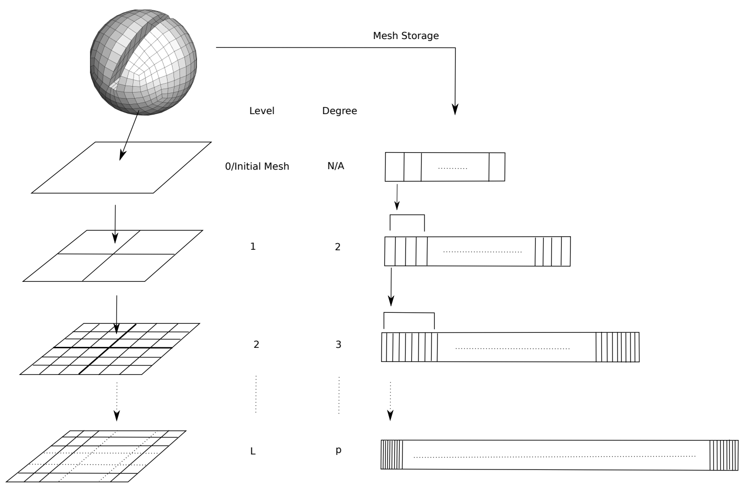


Figure 3.4: The data layout of meshes at each level. For example, starting with the first quad of the sphere mesh, after each level of refinement, the children are stored in a pre-decided order as defined in the refinement templates. This local order uniquely identifies each child of the parent.

requirement for a new level is estimated, the memory is allocated in contiguous blocks. During refinement, as each entity is subdivided, the new entities are stored according to the local order specified in the refinement template. Thus, children of the first entity in previous level are stored first, then the children of the second and so on. This data layout supports straightforward index based inter-level (i.e., parent-to-child or child-to-parent) queries, which are vital to multi-level methods. Figure 3.4 illustrates this memory layout. To provide mesh independence at each level, we duplicate vertices from the previous level to create the new mesh along with new vertices. As a result, the mesh hierarchy generation would return a sequence of meshes that are independent of each other while providing inter- and intra-level mesh access. If necessary, the vertices at a specific level can be renumbered to maintain small bandwidths for the assembled matrices.

Table 3.3: Estimates for new entities that will be created after a refinement of degree p. These estimates are for a 3D mesh with explicit curves and surfaces. Here $|\cdot|$ denotes the number of entities in a particular entity set. E^p and F^p are the total number of edges and faces in the 3D mesh at the previous level, respectively.

	#Entities at previous level	#Entities after refinement of degree p
Vertices	$\|V^p\|$	$\|V\| = \|V^p\| + nv_e * \|E^p\| + nv_f * \|F^p\| + nv_c * \|C\|$
Explicit Edges	$\left\|E^p_{exp}\right\|$	$\|E_{exp}\| = p * \left\|E^p_{exp}\right\|$
Explicit Faces	$\left\|F^p_{exp}\right\|$	$\|F_{exp}\| = p^2 * \left\|F^p_{exp}\right\|$
Cells	$\|C\|$	$\|C\| = p^3 * \|C^p\|$

3.4 Parallel Hierarchical Mesh Generation

Generating mesh hierarchies using uniform refinement in parallel may seem to be a relatively straightforward process, since each processor refines its local mesh. If the initial mesh distribution is balanced, then uniform refinement would not introduce any additional imbalance. Thus, there is no need to move mesh entities between processors. However, generation of each level of the hierarchy also introduces new entities on the shared interface between processors. Unless these new entities are resolved, the generated hierarchies are only useful for local operations. More complicated algorithms requiring shared information such as exchanging ghost-layers, solver setups that rely on knowing owned/ghosted entities for DoF distribution would break down. In this section, we discuss two parallel communication algorithms that resolves such new entities on the shared interface between processors. We also describe the refinement algorithm to generate multi-degree, multi-dimensional hierarchies of unstructured meshes in parallel.

3.4.1 Parallel Communication to Resolve Shared Entities

The MOAB library implemented with array-based data-structures have been designed to be scalable in memory layout and access. There are several optimized one-sided communication algorithms that make use of aggregation strategies to minimize total data transferred between processors. Using this parallel framework, the mesh hierarchy generation can be performed in a series of optimal steps. Once each processor loads a part of the distributed coarse mesh, local refinement for all the entities can be performed. However, the new entities on the shared interface created by the refinement have to resolved, so that other communication algorithms requiring shared mesh or data can take place. We discuss the following two approaches to communicate and resolve these new shared entities:

- The first approach uses a parallel merge algorithm using geometric proximity of the mesh vertices to resolve first the shared vertices and subsequently the subentities.

- The second approach uses a combinatorial matching algorithm using the coarse mesh to match local handles of new entities with their remote handles.

Merge based resolve interface algorithm

Since the refinement is based on pre-defined templates, the co-ordinates for the new vertices and entities on the shared interface between processors will be the same. We utilize this information to design a communication algorithm to resolve the newly created entities in preparation for synchronization of ghost layers and exchange of meta-data (MOAB tags[107]). The first approach to resolve shared new entities as each level is created based on parallel merge algorithm using geometric proximity of the mesh vertices. This parallel merge of the interface mesh algorithm first matches mesh vertices based on geometric proximity and then uses connectivity matching algorithm to decipher the corresponding entity in the local mesh. The merging algorithm derives its motivation from the vertex-matching algorithm described in [58, 110]. The algorithm proceeds by first partitioning the geometric bounding box of all vertices over processors, with each processor responsible for a distinct geometric region (plus a small epsilon layer whose thickness is twice the distance tolerance of the merge). Then, each processor retrieves the vertices on the skin of the local mesh and assembles a tuple list [110] that holds the coordinate positions of the vertices and the destination processor. Finally, the higher-dimensional entities on the skin are resolved using the connectivities. One disadvantage of the above merge algorithm is that it does not use the communication pattern that is already available from the coarsest mesh, leading to use of global communications instead of one-sided communication.

Combinatorial matching based resolve interface algorithm

To overcome the shortcoming of multiple rounds of global communications in the merge-based approach, we design an optimized algorithm for the resolution of the shared entities by taking into account the communication pattern of the coarsest mesh. In MOAB, the shared interface of a distributed mesh is resolved during the mesh loading step, which involves creating the following parallel information (stored using tags) for any entity:

- PSTATUS: This is a flag classifying the type of sharing information for an entity, and there are five types (owned, interface, ghost, shared, and multishared).

- SHARED: In the case an entity is shared with only one other process, the entity is classified as shared, and its remote handle and the remote processor rank is stored.
- MULTISHARED: In the case an entity is shared with more than one other process, the entity is classified as shared, and all of its remote handles in remote processors are stored including the current processor and local handle.

Any algorithm for resolution of entities on the shared interface has to create or update these three kind of information correctly. Now, we note that the local and remote representation of any shared interface entity may differ as shown in Figure 3.5. However, the remote interface entity can be oriented to match the local representation via a combination of its connectivity and subsequently, its local and remote child entities. For example, in 2D (Figure 3.6), the local child edges can be matched with the remote child edges depending on the orientation difference between the coarsest local edge and its remote edge. This observation along with the fact that the communication pattern for each processor does not change during and after hierarchy generation forms the basis of the optimized resolve shared entities algorithm.

The optimized algorithm starts by gathering a list of shared processors with the current processor. For each shared processor, a list of coarsest level entities are collected along with their children entities and their connectivities. This information is then send to the sharing processor while ensuring that the local handles of the coarsest entities are replaced with their remote handles on the receiving processor. On the receiving processor, the connectivity of the coarsest entities now follows the

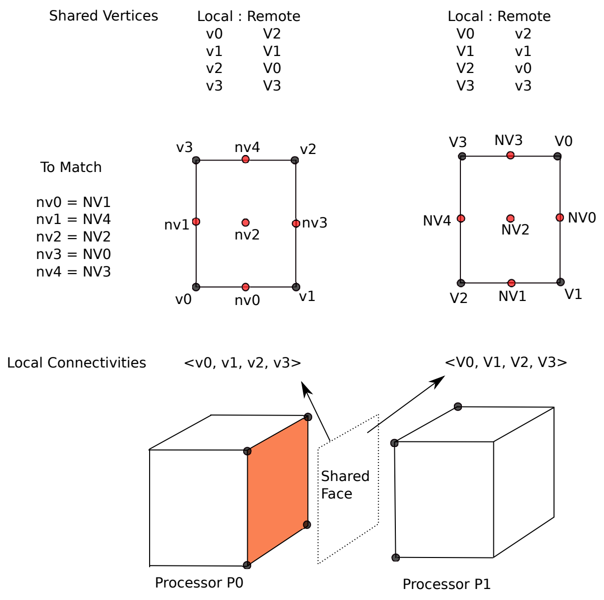


Figure 3.5: A shared face between two processors. After one level of refinement the new vertices and faces have to be matched.

order on the sending processor leading to detection of the combination difference between the local and remote representations. After this single round of one-sided communication, each process contains all the remote handles of new entities from its sharing processors. The algorithm then proceeds to decipher the type and match the local and remote handles of the new entities and finally updating the parallel information as described above. The outline of the optimized algorithm is given in Algorithms 3,4 and 5.

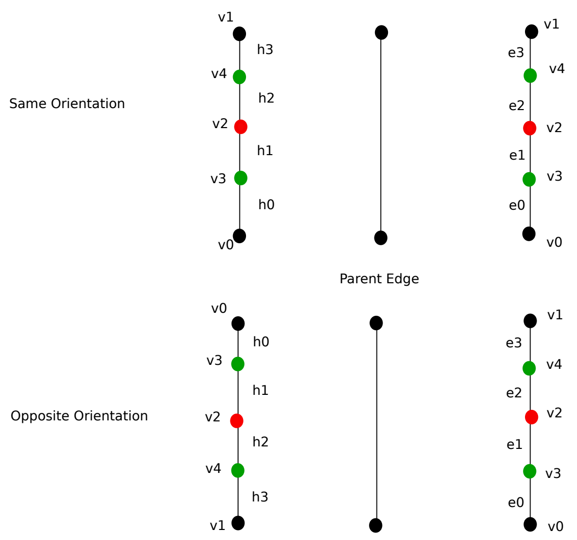


Figure 3.6: For a shared edge, there can be two options: either the remote edge has the same orientation or opposite orientation. Depending on the orientation, the new vertices and child edges can be appropriately matched.

Algorithm 3 The optimized resolve shared algorithm using coarse mesh information.

Require: : meshSets for N levels: Each meshSet contain the mesh for a level in the hierarchy.

Ensure: pstatus: update parallel status flag for entities in meshSets, shared/multishared: update shared or multishared tags for entities in meshSets.

```
 1: Obtain the list of shared processors SP = $P_s[i]$, $i$ = 1,..,$nsharedProcs$ with the
    current processor $P_c$;
 2: Create $localBuffs$ and $remoteBuffs$ for sending to and receiving data from
    SP;
 3: for each processor $P_s[i]$ in SP do
 4:     Obtain $sharedEnts$→list of coarsest shared entities;
 5:     Obtain $EntList$→$collect\_sending\_data$($P_s[i]$, $sharedEnts$);
 6:     Add to $localBuffs[i]$→$EntList$;
 7: end for
 8: Send $localBuffs$ to SP and receive data into $remoteBuffs$;
 9: Obtain $(remote\_procs, remote\_handles)$→
    $match\_remote\_handles(SP, localBuffs, remoteBuffs)$;
10: Call $update\_parallel\_info$($remote\_procs$, $remote\_handles$);
```

Algorithm 4 Algorithm $collect\_sending\_data$ to collect sending data to a shared processor based on the coarsest shared entities.

```
Require: : $to\_proc$: sending processor, $sharedEnts$: list of coarsest entitites
    shared with $to\_proc$
Ensure: $EntList$: list of entities including children at all levels to be sent to
    $to\_proc$
 1: Obtain from $sharedEnts$ the sets $F_0$, $E_0$, $V_0$ where
 2:     $F_0$ = list of shared coarsest faces,
 3:     $E_0$ = list of shared coarsest edges that are not part of the entities in $F_0$,
 4:     $V_0$ = list of shared coarsest vertices that are not part of entities in $F_0$ and $E_0$;
 5: Create $Flist$ = <$F_0$, $F_1$, .., $F_N$, $FE_0$,$FE_1$,..,$FE_N$,$FC_0$,$FC_1$,..,$FC_N$> where
 6:     $F_l$ = list of children entities of $F_0$ at level l,
 7:     $FE_l$ = list of bounding edges of entities at level l,
 8:     $FC_l$ = list of connectivities of entities at level l;
 9: Create $Elist$ = <$E_0$, $E_1$, .., $E_N$,$EC_0$,$EC_1$,..,$EC_N$> where
10:     $E_l$ = list of children entities of $E_0$ at level l,
11:     $EC_l$ = list of connectivities of entities at level l;
12: Create $Vlist$ = <$V_0$, $V_1$, .., $V_N$> where
13:     $V_l$ = list of duplicates of $V_0$ at level l;
14: Add to $EntList$→<$Flist$,$Elist$,$Vlist$>;
```

3.4.2 Refinement Algorithm

Figure 3.7 shows the flowchart of the refinement algorithm to generate a hierarchy of unstructured meshes in parallel. We have also developed a tool to expose this functionality in MOAB that can read in a mesh, generate the hierarchies in parallel for a given number of levels and a sequence of degree of refinements, and then write back to a HDF5 file for consumption of the mesh hierarchy in PDE solvers..

Algorithm 5 Algorithm $match\_remote\_handles$ to decipher remote handles for new entities in the mesh hierarchy.

```
Require: :  SP = $P_s[i]$, $i$ = 1,..,$nsharedProcs$: list of shared processors,
    $localBuffs$ and $remoteBuffs$: local and remote handles to be resolved.
Ensure: $pinfo \rightarrow (remote\_procs, remote\_handles)$
 1: for each processor $P_s[i]$ in SP do
 2:    Obtain <$locFlist$,$locElist$,$locVlist$> from $localBuffs[i]$;
 3:    Obtain <$remFlist$,$remElist$,$remVlist$> from $remoteBuffs[i]$;
 4:    Match children faces, their edges and vertices of shared faces
 5:    for each coarsest face $f_0^L$ in $locFlist \rightarrow F_0$ do
 6:       Find $f_0^R$ in $remFlist \rightarrow F_0$ corresponding to $f_0^L$;
 7:       Find the orientation difference $O_d$ between $f_0^L$ and $f_0^R$;
 8:       for level $l$=1:N do
 9:          Find childrens $CH(f_0^L)$ of $f_0^L$ at current level;
10:          Find parent $P^L$ of $CH(f_0^L)$ at level $l-1$;
11:          Find corresponding parent $P^R$ from $pinfo \rightarrow P^L$;
12:          Find childrens $CH(P^R)$ at current level;
13:          Using $O_d$ match handles of children faces, edges and vertices;
14:          Push to $pinfo \rightarrow (CH(f_0^L)$,$P_s[i]$,$CH(P^R))$;
15:       end for
16:    end for
17:    Match children edges and vertices of shared edges
18:    for each coarsest edge $e_0^L$ in $locElist \rightarrow E_0$ do
19:       Find $e_0^R$ in $remElist \rightarrow E_0$ corresponding to $e_0^L$;
20:       Find the orientation difference $O_d$ between $e_0^L$ and $e_0^R$;
21:       for level $l$=1:N do
22:          Find childrens $CH(e_0^L)$ of $e_0^L$ at current level;
23:          Find childrens $CH(e_0^R)$ of $e_0^R$ at current level;
24:          Using $O_d$ match handles of children edges and vertices;
25:          Push to $pinfo \rightarrow (CH(e_0^L)$,$P_s[i]$,$CH(e_0^R))$;
26:       end for
27:    end for
28:    Match vertices
29:    for each coarsest vertex $v_0^L$ in $locVlist \rightarrow V_0$ do
30:       Find $v_0^R$ in $remVlist \rightarrow V_0$ corresponding to $v_0^L$;
31:       Match duplicates of $v_0^L$ and $v_0^R$ at current level;
32:       Push to $pinfo \rightarrow (dup(v_0^L)$,$P_s[i]$,$dup(v_0^R))$;
33:    end for
34: end for
```

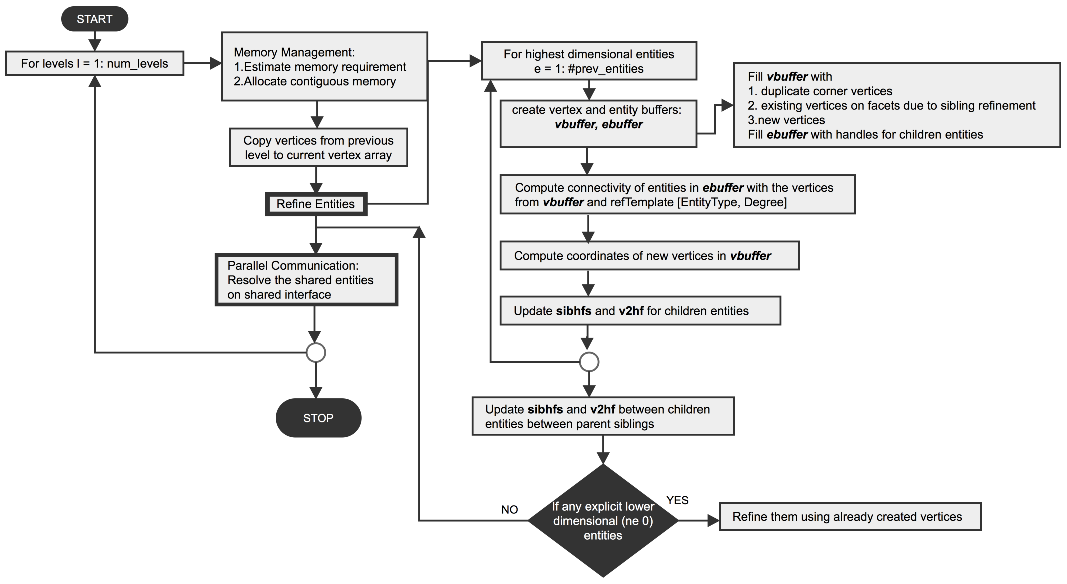

START
For levels l = 1: num_levels
Memory Management:
1.Estimate memory requirement
2.Allocate contiguous memory
Copy vertices from previous level to current vertex array
Refine Entities
Parallel Communication: Resolve the shared entities on shared interface
STOP
For highest dimensional entities e = 1: #prev_entities
create vertex and entity buffers: vbuffer, ebuffer
Fill vbuffer with
1. duplicate corner vertices
2. existing vertices on facets due to sibling refinement
3.new vertices
Fill ebuffer with handles for children entities
Compute connectivity of entities in ebuffer with the vertices from vbuffer and refTemplate [EntityType, Degree]
Compute coordinates of new vertices in vbuffer
Update sibhfs and v2hf for children entities
Update sibhfs and v2hf between children entities between parent siblings
If any explicit lower dimensional (ne 0) entities
NO
YES
Refine them using already created vertices

Figure 3.7: The flowchart for generation of mesh hierarchy with length num_levels.

Chapter 4

Conformal and Non-conformal Adaptive Mesh Refinement

In this chapter, we present a generalization of the *Array-based Half-Facet (AHF)* mesh data structure, called *Hierarchical AHF*, for hierarchical unstructured meshes generated from adaptive mesh refinement for solving PDEs. This data structure extends the *AHF* data structure [40] to support meshes with hierarchical structure, which often are generated from adaptive mesh refinement (AMR). The design goals of our data structure include generality to support efficient neighborhood queries, refinement and derefinement (conformal and non-conformal), and *hp*-FEM with mesh smoothing. Our data structure utilizes the *sibling half-facets* as a core abstraction, coupled with a tree structure for hierarchical information.

4.1 Adaptive Mesh Refinement and Hierarchical AHF

In this data model, we assume that each element has a standard numbering convention for its vertices and its facets. For standard elements, we follow the convention of the CGNS (CFD General Notation System) [91, 111]. We do not require explicit representation of intermediate dimensional entities between 1 and $d-1$. Instead, we treat the half-facets as *implicit entit*ies, and refer to a half-facet using the element ID and its local ID within the element.

In the process of AMR, to avoid the duplication of new vertices introduced by refinement, efficient adjacency queries are critical. The AHF data structure provides efficient query operations with nice memory performance. A hierarchical structure is generally necessary for multi-level methods for the linear system of numerical PDEs. In our data model the hierarchy is stored in an array-based tree-like structure. We refer to this data model as the *Hierarchical AHF*.

4.1.1 Hierarchical Structure

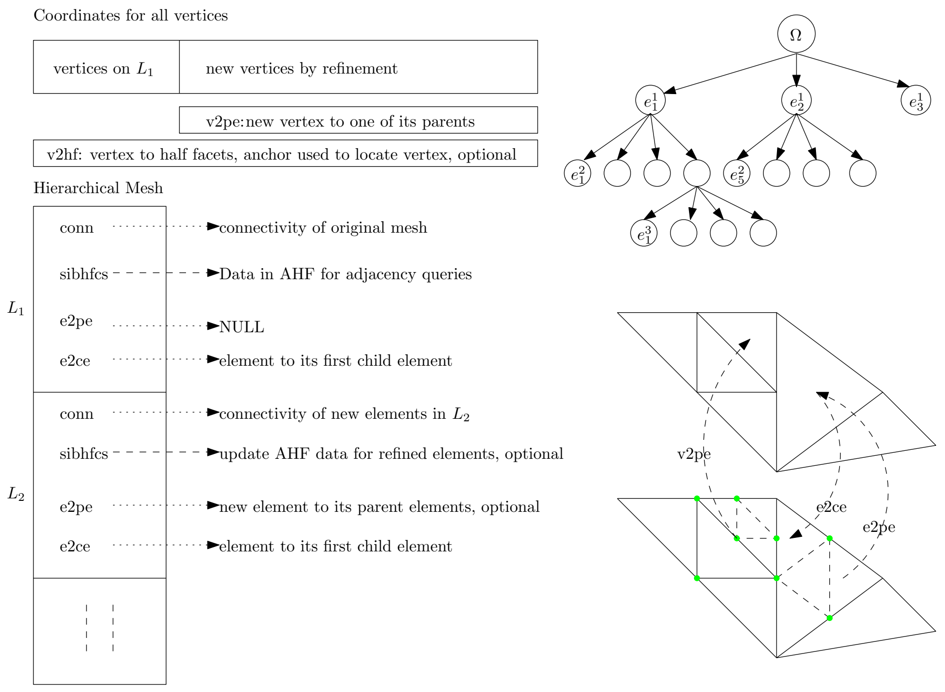
The design of the mesh data structure for adaptive mesh refinement assumes that we start with a conformal manifold mesh. An initial conformal mesh is easy to generate and it is natural to form a hierarchical structure by mesh adaptation. The refinement and derefinement requires efficient adjacency queries, which are provided by AHF.

The initial mesh is adaptively refined and the results will be stored in a hierarchical structure. The resulting elements from a subdivision of element *K* will be referred as the *child elements* of *K*, which in turn is called the *parent*. If element *K* is refined, then it is said to be *inactive*. Elements generated from subsequent refinement of the children of K will be called the *descendants* of K. On the first level, the original mesh is stored. Then some elements of the mesh are marked for refinement. The second level would be the child elements of these elements; see Figure 4.1. On each level, the child elements of the upper level will form a conformal mesh (which might not be manifold). This is analogous to quad-tree. For instance, in Figure 4.1, on level 1, element e_1^1, e_2^1, e_3^1 form the initial conformal mesh. On the second level, the children of e_1^1 and e_2^1 , i.e. $e_1^2, \dots, e_8^2$ will also form a conformal mesh. To traverse the tree, we store *e2ce* for each element, which is the mapping from the elements to the IDs of their child elements on the next level. On each level, *e2ce* is represented as an array. For regular refinement, *e2ce* will only store the ID of the first child element, since all children are stored in consecutive order in an array.

4.1.2 Hierarchical AHF

In the hierarchical mesh data structure, the topological information, i.e., the connectivity table of elements will be stored for each level of the mesh. The original mesh is treated as the first level of the mesh. During the refinement, some elements of the mesh are marked to be refined. The second level would be the child elements; see Figure 4.1. On each level, the connectivity will be stored in $conn$ of the mesh data structure. Each level of the sub-mesh will contain vertices both from the current level and previous levels, thus storing vertices for each level would be a waste of memory. Therefore we store the geometric data, i.e. coordinates of all vertices, in a separate array. The Hierarchical AHF representation is illustrated in Figure 4.1.

To support efficient intra- and inter-level queries, auxiliary information is nec-



Coordinates for all vertices
vertices on L_1
new vertices by refinement
v2pe:new vertex to one of its parents
v2hf: vertex to half facets, anchor used to locate vertex, optional
Hierarchical Mesh
L_1
conn
connectivity of original mesh
sibhfcs
Data in AHF for adjacency queries
e2pe
NULL
e2ce
element to its first child element
L_2
conn
connectivity of new elements in L_2
sibhfcs
update AHF data for refined elements, optional
e2pe
new element to its parent elements, optional
e2ce
element to its first child element
Ω
e_1^1
e_2^1
e_3^1
e_1^2
e_5^2
e_1^3
v2pe
e2ce
e2pe

Figure 4.1: Hierarchical array-based half-facet data structure for a multi-level mesh.

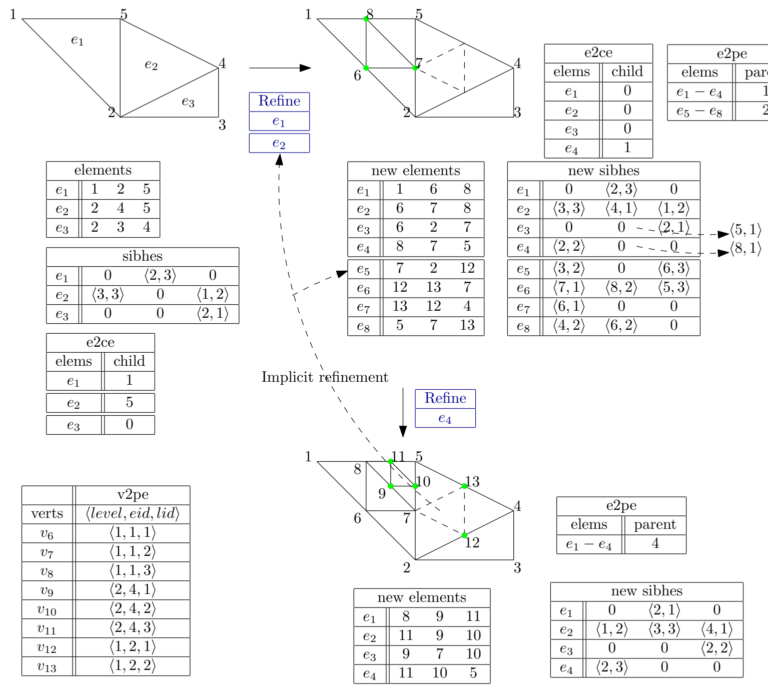
essary. For the intra-level queries, the neighboring information, i.e. AHF data will be stored in an array for each level sub-mesh. Since the sub-mesh is conformal on each level, AHF data can be built in a natural way and the data is represented as *sibhfcs* (sibling half-facets) in the mesh data structure in Figure 4.1. In the process of mesh adaptation, the neighbor information *sibhfcs* will be updated incrementally. For inter-level queries, extra information (like *e2pe* and *e2ce)* is stored in arrays. For each element on a certain level, *e2ce* is the ID of the first child element on the next level. On the next level, other child elements of this element will be stored next to the first child element. *e2ce* is stored in an array for each level and it is necessary for inter-level traversals. *e2pe*, element to parent element, is the ID of the parent element on previous level of this element and it is optional.

To support efficient queries to the parent element for each new vertex, a separate mapping *v2pe* is stored. For each new vertex, *v2pe* is an array of tuples: *level, eid, lid*, where *level* is the level of its parent element, *eid* is the ID of the parent element in *level*, *lid* is the canonical ID of this vertex in its parent ID. If the 1-irregularity rule is applied, the *lid* would be the same as the local ID of the refined edge. *v2pe* is generally necessary for multi-level methods. Also we could use *v2pe* to determine which vertex is a hanging node on which level. Generally, if the 1-irregularity rule is applied, vertex v could only be a hanging node on level *v2pe(v).level+1*. This could be further determined by checking if *v2pe(v).eid*'s sibling elements are refined. If not, then v is a hanging node.

In Figure 4.2, we illustrate the data structure by refining a simple mesh. First, a user specifies refinement of e_1. The new elements will be created and *e2ce* in level 1 will be updated. Then the user specifies e_4 on the second level to be refined. Here the 1-irregularity rule is applied to keep the mesh graded. This will introduce an implicit refinement of e_2 on the first level. Correspondingly, data on level 2 will be updated. *v2pe* will be stored in a separate array.

4.2 Construction and Modification of Hierarchical AHF

In this section, we describe some detailed algorithms for the construction of Hierarchical AHF, as well as some query and modification operations. Since AHF is array-based, these algorithms can be implemented in many programming languages, including MATLAB, C/C++, FORTRAN, etc. We will also describe our implementation in MATLAB.



elements			
e_1	1	2	5
e_2	2	4	5
e_3	2	3	4

sibhes			
e_1	0	$\langle 2,3\rangle$	0
e_2	$\langle 3,3\rangle$	0	$\langle 1,2\rangle$
e_3	0	0	$\langle 2,1\rangle$

e2ce	
elems	child
e_1	1
e_2	5
e_3	0

e2ce	
elems	child
e_1	0
e_2	0
e_3	0
e_4	1

e2pe	
elems	parent
$e_1 - e_4$	1
$e_5 - e_8$	2

new elements			
e_1	1	6	8
e_2	6	7	8
e_3	6	2	7
e_4	8	7	5
e_5	7	2	12
e_6	12	13	7
e_7	13	12	4
e_8	5	7	13

new sibhes			
e_1	0	$\langle 2,3\rangle$	0
e_2	$\langle 3,3\rangle$	$\langle 4,1\rangle$	$\langle 1,2\rangle$
e_3	0	0	$\langle 2,1\rangle$
e_4	$\langle 2,2\rangle$	0	0
e_5	$\langle 3,2\rangle$	0	$\langle 6,3\rangle$
e_6	$\langle 7,1\rangle$	$\langle 8,2\rangle$	$\langle 5,3\rangle$
e_7	$\langle 6,1\rangle$	0	0
e_8	$\langle 4,2\rangle$	$\langle 6,2\rangle$	0

	v2pe
verts	$\langle level, eid, lid\rangle$
v_6	$\langle 1,1,1\rangle$
v_7	$\langle 1,1,2\rangle$
v_8	$\langle 1,1,3\rangle$
v_9	$\langle 2,4,1\rangle$
v_{10}	$\langle 2,4,2\rangle$
v_{11}	$\langle 2,4,3\rangle$
v_{12}	$\langle 1,2,1\rangle$
v_{13}	$\langle 1,2,2\rangle$

e2pe	
elems	parent
$e_1 - e_4$	4

new elements			
e_1	8	9	11
e_2	11	9	10
e_3	9	7	10
e_4	11	10	5

new sibhes			
e_1	0	$\langle 2,1\rangle$	0
e_2	$\langle 1,2\rangle$	$\langle 3,3\rangle$	$\langle 4,1\rangle$
e_3	0	0	$\langle 2,2\rangle$
e_4	$\langle 2,3\rangle$	0	0

Figure 4.2: Example of Hierarchical AHF under refinement.

4.2.1 Construction of Hierarchical AHF

In the half-facet data structure, there are two components: *sibhfcs* (sibling half-facets) and *v2hf* (vertex to half-facet). The former is central to AHF, as nearly all adjacency queries require it. These sibling half-facets should map to each other and form a cycle. The latter array, *v2hf*, is optional for many operations; for Hierarchical AHF, it is not built for new vertices. Instead, *v2pe* (vertex to parent element) is constructed to store information of new vertices.

In a hierarchical mesh, the AHF for the initial mesh will be constructed first and then the sub-mesh of each level is constructed incrementally, taking advantage of ancestor information. In general, the refinement and derefinement require different algorithms. In the following, we describe these two parts in a manner independent of the dimension of the mesh.

Hierarchical AHF: Refinement

Algorithm 6 outlines the steps for mesh refinement, which is applicable to half-facets in arbitrary dimensions, and is particularly efficient in 1 to 3 dimensions. New elements are created and appended in corresponding levels. Meanwhile, the adjacency information, *sibhfcs*, is updated. This step requires the input of elements that are marked for refinement:

refTags: arrays stored in a hierarchical structure, which store the elements to be refined on each level.

The computational cost of Algorithm 6 is linear, assuming that the number of elements incident on an edge is bounded by a small constant c. For the storage requirement, let $|C_r|$ denote the number of elements to be refined in a certain level of the mesh. The amortized memory storage increased by refinement will be approximately $(2^d v_c + 2^d f_c + 2^d)|C_r|$ integers, for the connectivity, the neighbor information, and inter-level maps, with extra space for new vertices coordinates and $v2pe$ map. Here v_c and f_c are the numbers of vertices per cell and the number of faces per cells, 2^d is the number of children per element.

Hierarchical AHF: Derefinement

During the second step, we update the sibling half-facets during derefinement. This step requires the input of elements that are marked for derefinement:

derefTags: a hierarchical structure which stores elements to be derefined on each level. For each element e in derefTags, we assume that e is refined and the derefinement operation will remove all children of e and set e to be active.

Algorithm 6 Update Sibling Half-Facets for Refinement.

Require: hielems: hierarchical mesh data, refTags

Ensure: sibhfcs: cyclic mappings of sibling half-facets for each level of mesh

```
 1: for each $level$ in hielems do
 2:   for each element $e$ in $refTags(level)$ do
 3:     for each $edge$ in element $e$ do
 4:       Loop through elements in $level$ incident to $edge$ to check if $edge$ is refined
 5:       if $edge$ is not refined then
 6:         Refine $edge$ by inserting vertex $v$ in the middle;
 7:         Update $v2pe$ for vertex $v$, $v2pe(v) = \langle level, e, edge \rangle$;
 8:       end if
 9:     end for
10:     Refine element $e$ by predefined strategy and update $e2ce(e)$
11:     Construct $sibhfcs$ for children of element $e$;
        {Update sibhfcs for submesh on level+1:}
12:     for each $facet$ in element $e$ do
13:       Check opposite element of $facet$ on $level$ of submesh
14:       if opposite element is refined then
15:         Update $sibhfcs$ for children of element $e$;
16:         Update $sibhfcs$ for children of opposite element;
17:       else
18:         Update $sibhfcs$ for children of element $e$;
19:       end if
20:     end for
21:   end for
22: end for
```

Algorithm 7 Update Sibling Half-Facets for Derefinement.

```
Require: hielems: hierarchical mesh data, derefTags
Ensure: sibhfcs: cyclic mappings of sibling half-facets for each level of mesh
 1: for each $level$ in hielems do
 2:   for each element $e$ in $derefTags(level)$ do
 3:     for each $edge$ in element $e$ do
 4:       Loop through elements in $level$ incident to $edge$;
 5:       if none of incident elements is refined then
 6:         Derefine $edge$ by removing vertex $v$ which is in the middle;
 7:         Update $v2pe$ for vertex $v$, $v2pe(v) = \langle 0, 0, 0 \rangle$;
 8:       else
 9:         $edge$ cannot be derefined;
10:         vertex $v$ which is in the middle is still active
11:       end if
12:     end for
13:     Derefine element $e$ and set $e2ce(e) = 0$;
14:     Set all its children mute;
        {Update $sibhfcs$ for submesh on $level + 1$:}
15:     for each $facet$ in element $e$ do
16:       Check opposite element of $facet$ on $level$ of submesh
17:       if opposite element is refined then
18:         Update $sibhfcs$ for children of opposite element;
19:       end if
20:       Set $sibhfcs$ for children of element $e$ to zeros;
21:     end for
22:   end for
23: end for
```

Algorithm 7 outlines the procedure for this stage, which is applicable to half-facets of arbitrary dimensions. Particularly, a vertex is active if and only if it has incident cells. A hanging node will be set as inactive if all incident elements are removed during derefinement.

Similar to Algorithm 6, the computational cost of Algorithm 7 is also linear, assuming that the number of elements incident on an edge is bounded by a small constant c. To analyze the storage requirement, let $|C_d|$ denote the number of elements to be coarsened in a certain level of the mesh. The update will introduce approximately $(2^d v_c + 2^d f_c + 2^d)|C_d|$ "holes" in the element connectivity, *sibhfcs* arrays and the map $e2pe$. Dynamic memory management could be utilized to reuse such holes, for instance, by building a queue to record the holes in the corresponding array introduced by deletion and having any new insertion reuse the memory.

4.2.2 Mesh Adaptation

Mesh refinement and derefinement can be implemented relatively easily in AHF. For hierarchical meshes, AHF is particularly attractive because the adaptivity could be performed efficiently and AHF can be modified incrementally. This leads to very modular adaptivity strategies. To avoid excessive memory copying, we expand the array by a small percentage (e.g. by 20%) each reallocation, so that the amortized cost for the local modifications is constant.

The data structure could support a refinement strategy whether the mesh is required to be conformal or not. In our MATLAB implementation, we support both regular refinement and red-green refinement (Figure 4.3). Particularly, we enforce the 1-irregularity rule for the non-conformal refinement. The Kelly error indicator [65] is utilized for estimating accuracy and marking elements. The AHF code [40] is used to generate sibling half facet data for the initial mesh.

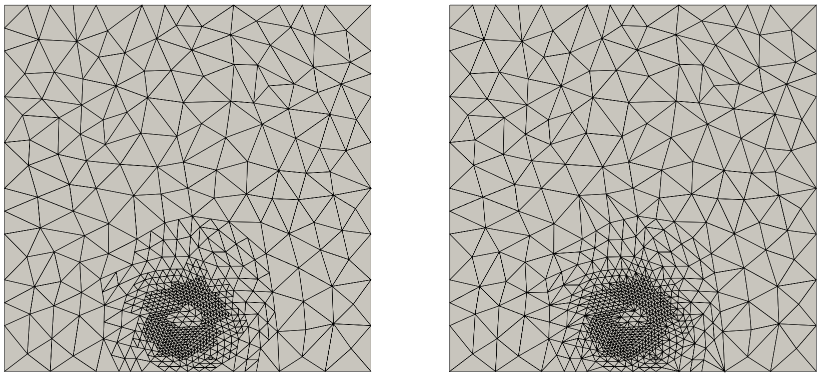


Figure 4.3: Red-Green Refinement

4.3 Adaptive Finite Element Method

4.3.1 Continuity Constraints

For FEM over non-conformal meshes, certain constraints need to be applied for hanging nodes to guarantee continuity. One simple of such constraints is averaging. For instance, in Figure 4.4(a), we assume the degree of freedom(DOF) of node i with coordinates $\boldsymbol{x}_i$ is u_i, then for hanging nodes 4, 5, 6, constraints will be applied as: $u_4 = \frac{1}{2}u_1 + \frac{1}{2}u_2$, $u_5 = \frac{1}{2}u_2 + \frac{1}{2}u_3$, $u_6 = \frac{1}{2}u_1 + \frac{1}{2}u_3$. Then after the stiffness matrix is assembled, the DOFs for hanging nodes like 4, 5, 6 will be eliminated according to the constraints. If the original system is $AU = b$, A is the stiffness matrix, U is the vector of DOFs, and b is the load vector. Then the constraints could be represented as

$$U = \Pi^T \times U_R \tag{4.3.1}$$

Here U_R is the vector of DOFs for regular nodes, then each u_i is a linear combination of U_R, no matter node i is a regular node or hanging nodes. Π is the constraints relation matrix: if π_i is the i-th column of Π, then we have $u_i = \pi_i^T * U_R$, i.e. π_i is

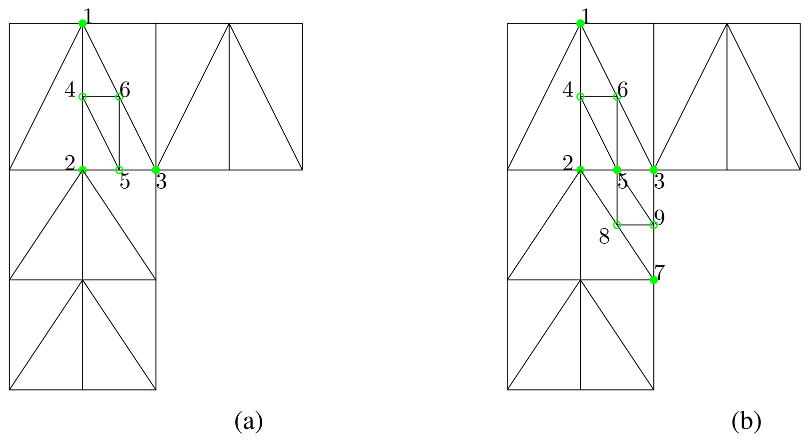


Figure 4.4: hanging nodes

the "global" constrained vector. More specifically, if node i is a regular node, then π_i is a unit vector with 1 at position i and zeros otherwise; if node i is a hanging node, π_i corresponds to the continuity constraints. For example, in h-refinement with one irregular rule, if node i is a hanging node, then it should be the averaging of two regular nodes, say vertices l and r, then

$$\pi_i = [0 \ \cdots \ 0 \ \ \underset{l\text{th}}{1/2} \ \ 0 \ \cdots \ \underset{i\text{th}}{0} \ \cdots \ 0 \ \ \underset{r\text{th}}{1/2} \ \ 0 \cdots \ 0]^T .$$

From this, we could get a reduced system, still symmetric

$$\Pi A \Pi^T U_R = \Pi b. \tag{4.3.2}$$

4.3.2 Hanging Nodes in FEM

When the mesh is refined, though hanging nodes are allowed, some of them will turn into regular nodes in the process, like node 5 in Figure 4.4(b). Regular nodes have their own DOFs and therefore they could actually improve the local resolution by introducing extra DOFs over the original mesh. This means the mesh is getting finer by regular nodes. But what actually do hanging nodes introduce?

For h-refinement, the answer is that if all the nodes introduced by refining the same element stay as hanging nodes, then these irregular nodes simply introduce nothing! Lets discuss this with the example in Figure 4.4(a).

Assume the nodal basis functions on a triangle Δijk with nodes i, j, k are $T_{i,jk}$, $T_{j,ki}$, $T_{k,ij}$ such that $T_{i,jk}$ is 1 at node i and 0 at nodes j, k, similarly for the rest two. Then over the coarse mesh without hanging nodes 4, 5, 6, the element solution for $\Delta 123$ is

$$u_1 T_{1,23} + u_2 T_{2,13} + u_3 T_{3,12} \tag{4.3.3}$$

If the mesh is refined, introducing hanging nodes 4, 5, 6, then the element solution for $\Delta 123$ should be

$$\begin{array}{ccc} \tilde{u}_1 T_{1,46} + & \tilde{u}_2 T_{2,45} + & \tilde{u}_3 T_{3,56} + \\ \tilde{u}_4 N_4 + & \tilde{u}_5 N_5 + & \tilde{u}_6 N_6 \end{array}$$

Here $\tilde{u}_i$ is the DOF of node i under refined mesh,

$$N_4 = \begin{cases} T_{4,25} & \text{on } \Delta 425 \\ T_{4,56} & \text{on } \Delta 456 \\ T_{4,61} & \text{on } \Delta 461 \\ 0 & \text{else} \end{cases} \tag{4.3.4}$$

Similarly for N_5, N_6. Apply the constraints, we get

$$\begin{array}{lll}\tilde{u}_1 T_{1,46}+ & \frac{1}{2}\tilde{u}_1 N_4+ & \dfrac{1}{2}\tilde{u}_1 N_6 + \\ \tilde{u}_2 T_{2,45}+ & \frac{1}{2}\tilde{u}_2 N_4+ & \dfrac{1}{2}\tilde{u}_2 N_5 + \\ \tilde{u}_3 T_{3,56}+ & \frac{1}{2}\tilde{u}_3 N_5+ & \dfrac{1}{2}\tilde{u}_3 N_6\end{array}$$

Then we could assemble a new vertex basis function for node 1 as

$$\tilde{T}_{1,23} = T_{1,46} + \frac{1}{2}N_4 + \frac{1}{2}N_6 \tag{4.3.5}$$

$T_{1,46}$, N_4, N_6 are hat functions so $\tilde{T}_{1,23} = 1$ on node 1, $\tilde{T}_{1,23} = 1/2$ on nodes 4, 6, $\tilde{T}_{1,23} = 0$ on nodes 2, 3, 5, the same as $T_{1,23}$. Over $\Delta 146$,

$$\tilde{T}_{1,23} = T_{1,46} + \frac{1}{2}T_{4,61} + \frac{1}{2}T_{6,41} \tag{4.3.6}$$

This is a linear function on $\Delta 146$ which means it could be determined by its values on nodes 1, 4, 6. From this we could get

$$T_{1,23} = \tilde{T}_{1,23} \text{ on } \Delta 146 \tag{4.3.7}$$

Similarly apply the same argument, we could get

$$T_{1,23} = \tilde{T}_{1,23} \text{ on } \Delta 123 \tag{4.3.8}$$

and further over $\Delta 123$

$$T_{2,31} = \tilde{T}_{2,31} \tag{4.3.9}$$

$$T_{3,12} = \tilde{T}_{3,12} \tag{4.3.10}$$

When we apply the constraints on the linear system, the DOFs of hanging nodes are distributed to the parent nodes. This in fact corresponds to use the assembled nodal basis function(see Appendix A). In this simple case, hanging nodes introduce nothing since the assembled nodal basis functions are the same as the ones in the original mesh.

Let's take another example in Figure 4.4(b). On the refined mesh, nodes 4, 6 are hanging nodes and the element solution for $\Delta 123$ should be

$$\begin{array}{ccc} \tilde{u}_1 T_{1,46}+ & \tilde{u}_2 T_{2,45}+ & \tilde{u}_3 T_{3,56}+ \\ \tilde{u}_4 N_4+ & \tilde{u}_5 N_5+ & \tilde{u}_6 N_6 \end{array}$$

Apply the constraints for nodes 4, 6, we get the element solution

$$\begin{array}{lll} \tilde{u}_1 T_{1,46}+ & \frac{1}{2}\tilde{u}_1 N_4+ & \dfrac{1}{2}\tilde{u}_1 N_6 + \\ \tilde{u}_2 T_{2,45}+ & \frac{1}{2}\tilde{u}_2 N_4 + & \\ \tilde{u}_3 T_{3,56}+ & \frac{1}{2}\tilde{u}_3 N_6 + & \\ \tilde{u}_5 N_5 & & \end{array}$$

We could tell that the nodal basis for node 1 introduced by this element solution is the same as 4.3.6. But for nodes 2, 3, the situation is different. For example

$$\tilde{T}_{2,31} = T_{2,45} + \frac{1}{2}N_4 \tag{4.3.11}$$

Thus

$$\tilde{T}_{2,31} = \begin{cases} T_{2,45} + \frac{1}{2}T_{4,25} & \text{on } \Delta 254 \\ \frac{1}{2}T_{4,56} & \text{on } \Delta 456 \\ 0 & \text{on } \Delta 536 \\ \frac{1}{2}T_{1,46} & \text{on } \Delta 461 \end{cases} \tag{4.3.12}$$

which means that $\tilde{T}_{2,31} \neq T_{2,31}$ on $\Delta 456$, $\Delta 536$, $\Delta 461$. Similarly for node 3, $\tilde{T}_{3,12} \neq T_{3,12}$. In this case, node 5 will introduce something new. On the other hand, when eliminating the DOFs of hanging nodes 4, 6, the stiffness matrix will be altered.

Therefore, only the introduced regular nodes by refinement would affect the linear system! The continuity constraints reflects only the information of solution on the refined element, while neglecting information from its neighbors. Thus, hanging nodes contribute little to the accuracy.

Remark 1. For quad tree, the center node after refinement will serve as regular node, so the refined element will alway introduce new DOF.

Remark 2. In 3-D, the refined tetrahedron, like refined triangle, does not always introduce new DOFs. For Octree, similar as quad tree, new DOF will be introduced when hexahedron is refined.

4.4 Rules for Refinement

4.4.1 Introduction

Definition 3. An element is called a **Phantom Element** if it is illusional and has no real effect, if its introduction does not change the linear system and in turn the accuracy of the finite element approximation.

Clearly, phantom elements should be eliminated, since they waste computational resources without introducing anything useful. In Figure 4.5(b), the original mesh is refined with ∞-irregularity rule and results in phantom elements whose nodes are marked as hollow. According to our analysis in Section 4.3.2, these hanging nodes will all be eliminated and the resulted system is the same as the original mesh in Figure 4.5(a).

Theorem 4. *An element is phantom if and only if all its nodes are hanging nodes.*

Proof. If an element is not phantom, its regular nodes would appear in the linear system would change. On the other hand, If an element is phantom, then all its hanging nodes will be removed from the linear system. In addition, for the nodes of a phantom element, the constraints ensure that removing them would produce the exactly the same shape functions and test functions as if the phantom elements were not introduced. This completes the proof. □

What's the best k for k-irregularity rule? To answer this question, note that any k-irregular mesh can be converted into a 1-irregular mesh. During the conversion, we convert some existing hanging nodes into regular nodes and introduce some new hanging nodes. These new hanging nodes would not appear in the resulting linear system.

The question is how the error would behave for the hanging nodes into regular nodes? We can think of it in the following three aspects:

1) Condition numbers: We can reduce any k-irregular meshes into a 1-irregular mesh. In this step, h would not be reduced if the original mesh is well shaped. The dominating part of the condition number is $O(1/h^2)$. Therefore, 1-irregularity should have similar condition number as 2-irregularity when there are no phantom elements.

2) Accuracy: Since we converted some hanging nodes into regular nodes, the local errors at those points may be reduced. For functions with a smooth distribution of errors, this is desirable.

3) Efficiency: Since we have regular nodes, the number of DOFs increases during the conversion by a small fraction. For a direct solver, this can increase the cost of solving the linear system. However, for multigrid solvers, this increase is negligible. On the other hand, having k-irregularity would make it more expensive to perform local searches in the data structure. The local searches involve indirect memory access and has no memory locality, so in practice its adverse impact on performance will likely be much more than the increased cost of matrix-vector multiplication, which has good locality.

Overall, 1-irregularity is preferred since it produces smoother distributions of errors, and also have comparable condition numbers and efficiency as k-irregularity, especially when multigrid methods are used. When the original mesh is bad, then probably the important regions fall into isolated elements. Moreover, if we allow arbitrary levels of refinement, then the refinement would result of phantom elements which contribute little to improve accuracy.

4.4.2 Refine Strategy

In order to increase accuracy via introducing new DOFs, not just phantom elements, we should define the error measures along facets (edges in 2-D and facets in 3-D). If the error is relatively large along a facet, we would ensure the new vertices on the facets are regular nodes. In triangular mesh, when one triangle is refined according to error indicator, not necessarily all new nodes serving as regular nodes, thus local resolution improves little. This contradicts to our expectation. Instead of refining elements, if we refine facets according to error indicator, then new nodes would be regular and they have their own DOFs. After refining facets then we refine elements sharing these facets. At last we could regularize by k-irregularity rule, k-neighbor rule. Actually, for each element the Kelly error indictor computes the gradient change along each facet and simply sums up these errors to serve as the error of the element. Therefore Kelly error works fine for this refine strategy.

4.4.3 Condition Number of Π in h-refinement

From the formulation, we could tell that Π should have the form

$$\Pi_{n\times m} = \left[\begin{array}{cc} I_{n\times n} & C_{n\times s} \end{array} \right] = \left[\begin{array}{ccccc:ccc} 1 & 0 & \cdots & 0 & 0 & 1/2 & 1/2 & \cdots \\ 0 & 1 & \cdots & 0 & 0 & 1/2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & 0 & 1/2 & \cdots \end{array} \right] \tag{4.4.1}$$

n is the number of regular nodes and $s = m - n$ is the number of hanging nodes. Here we assume that the first n nodes are regular nodes, and the rest are hanging nodes. Here we assume that the first n nodes are regular nodes, and the rest are hanging nodes. If not, suppose original constraint matrix is Π_o and we always could find a unitary matrix E such that $\Pi_o \times E = \Pi$ which has above form. Then the linear system could reduced to $\Pi(E^{-1}AE^{-T})\Pi^T U_R = \Pi(E^{-1}b)$.

To get the condition number of Π, we compute the eigenvalues of $\Pi\Pi^T$ and get

$$\Pi\Pi^T = I + CC^T \tag{4.4.2}$$

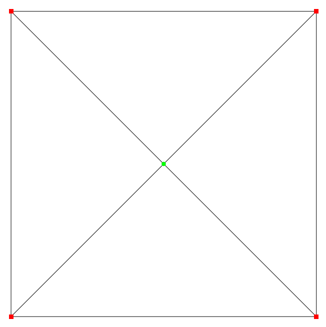
Suppose $CC^T = U\Sigma U^T$, then $\Pi\Pi^T = U(I+\Sigma)U^T$. This means that the eigenvalues $\lambda_1(\Pi\Pi^T) \geq \lambda_2(\Pi\Pi^T) \geq \cdots \geq \lambda_n(\Pi\Pi^T)$ of $\Pi\Pi^T$ satisfies

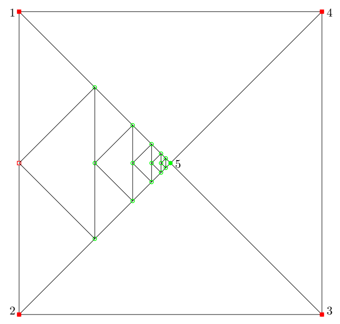
$$\lambda_i(\Pi\Pi^T) = 1 + \lambda_i(CC^T), \;\; i = 1, \cdots n. \tag{4.4.3}$$

Thus if we have the estimation of eigenvalues of CC^T , it trivial for $\Pi\Pi^T$.

For a hanging node j, we could find its parent nodes. If any one of its parent nodes is not regular, than we can find parent nodes of node j's parent nodes. Keep tracing back and these nodes appearing are called ancestor nodes of node j. Continue until the ancestors are regular. Those regular ancestor nodes of hanging node j are called **root nodes** of j and node j is called **descendant node** of these root nodes. For instance, in Figure 4.4(a), node 1 and 2 are the parent nodes and root nodes of node 4. In Figure 4.5, the nodes marked circle inside $\Delta 125$ are descendants of nodes 1, 2, 5.

Note that non-zeros of$j$-th column in $C = \{c_{ij}\}_{n\times s}$ correspond the weights of


(a) Original mesh



(b) Infinity-irregularity rule

Figure 4.5: Irregularity rules

hanging node j against their ancestor nodes, i.e. $\boldsymbol{x}_{n+j} = \sum_{i=1}^{n} c_{ij}\boldsymbol{x}_i$, $j = 1, \cdots, s$. Since hanging nodes are generated by inserting nodes at the middle of an edge, $c_{ij} \geq 0$ and $\sum_{i=1}^{b} c_{ij} = 1$. Non-zeros of i-th row of C correspond to descendant nodes of regular node i, and the values are descendant nodes's weights over node i. Suppose $M = CC^T$, then the diagonal entry of M is $\sum_{j=1}^{s} c_{ij}^2, i = 1, \cdots n$ which is the squared sum of contribution from descendants of node i in terms of weights. The off-diagonal (i, k) entry of M is $\sum_{j=1}^{s} c_{ij}c_{kj}$. We could tell that $\sum_{j=1}^{s} c_{ij}c_{kj} \neq 0$ if and only if there is at least one hanging node whose root nodes include both node i and k.

We define regular node i and k are **directly connected** if $\sum_{j=1}^{s} c_{ij}c_{kj} \neq 0$, which implies that nodes i and k share one descendent node. A path is an ordered sequence of nodes $\{i_1, i_2, \cdots, i_l\}$ such that each node is directly connected to next node. Node i and k are **connected** if there is a path between i and k. Then by this definition a graph is formed over the mesh, such that regular nodes without descendants are isolated nodes in graph, nodes i and k are connected by an edge if they are directly connected. One **connected component** of this graph is a group of nodes such that any two of such nodes are connected, and any other node is not connected to any node in this group. A regular node without descendant is isolated and a trivial connected component. For example in Figure 4.4(b), nodes 1, 2 are directly connected, and one connected component is {1, 2, 7, 3}. In Figure 4.5, nodes 1, 2, 5 form a connected component, nodes 3, 4 are isolated.

Suppose the nontrivial connected components are $G_1, \cdots, G_{nc}$, with n_i regular nodes in each and if reorder the regular nodes as $G_1, \cdots, G_{nc}$, and then regular nodes without descendant, then CC^T will be transformed to a block diagonal matrix

$$\begin{bmatrix} M_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & M_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & M_{nc} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}_{n \times n} \tag{4.4.4}$$

Where $M_1, \cdots, M_{nc}$ correspond to nodes in groups $G_1, \cdots, G_{nc}$. This is due to the fact that nodes sharing the same descendant node must lie in the same group and $\sum_{j=1}^{s} c_{ij} c_{kj} \neq 0$ if and only if there is at least one hanging node whose root nodes include both node i and k.

For each $M_i, i = 1, \cdots, nc$ we estimate its eigenvalues. Since $M = CC^T$, so there exists block $C_i \in \mathbb{R}_+^{n_i \times s_i}$ of C such that $M_i = C_i C_i^T$. For C_i, similarly as C, the rows and columns have the same meaning as the ones of C. Thus

$$\begin{aligned}(1,\cdots,1)_{1\times n_i}C_iC_i^T &= (1,\cdots,1)_{1\times s_i}C_i^T \\ &= (\omega_1^i,\cdots,\omega_{n_i}^i)\end{aligned}$$

Since non-zeros of k-th row of C correspond to descendant nodes of regular node k, and the values are descendant nodes's weights over node k, thus similarly for C_i, ω_l^i is the sum of weights from descendent nodes of l-th regular node in group G_i, $l = 1,\cdots,n_i$, $\omega_l^i \geq 0$. Magnitude of $\{\omega_l^i\}$ depend on if the hanging nodes is well distributed: if hanging nodes are clustered, then $\{\omega_l^i\}$ will be large. Especially when 1-irregularity rule is applied, ω^i will be small. In Figure 4.4(b), for connected component {1, 2, 7, 3}, the weights contributed by hanging nodes are $(1,1,1,1)$. In Figure 4.5, connected component {1, 2, 5} has contribution from clustered hanging nodes $(1.9375, 1.9375, 11.125)$.

Suppose ω^i is the maximum, construct $\hat{M}_i$ as

$$\hat{M}_i = M_i + diag(\omega^i-\omega_1^i,\cdots,\omega^i-\omega_{n_i}^i) \tag{4.4.5}$$

Then

$$(1,\cdots,1)_{1\times n_i}\hat{M}_i = (\omega^i,\cdots,\omega^i)_{1\times n_i} \tag{4.4.6}$$

This means that $\hat{M}_i/\omega^i \in \mathbb{R}_+^{n_i\times s_i}$ is a symmetric column stochastic matrix which has the property that maximum eigenvalue is 1.

Therefore,

$$M_i = \hat{M}_i - diag(\omega^i-\omega_1^i,\cdots,\omega^i-\omega_{n_i}^i) \tag{4.4.7}$$

$\hat{M}_i$ is symmetric thus normal, according to Exercise 26.3(b) in [113], we get for the maximum eigenvalue $\lambda_1(M_i)$ of M_i, there exist an eigenvalue $\lambda(\hat{M}_i)$ of $\hat{M}_i$ such that

$$\begin{aligned} |\lambda_1(M_i) - \lambda(\hat{M}_i)| &\leq ||diag(\omega^i - \omega^i_1, \cdots, \omega^i - \omega^i_{n_i})||_2 \\ &= \max_{l=1}^{n_i}(|\omega^i - \omega^i_l|) \end{aligned}$$

which gives us

$$\lambda_1(M_i) \leq \omega^i + \max_{l=1}^{n_i}(|\omega^i - \omega^i_l|) \tag{4.4.8}$$

Notice that if $\{\omega^i_l\}_{l=1}^{n_i}$ are the same then $\lambda_1(M_i) = \omega^i$. If hanging nodes are well distributed and 1-irregularity rule is applied, $\lambda_1(M_i)$ will be small. This is the case in Figure 4.4(b), for component {1, 2, 7, 3}, $\lambda_1(M) = 1$. In Figure 4.5, $\lambda_1(M) = 9.4563$ for component {1, 2, 5}.

For the maximum eigenvalue $\lambda_1(M)$ of $M = CC^T$, it's easy to get

$$\begin{aligned} \lambda_1(M) &\leq \max_{i=1}^{nc}(\omega^i + \max_{l=1}^{n_i}(|\omega^i - \omega^i_l|)) \\ &\leq 2\max_{i=1}^{nc}\omega^i \end{aligned}$$

Since M is symmetric semidefinite, the minimum $\lambda_n(M) \geq 0$. For adaptive mesh refinement, typically $\lambda_n(M) = 0$ due to the fact that most of the regular nodes result in having no descendent nodes. This implies that

$$\begin{aligned}\operatorname{cond}(\Pi) &= \sqrt{\frac{\lambda_1(\Pi\Pi^T)}{\lambda_n(\Pi\Pi^T)}} \\ &\leq \sqrt{1+\lambda_1(M)} \\ &\leq \sqrt{1+2\max_{i=1}^{nc}\omega^i}\end{aligned}$$

For 1-irregularity rule, since $\omega^i \leq 1$, $\operatorname{cond}(\Pi) \leq \sqrt{1+\max_{k=1}^{n} H_{kk}}$, here H_{kk} is the number of edges which are incident to node k and have hanging nodes on them. In this case, it seems $\operatorname{cond}(\Pi)$ is related to the smallest angle θ_{min} in the sense that $H_{kk} \leq \frac{2\pi}{\theta_{min}}, \forall k$.

In conclusion, the condition number of Π is determined by distribution of hanging nodes and arbitrary level of hanging node could possibly enlarge the condition number of Π unnecessarily.

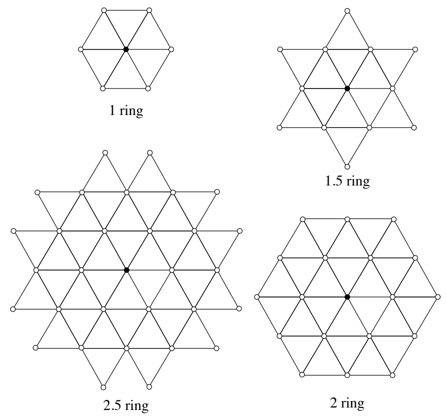
Chapter 5

High-order Reconstruction

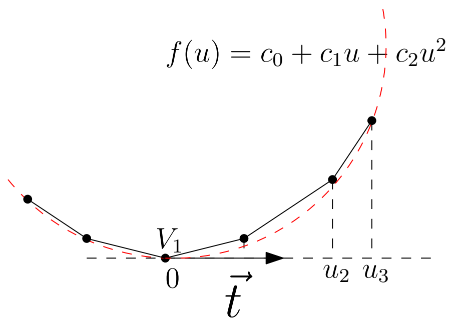
In this chapter, we first introduce our implementation of WALF [61] in MOAB under the Discrete Geometry Module. This could be further utilized by PDE solvers to handle curved boundaries. Then we present a novel method, FAH-WALF: Feature-Aware Hermite-style High-order Surface Reconstruction, for robust reconstructions of unstructured surface meshes to provably high-order accuracy, including the reconstruction of sharp features in the geometry. Our method utilizes a Hermite-style least-squares approximations to achieve high-order accuracy. FAH-WALF significantly extends the scope of WALF by supporting coarser input meshes and also guaranteeing G^0 continuity near sharp features.

5.1 Parallel WALF for Curved Boundary Reconstruction

A key aspect of the refinement algorithm is the positioning of the new vertices as entities are refined. Using a linear point projection scheme for the new vertices



(a) neighbors chosen for fitting on triangular mesh



(b) A curve fitting

Figure 5.1: vertex-based polynomial fittings

would compromise the accuracy of the geometry and in turn that of the finite element solver. To address this issue, we incorporate a polynomial based high-order boundary reconstruction strategy, *Weighted Averaging of Local Fittings (WALF)*, in [61]. WALF constructs third and even higher order accuracy, while guaranteeing C^0 continuity. There are two essential parts for WALF: vertex based local polynomial fittings and high-order continuous surface reconstruction via weighted averaging.

5.1.1 WALF: Weighted Averaging of Local Fittings

Vertex-based polynomial fittings

The first step of WALF is constructing high-order vertex based fittings. Given one vertex, a local coordinates system is formed and the surface/curve will be represented as a height function in the local system. Carefully choose the neighbor vertices and a least square polynomial fitting problem will be formed, see Figure 5.1:

1. Local parameterization around vertex *V*: local *uv*-plane is chosen as the estimated tangent plane, and the surface could be represented as a local height function $[u, v, f(u, v)]$. The *uv*-plane is chosen to be perpendicular to the estimated normal at vertex V.

2. Approximate $f(u, v)$ around a neighborhood of *V*, using polynomial fitting, up to degree *d*:
$$\varphi(\boldsymbol{u}) = \underbrace{\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!}}_{\text{Taylor polynomial}} + \underbrace{\sum_{j,k\geq 0}^{j+k=d+1} \frac{\partial^{j+k}}{\partial u^j \partial v^k} \varphi(\tilde{u}, \tilde{v}) \frac{\tilde{u}^j \tilde{v}^k}{j!k!}}_{\text{remainder}},$$

3. The formed system is a rectangular linear system. Weighted least square approximation is applied to obtain solution.

Continuous, high-order surface reconstruction

Consider a triangle composed of vertices $\boldsymbol{x}_i, i = 1, 2, 3$ and a point $\boldsymbol{x}$ in the triangle. Suppose the barycentric coordinates for $\boldsymbol{x}$ is $\boldsymbol{x} = \sum_{i=1}^{3} \xi_i \boldsymbol{x}_i$ and we have obtain the three local fittings $\boldsymbol{q_i}(\boldsymbol{u_i})$ around $\{\boldsymbol{x}_i\}_{i=1,2,3}$. For each local fitting, we could obtain an estimation of exact geometry of $\boldsymbol{x}$, $\boldsymbol{q_i}$, by projecting it onto local frame $u_i v_i w_i$ and apply local fitting $\boldsymbol{q_i}(\boldsymbol{u_i})$. The we defined the weighted averaging of local fitting for $\boldsymbol{x}$ is
$$\boldsymbol{q}(\boldsymbol{x}) = \sum_{i=1}^{3} \xi_i \boldsymbol{q}_i \tag{5.1.1}$$
WALF constructs a G^0 continuous surface [61], as illustrated in Figure 5.2.

Remark 5. Note that high-order surface reconstruction could create invalid elements. When such cases are detected, we could either simply cancel the high order

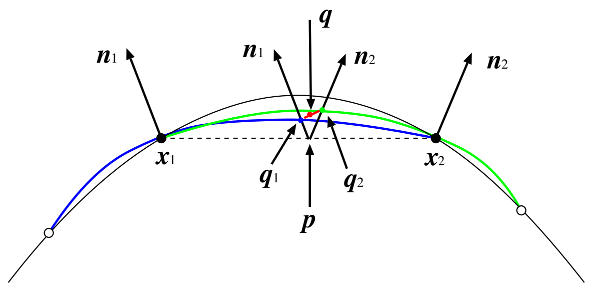


Figure 5.2: 2-D illustration of weighted averaging of local fitting

projection or apply strategies such as the one in [77] to improve geometric approximation.

5.1.2 High-order Surface Reconstruction in MOAB

The reconstruction algorithm has been implemented as a submodule, *Discrete Geometry,* under the framework of MOAB. It supports high-order surface reconstruction in serial and parallel. It could take a mesh set as input and all reconstruction will be performed on this mesh. This mesh set could be a mesh in MOAB (Figure 5.3(a)), or only subset of it (Figure 5.3(b)), or a mesh subset with proper ghost layers in parallel environment (Figure 5.3(c)).

Local polynomial fitting

Given an degree d for vertex V, proper stencil around V will be selected and a polynomial fitting is applied for the height function which is zero at V_1. The fitting could be interpolation or lease square fitting. For a given degree d for fitting, the required rings of neighborhood for each vertex in triangular mesh is $(d+1)/2$ for interpolation and $(d+2)/2$ for least square fitting. During the fitting process, cer-

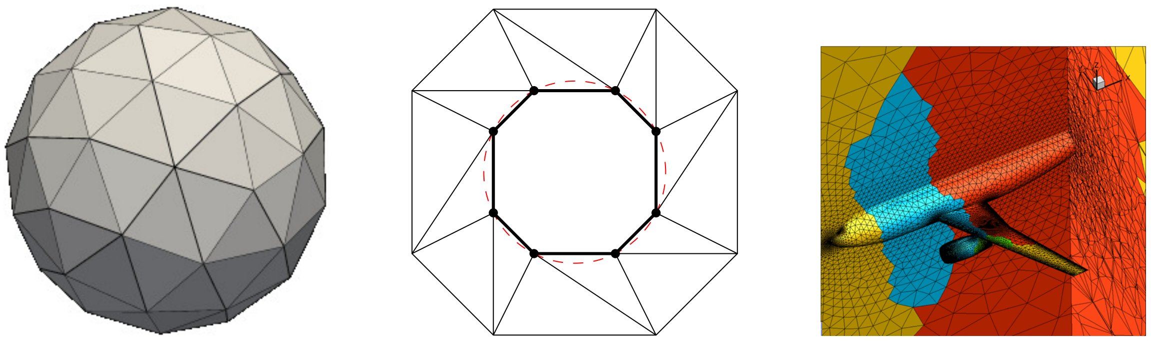

(a) Sphere mesh (b) Inner circle for reconstruction (c) Mesh in parallel

Figure 5.3: Mesh set for reconstruction

tain neighbor points probably will be removed if they might make the local height function folded in *uv*-plane.If the stencil is too small after removing, the degree will be downgraded. If the degree *d* is too large for current stencil, the resulted linear system may be ill-conditioning and the fitting could be over-fitting. Certain strategy is adopted to downgrade degree *d*, thus the output of reconstruction might have lower degree than user input *d*.

Preprocessing under parallel environment

The local polynomial fitting requires *n*-ring neighbor vertices of the given vertex *V*. In parallel environment, mesh is partitioned into different processors. In each processor, for vertices on partition boundary, ghost layers should be retrieved before performing the local fittings. In parallel environment, if the required *n*-ring stencils for local fittings of each vertex could be obtained, including nodes on partition boundary, then reconstruction on each processor gives the same result as in serial environment, even for the shared vertices. In such case therefore no communication is required during or after surface reconstruction. To guarantee this, the number of layers of ghost elements is determined according to the degree of fittings. If the

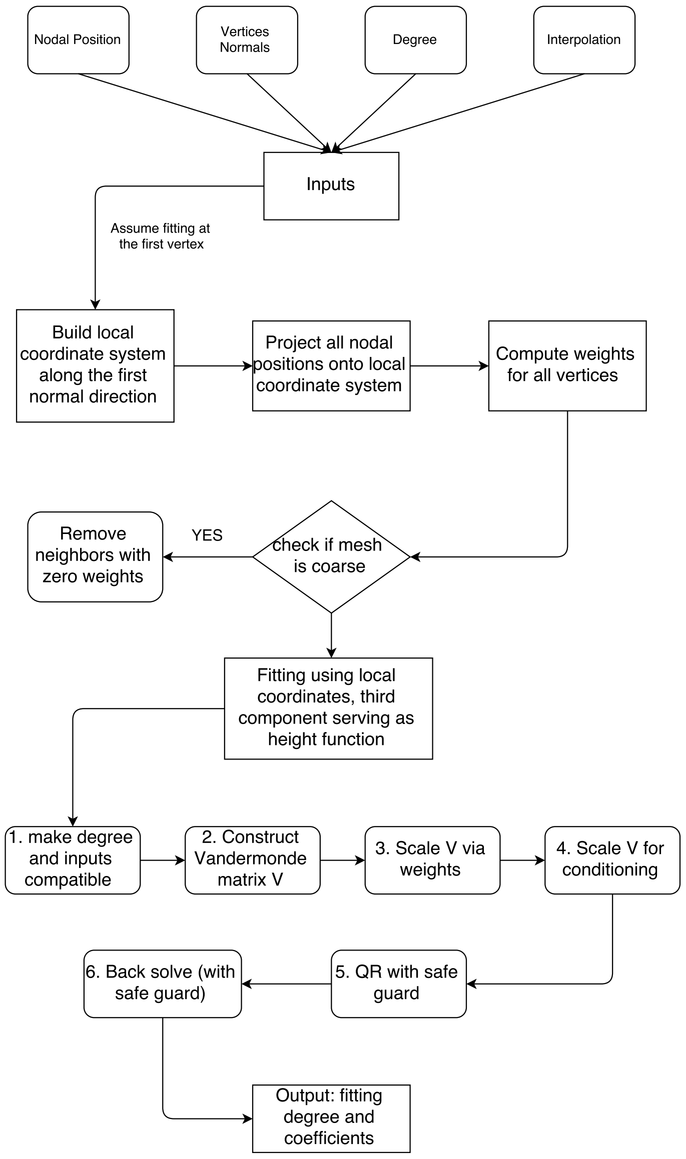

Nodal Position
Vertices Normals
Degree
Interpolation
Inputs
Assume fitting at the first vertex
Build local coordinate system along the first normal direction
Project all nodal positions onto local coordinate system
Compute weights for all vertices
check if mesh is coarse
YES
Remove neighbors with zero weights
Fitting using local coordinates, third component serving as height function
1. make degree and inputs compatible
2. Construct Vandermonde matrix V
3. Scale V via weights
4. Scale V for conditioning
5. QR with safe guard
6. Back solve (with safe guard)
Output: fitting degree and coefficients

Figure 5.4: Work flow of local fitting

degree of fitting requires k-ring stencil, then we will retrieve k+1 layers of ghost elements, to guarantee that one each processor, not only vertices on the partition boundary get the same stencils, but also the ghost vertices has exact the same normal estimation as in serial.

5.2 Hermite-style High-order Surface Reconstruction

The WALF method described in the Section 5.1.1 has two main issues. First, the method may be inaccurate (due to lack of points in the stencils) or return lower-order accuracy (degradation due to safeguards against oscillations) [61] for relatively coarse meshes. Secondly, it does not guarantee G^0 continuity along sharp features (ridges and corners) leading to leaks near such features. In this section, we address the first issue by extending the purely point-based least-squares fitting to include normal-based information, similar to Hermite interpolation. We describe the extended formulation, including new criteria for selecting the neighborhood and the weighting schemes, and present the proof of consistency and stability for Hermite-style approach. When coupled with the weighted-averaging scheme in WALF, we then obtain Hermite-style WALF method.

Point and Normal based Local Polynomial Fittings

Given a smooth surface defined in the global xyz coordinate system, it can be transformed into a local uvw coordinate system by translation and rotation. Let the origin of the local frame be at $\boldsymbol{x_0} = [x_0, y_0, z_0]^T$ (note that for convenience we treat points as column vectors) on the surface. Both coordinate frames are orthonormal right-hand systems. Suppose the exact normal direction of the surface is given. Let $\boldsymbol{n}_0$ be an approximate normal direction with unit length at the origin, then the *uv*-plane is the tangent plane of the surface at the origin. $\boldsymbol{t}_0^1$ and $\boldsymbol{t}_0^2$ are the unit vectors in the global coordinate system along the positive directions of the u and v axes, respectively. For each point $\boldsymbol{x}$, it could be transformed to a point $p(\boldsymbol{u}) = [u, v, f(\boldsymbol{u})] = \boldsymbol{Q}_0^T(\boldsymbol{x} - \boldsymbol{x}_0)$, $\boldsymbol{u} = (u, v)$, where

$$\boldsymbol{Q}_0 = \begin{bmatrix} \boldsymbol{t}_0^1 & \boldsymbol{t}_0^2 & \boldsymbol{n}_0 \end{bmatrix} \tag{5.2.1}$$

In general, f is not a one-to-one mapping over the whole surface, but if the *uv* plane is close to the tangent plane at a point $\boldsymbol{x}_0$, then f would be one-to-one in a neighborhood of $\boldsymbol{x}_0$. We refer to this function $f(u) : \mathbb{R}^2 \to \mathbb{R}$ as a local height function at $\boldsymbol{x}_0$ in the uvw coordinate frame. The Jacobian of $p(\boldsymbol{u})$ with respect to $\boldsymbol{u}$, denoted by $\boldsymbol{J}$ is then

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{p}_u & \boldsymbol{p}_v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ f_u & f_v \end{bmatrix}. \tag{5.2.2}$$

The *first fundamental form* of the surface is given by the quadratic form $I(d\boldsymbol{u}) = d\boldsymbol{u}^{\boldsymbol{T}} G d\boldsymbol{u}$ where

$$\boldsymbol{G} = \boldsymbol{J}^T\boldsymbol{J} = \begin{bmatrix} 1 + f_u^2 & f_u f_v \\ f_u f_v & 1 + f_v^2 \end{bmatrix}. \tag{5.2.3}$$

Let $g = \det(\boldsymbol{G}) = 1 + f_u^2 + f_v^2$, and $\ell = \sqrt{g}$. Then, $\ell = \|p_u \times p_v\|$. In the local coordinate system, the unit normal to the surface is then

$$\boldsymbol{w} = \frac{\boldsymbol{p}_u \times \boldsymbol{p}_v}{\ell} = \frac{1}{\ell} \begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix}. \tag{5.2.4}$$

Given the normal direction $\boldsymbol{n_i}$ at point $\boldsymbol{x}_i$ (in global coordinate) or $[u_i, v_i, f_i]^T$ (in local coordinate) , we have the following equation

$$(Q_0^T \cdot \boldsymbol{n}_i) \times \tilde{\boldsymbol{w}}_i \approx 0. \tag{5.2.5}$$

Here $\boldsymbol{w}_i = \frac{\boldsymbol{p}_u \times \boldsymbol{p}_v}{\ell}(\boldsymbol{u}_i) = \frac{1}{\ell} \begin{bmatrix} -f_u(\boldsymbol{u}_i) \\ -f_v(\boldsymbol{u}_i) \\ 1 \end{bmatrix}$ is the normal at $\boldsymbol{x}_i$ in the local coordinate system, and $\tilde{\boldsymbol{w}}_i$ is its approximation by the degree *d* polynomial fitting. Let $\boldsymbol{Q}_0^T \cdot \boldsymbol{n}_i = [\alpha_i, \beta_i, \gamma_i]^T$. Then, using 5.2.5 we obtain

$$\begin{aligned} \gamma_i \cdot f_u(\boldsymbol{u}_i) &\approx -\alpha_i, \\ \gamma_i \cdot f_v(\boldsymbol{u}_i) &\approx -\beta_i. \end{aligned}$$

Given a positive integer *d*, a function *f(u)* can be approximated to *d+1* order accuracy around the origin $\boldsymbol{u}_0$ as

$$f(\boldsymbol{u}) = \sum_{p=0}^{d} \sum_{j,k \geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!} + O(\|\boldsymbol{u}\|^{d+1}), \tag{5.2.6}$$

assuming *f* has *d + 1* continuous derivatives.

We have the point location based approximation

$$\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u_i^j v_i^k}{j!k!} \approx f_i. \tag{5.2.7}$$

along with the normal direction based approximation

$$\begin{aligned} \sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \cdot j \cdot \frac{u_i^{j-1} v_i^k}{j!k!} &\approx -\frac{\alpha_i}{\gamma_i}, \\ \sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \cdot k \cdot \frac{u_i^{j} v_i^{k-1}}{j!k!} &\approx -\frac{\beta_i}{\gamma_i}. \end{aligned} \tag{5.2.8}$$

Using these three equations, we get the following linear system

$$\boldsymbol{V}\boldsymbol{X} \approx \boldsymbol{F}, \tag{5.2.9}$$

where $\boldsymbol{X}$ is an n-vector composed of c_{jk}, and $\boldsymbol{V}$ is $3m \times n$ (n is determined by degree of fitting polynomial), and $\boldsymbol{F}$ is a $3m$-vector. For example, a degree 2 fitting over m points in the local stencil results in the following $\boldsymbol{V}$ and $\boldsymbol{F}$

$$\boldsymbol{V} = \begin{pmatrix} 1 & u_1 & v_1 & u_1^2 & u_1v_1 & v_1^2 \\ 1 & u_2 & v_2 & u_2^2 & u_2v_2 & v_2^2 \\ & \cdots & & & \cdots & \\ 1 & u_m & v_m & u_m^2 & u_mv_m & v_m^2 \\ 0 & 1 & 0 & 2u_1 & v_1 & 0 \\ 0 & 1 & 0 & 2u_2 & v_2 & 0 \\ & \cdots & & & \cdots & \\ 0 & 1 & 0 & 2u_m & v_m & 0 \\ 0 & 0 & 1 & 0 & u_1 & 2v_1 \\ 0 & 0 & 1 & 0 & u_2 & 2v_2 \\ & \cdots & & & \cdots & \\ 0 & 0 & 1 & 0 & u_m & 2v_m \end{pmatrix}, \boldsymbol{F} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \\ -\frac{\alpha_1}{\gamma_1} \\ -\frac{\alpha_2}{\gamma_2} \\ \vdots \\ -\frac{\alpha_m}{\gamma_m} \\ -\frac{\beta_1}{\gamma_1} \\ -\frac{\beta_2}{\gamma_2} \\ \vdots \\ -\frac{\beta_m}{\gamma_m} \end{pmatrix}. \tag{5.2.10}$$

Remark. If γ_i is zero, then it means $f(\boldsymbol{u})$ will have foldings around $\boldsymbol{x}_i$. This could be avoided by carefully choosing the weights for each neighboring vertices, such that all the vertices that cause the foldings will have zero weights and thus will be eliminated from the system; see following discussion.

Remark. The Hermite-style fitting is interpolatory, if the local polynomial passes through the origin of fittings, i.e.,

$$\begin{aligned} f(\boldsymbol{u}_0) &= 0; \\ f_u(\boldsymbol{u}_0) &= 0; \\ f_v(\boldsymbol{u}_0) &= 0. \end{aligned}$$

Thus, the local fitting polynomial must be at least second order:

$$f(\boldsymbol{u}) = \sum_{p=2}^{\infty} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!}. \tag{5.2.11}$$

Weighted Least Squares Formulation

Equation (5.2.9) could be solved under the framework of *weighted linear least square* to minimize the weighted norm,

$$\min_{\boldsymbol{X}} \left\| \boldsymbol{\Omega}(\boldsymbol{V}\boldsymbol{X} - \boldsymbol{F}) \right\|_2, \tag{5.2.12}$$

where $\boldsymbol{\Omega} = \text{diag}(w_1, w_2, \ldots, w_{3m})$ is the *weighting matrix* of size $3m \times 3m$. The weighting matrix $\boldsymbol{\Omega}$ assigns priorities to different rows of the linear system corresponding to different points that are being fit. Note that $\boldsymbol{\Omega}$ has no effect on the solution if $\boldsymbol{V}$ is a nonsingular square matrix, but different $\boldsymbol{\Omega}$ would lead to different solutions for rectangular matrices. In general, for the ith ($i \leq m$) row corresponding to the ith point $\boldsymbol{x}_i$, it is desirable to assign its weight w_i to some larger value if $\boldsymbol{x}_i$ is close to the origin of the local coordinate system $\boldsymbol{x}_0$, or a smaller value (or even zero) of $\boldsymbol{x}_i$ is far away from $\boldsymbol{x}_0$ or its normal $\boldsymbol{n}_i$ is too far from the normal $\boldsymbol{n}_0$ at $\boldsymbol{x}_0$. In particular, we choose the weight at the ith vertex as

$$w_i = \frac{\gamma_i^+}{\left(\|\boldsymbol{u}_i\|^2/h + \epsilon\right)^{d/2}}, \tag{5.2.13}$$

where $\gamma_i^+ \equiv \max(0, \boldsymbol{n}_i^T \boldsymbol{n}_0)$, $h \equiv \sum_{i=1}^{m} \|\boldsymbol{u}_i\|^2/m$, and $\epsilon \approx 0.01$. The factor γ_i^+ serves as a safeguard against drastically changing normals for coarse meshes or nonsmooth areas. The denominator $\left(\|\boldsymbol{u}_i\|^2/h + \epsilon\right)^{d/2}$ prevents the weights from becoming too large at points that are too close to $\boldsymbol{u}_0$ and is approximately equal to $w_i \approx (\|\boldsymbol{u}_i\|^2/h)^{-d/2}$. Because $\boldsymbol{\Omega}$ allows the flexibility to underweight (and even filter out) undesirable points, we use a simple procedure to select points based on mesh connectivity when constructing the linear system.

For equations of type (5.2.7), we could use weight ω_i like (5.2.13) to assign a priority to the *i*th vertex in local stencil, by scaling the *i*th row of $\boldsymbol{V}$, $i = 1, \cdots m$. However, for equations of type (5.2.8), applying the same weights $\{\omega_i\}$ would cause numerical instability issue. Since equations (5.2.8) correspond to the derivatives of the local fitting polynomial, their magnitude are one order lower than equation (5.2.7) in terms of local step size. Thus, the linear system could be ill-conditioned if we directly apply the same weights. This could be avoided if we scale the weights with local step size. For (5.2.8) of derivative against u, we apply weight as $\omega_i h_u$, where h_u is the step size along the direction of *u*. Similarly we apply $\omega_i h_v$ to equations corresponding to derivatives against *v*.

As suggested in [61], the weighting matrix $\boldsymbol{\Omega}$ scales the rows of $\boldsymbol{V}$ and hence cannot improve the scaling of the columns. This issue is resolved by introducing a column *scaling matrix* $\boldsymbol{T}$ and then impose the minimization as

$$\min_{\boldsymbol{x}} \|\boldsymbol{\Omega V T Y} - \boldsymbol{\Omega F}\|_2, \tag{5.2.14}$$

where $\boldsymbol{X} = \boldsymbol{TY}$. Unlike $\boldsymbol{\Omega}$, the scaling matrix $\boldsymbol{T}$ does not change the exact solution of $\boldsymbol{X}$. However, $\boldsymbol{T}$ can significantly improve the conditioning of the linear system and in turn improve the accuracy in the presence of rounding errors. In general, given a weighting matrix $\boldsymbol{\Omega}$, let $\boldsymbol{v}_i$ denote the ith column vector of $\boldsymbol{\Omega V}$. We choose

$\boldsymbol{T}$ to be the diagonal matrix with entries

$$T_{ii} = 1/\|\boldsymbol{v}_i\|_2 \tag{5.2.15}$$

for $i = 1, \ldots, n$. Let $\boldsymbol{A} = \boldsymbol{\Omega V T}$. This scaling matrix approximately minimizes the condition number of $\boldsymbol{A}$ (see [51, p. 265] and [114]). Note that the scaling matrix $\boldsymbol{T}$ cannot improve the condition number of $\boldsymbol{V}$ if the ill-conditioning is caused by the lack of points or some unfortunate selection of points, which is a well-known issue [75]. It can be alleviated by including additional points in the fitting if possible. However, if the number of points is fixed, solving this ill-conditioned system is similar to solving an under-constrained linear system. Instead of using the truncated SVD to solve this linear system, we use the QR factorization. Let the QR of $\boldsymbol{A}$ be

$$\boldsymbol{A} = \boldsymbol{QR}, \tag{5.2.16}$$

where $\boldsymbol{Q}$ is $3m \times n$ with orthonormal column vectors, $\boldsymbol{R}$ is a $n \times n$ upper-triangular matrix. If $\boldsymbol{A}$ has full-rank, the condition number of $\boldsymbol{A}$ is the same as that of $\boldsymbol{R}$, which can be estimated accurately and efficiently using a variant of back substitution. If the condition number $\boldsymbol{R}$ is large (e.g., $\geq 10^6$), we then reduce the degree of the fitting by removing the last few columns in $\boldsymbol{R}$ that correspond to highest derivatives. Let $\tilde{\boldsymbol{Q}}$ and $\tilde{\boldsymbol{R}}$ denote the reduced matrices. The final solution of $\boldsymbol{X}$ is given by

$$\boldsymbol{X} = \boldsymbol{T}\tilde{\boldsymbol{R}}^{-1}\tilde{\boldsymbol{Q}}^T\boldsymbol{\Omega F}, \tag{5.2.17}$$

where $\tilde{\boldsymbol{R}}^{-1}$ denotes a back substitution step. Compared to the solution based on SVD, this procedure is more accurate asymptotically, as it gives highest priority to the lower-order coefficients of the polynomial while maintaining good scaling of the matrix, and at the same time it is more efficient than SVD. If the degree of fitting is reduced due to either an ill-conditioned $\tilde{\boldsymbol{R}}$ or insufficient number of points in the stencil, a cure could be increasing the size of the stencil so that the system is more stable.

Stencil Selection

The stencil selection is important for the accuracy and efficiency of the local polynomial fittings, since a too large stencil tends to cause overfitting, while a too small stencil delivers low order accuracy. For a degree *d* polynomial fitting for 3D imbedded surface, we have $n = (d+1)(d+2)/2$ unknowns to determine, which means we need at least n equations. The WALF method generally chooses a $(d+1)/2$ ring stencil for origin interpolatory fittings. For coarse meshes, fitting over a large stencil could lose certain local features, which implies that WALF could over-smooth the underlying surface if high-order reconstruction is applied. However, Hermite-style fitting requires a much more compact local stencil due to extra information from normal directions. For a 3D imbedded surface, three independent equations could be established from each vertex: one from vertex position, and two from normal direction. As a result, only one-third of the vertices required by WALF are good enough for FAH-WALF. We typically choose 1-*ring* for FAH-WALF interpolation with degree under 4. In case of insufficient vertices, our local stencil selection procedure follows an adaptive approach such that at least $n$vertices are selected. Table 5.1 shows a typical stencil selection for WALF and FAH-WALF, where the number of vertices in local stencil is calculated as an average of vertices in local

Table 5.1: Comparison of average stencil sizes for WALF and FAH-WALF.

	degree 2		degree 3		degree 4		degree 5		degree 6	
#unknowns	6		10		15		21		28	
	#rings	#verts	#rings	#verts	#rings	#verts	#rings	#verts	#rings	#verts
WALF	1.5	12.6	2	18.4	2.5	29.6	3	37.5	3.5	52.7
FAH-WALF	1	6.8	1	6.8	1	6.8	1.5	12.6	2	18.4

stencil of a random sampled vertices of an unstructured triangular torus mesh with 336 triangles.

5.3 Accuracy and Stability of Hermite-style Fittings

The theoretical foundation for Hermite-style surface is based on that of the local least squares polynomial fitting. Let *h* denote the average edge length of the mesh. We consider the errors in terms of *h*. Then, the following proposition could be established:

Theorem 6. *Given a set of points $[u_i, v_i, \tilde{f}_i]$ that interpolate a smooth height function f or approximate f with an error of O(h^{d+1}) and the corresponding gradients are approximated to O(h^d). Assume the point distribution and the weighting matrix are independent of the mesh resolution, and the condition number of the scaled matrix $\boldsymbol{VT}$ is bounded by some constant. The degree-d weighted least squares fitting approximates c_{jk} to O($h^{d-j-k+1}$).*

The proof of this proposition could be obtained following the analysis of [62]. Note that Hermite-style interpolation satisfies the assumption of this proposition if the nodal position could be approximated to $O(h^{d+1})$ and the normals are approximated to $O(h^d)$. This means that $\boldsymbol{Q}_0 \cdot \boldsymbol{n}_i = [\alpha_i, \beta_i, \gamma_i]^T$ is approximated in $O(h^d)$.

The gradient of local height function equals $-\alpha_i/\gamma_i$ or $-\beta_i/\gamma_i$. We assume that γ_i is bounded away from zero, so f_u and f_v are approximated to $O(h^d)$. Note that a necessary condition for the accuracy is that the condition number of the scaled matrix $\boldsymbol{A}$ must be bounded. We achieve well-conditioning by either expanding the neighborhood or reducing the degree of fitting if the condition number is determined to be large, and in turn guarantee both accuracy and stability.

Continuous, High-order Surface Reconstruction

Hermite-style reconstruction delivers high-order accuracy. To construct a G^0 continuous surface, we use the same strategy as in WALF, by averaging of local fittings by equation (5.1.1). Under the same assumption as Proposition 6, we obtain the following property of FAH-WALF.

Proposition 7. *Given a mesh whose vertices approximate a smooth surface Γ with an error of O(h^{d+1}) and the normal directions are also approximated to $O(h^d)$, then distance between each point on the WALF reconstructed surface and its closest point on Γ is O($h^{d+1} + h^6$).*

We refer the readers to [61] for the proof of the proposition. The bound of h^6 is due to the discrepancy of local coordinate systems at different vertices.

5.4 High-order Curve Reconstruction

Point-based Curve Reconstruction

The high-order curve reconstruction follows the same underlying principles as the high-order surface reconstruction as described in previous section. For completeness, we provide a brief description here. Given a smooth curve in the global *xyz* coordinate system, each of its coordinates (x, y, z) can be parameterized as $(x(t), y(t), z(t))$ where t is a parameter in the tangent direction $\boldsymbol{t}$ at the origin of fitting $\boldsymbol{x_0} = [x_0, y_0, z_0]^T$. The Taylor series expansion of any coordinate function f about the origin $t_0 = 0$ is

$$f(t) = \sum_{j=0}^{\infty} c_j \frac{t^j}{j!} \tag{5.4.1}$$

where $c_j = \frac{d^j}{dt^j} f(0)$. Given a positive integer *d*, the function *f(t)* can be approximated to *d+1* order accuracy about the origin t_0 as

$$f(t) = \sum_{j=0}^{d} c_j \frac{t^j}{j!} + O(\|t\|^{d+1}) \tag{5.4.2}$$

assuming *f* has *d + 1* continuous derivatives. We can find the coefficients c_j's of the approximated polynomial $f(t)$ by fitting it to a set of points sampling a small patch of the smooth curve. For purely point based fitting, plugging in each given point $[t_i, f_i]$ into the fitting, we obtain an approximate equation

$$\sum_{i=0}^{d} c_j \frac{u_i^j}{j!} \approx f_i. \tag{5.4.3}$$

Here we have $n = d + 1$ unknowns, i.e. c_j for $0 \leq j \leq d$. The rectangular linear system formed is solved using a robust QR factorization as described in the previous section. To obtain a G^0 continuity of the reconstructed curve, we perform a weighted averaging of the local fittings (WALF) at the vertices. In particular, consider an edge composed of vertices $\boldsymbol{x}_i$, $i = 1, 2$, and any point $\boldsymbol{p}$ in the edge. For each vertex $\boldsymbol{x}_i$, we obtain a point $\boldsymbol{q}_i$ for $\boldsymbol{p}$ from the local fitting at $\boldsymbol{x}_i$, by projecting $\boldsymbol{p}$ onto its tangent direction. Let ξ_i, $i = 1, 2$ denote the barycentric coordinates of $\boldsymbol{p}$ within the edge, with $\xi_i \in [0, 1]$ and $\sum_{i=1}^{2} \xi_i = 1$. We define

$$\boldsymbol{q}(\boldsymbol{u}) = \sum_{i=1}^{2} \xi_i \boldsymbol{q}_i(\boldsymbol{u}) \tag{5.4.4}$$

as the approximation to point $\boldsymbol{p}$. WALF constructs a G^0 continuous curve, as can be shown using the properties of finite-element basis functions.

Hermite-style Curve Reconstruction

At a point $\boldsymbol{p}_0\,(x_0, y_0, z_0)$ on the curve $\varUpsilon$, let $Q = \left[\hat{\boldsymbol{t}}_0,\ \hat{\boldsymbol{n}}_0,\ \hat{\boldsymbol{b}}_0\right]$ be a local coordinate system. One option is to use the trio: tangent, principal normal, bi-normal. Another option is to start with the tangent direction to find two orthogonal vectors. The initial tangent could be an input or an approximation from the curve mesh. Now, any point $\boldsymbol{p}\,(x, y, z)$ near $\boldsymbol{p}_0\,(x_0, y_0, z_0)$ can be parameterized as

$$\boldsymbol{p}\left(x,y,z\right)\xrightarrow{Q}\boldsymbol{p}\left(u,s(u),t(u)\right) \tag{5.4.5}$$

$$\text{i.e.,}\begin{bmatrix}x\\y\\z\end{bmatrix}=Q\begin{bmatrix}u\\s(u)\\t(u)\end{bmatrix}+\begin{bmatrix}x_0\\y_0\\z_0\end{bmatrix} \tag{5.4.6}$$

where $u=\left(\boldsymbol{x}-\boldsymbol{x}_0\right).\hat{\boldsymbol{t}}_0$. The local tangent is

$$\boldsymbol{t}=\frac{1}{\sqrt{1+\left(s'(u)\right)^2+\left(t'(u)\right)^2}}\begin{bmatrix}1\\s'(u)\\t'(u)\end{bmatrix} \tag{5.4.7}$$

We want to approximate $s(u)$ and $(t(u)$ to high-order by using degree q Taylor series i.e.,

$$s(u)\approx\sum_{k=0}^{q}c_k\frac{u^k}{k!} \tag{5.4.8}$$

$$t(u)\approx\sum_{k=0}^{q}d_k\frac{u^k}{k!} \tag{5.4.9}$$

The standard point based formulation finds the coefficients c_k and d_k for $k=0,\ldots,q$ by fitting 5.4.8 and 5.4.9 to a set of points $\boldsymbol{p}_i\left(x_i,y_{i,}z_i\right)$ in the vicinity of $\boldsymbol{p}_0$ i.e.,

$$\sum_{k=0}^{q} c_k \frac{u_i^k}{k!} = s_i \tag{5.4.10}$$
$$\sum_{k=0}^{q} d_k \frac{u_i^k}{k!} = t_i \tag{5.4.11}$$

where

$$u_i = (\boldsymbol{x}_i - \boldsymbol{x}_0)\,.\hat{\boldsymbol{t}}_0$$
$$s_i = s(u_i) = (\boldsymbol{x}_i - \boldsymbol{x}_0)\,.\hat{\boldsymbol{n}}_0$$
$$t_i = t(u_i) = (\boldsymbol{x}_i - \boldsymbol{x}_0)\,.\hat{\boldsymbol{b}}_0$$

Let $\boldsymbol{t}_i$ be the given (in global coordinates) tangent at $\boldsymbol{p}_i\,(x_i, y_{i,} z_i)$ and $\tilde{\boldsymbol{t}}_i$be the local approximated tangent. Thus, $Q^T\boldsymbol{t}_i \times \tilde{\boldsymbol{t}}_i \approx 0$. Let $Q^T\boldsymbol{t}_i = [\alpha_i, \beta_i, \gamma_i]^T$. Thus, $s_u \approx \frac{\beta_i}{\alpha_i}$ and $t_u \approx \frac{\gamma_i}{\alpha_i}$. Including the first order derivatives information into the Taylor series approximation, we get

$$\sum_{k=0}^{q} c_k \frac{u_i^k}{k!} = s(u_i)$$
$$\sum_{k=0}^{q-1} c_{k+1} \frac{u_i^k}{k!} = s_u(u_i)$$

and

$$\sum_{k=0}^{q} d_k \frac{u_i^k}{k!} = t(u_i)$$
$$\sum_{k=0}^{q-1} d_{k+1} \frac{u_i^k}{k!} = t_u(u_i)$$

.

The coefficients c_k and d_k can be obtained by solving the system $\boldsymbol{A}\begin{bmatrix} \boldsymbol{c} & | & \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{s} & | & \boldsymbol{t} \end{bmatrix}$.

5.5 Feature-aware Reconstruction

In the presence of ridge vertices or feature curves embedded in a curve/surface mesh (see Figure 5.5), one-sided WALF reconstructions are required since the curve/surface is no longer smooth along such ridge vertices and/or feature curves. These different reconstruction results along the same ridges/features would introduce gaps which leads to ambiguity in point projections for points very near or on the ridges/features. If we could one of these reconstruction results as projection method, then this would cause discontinuity along the ridges/features. In this section, we propose a linear correction based approach to blend the reconstructed curves and neighboring surfaces to recover G^0 continuity, while preserving the high-order of accuracy.

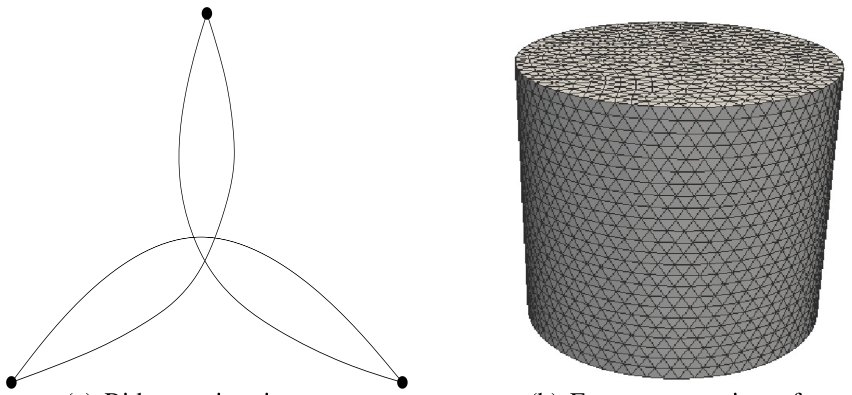

(a) Ridge vertices in curve (b) Feature curves in surface

Figure 5.5: Ridges and Features

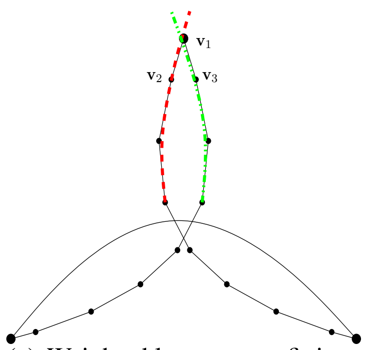
5.5.1 Ridge-aware Curve Reconstruction

For 1D curve mesh, if we assume that the input vertices have accurate coordinates and therefore enforce the local fitting to pass the center, then the reconstructed curve is guaranteed to be G^0 continuous. To see this, for example in Figure 5.6(a), $\boldsymbol{v}_1$ is a ridge vertex. Since the curve is no longer smooth at $\boldsymbol{v}_1$, we need one-sided local polynomial fittings centered at $\boldsymbol{v}_1$ along the directions of $\boldsymbol{v}_1 \to \boldsymbol{v}_2$ and $\boldsymbol{v}_1 \to \boldsymbol{v}_3$. Assume the fitting results centered at $\boldsymbol{v}_1$ are $\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2}$ and $\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3}$. Then for any point $\boldsymbol{x}$ in edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$, we have reconstruct

$$\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2 \rangle}(\boldsymbol{x}) = (1-\alpha)\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2} + \alpha \boldsymbol{p}_{\boldsymbol{v}_2,\boldsymbol{v}_1} \tag{5.5.1}$$

where $\boldsymbol{x} = (1-\alpha)\boldsymbol{v}_1 + \alpha \boldsymbol{v}_2$. Similarly for any point $\boldsymbol{y}$ in edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_3 \rangle$, we have

$$\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3 \rangle}(\boldsymbol{y}) = (1-\beta)\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3} + \beta \boldsymbol{p}_{\boldsymbol{v}_3,\boldsymbol{v}_1} \tag{5.5.2}$$



(a) Weighted least square fittings

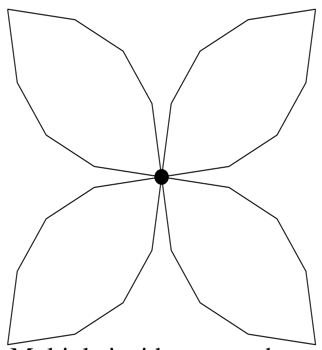

(b) Multiple incident smooth curves

Figure 5.6: Ridge-aware Reconstruction

where $\boldsymbol{y} = (1-\beta)\boldsymbol{v}_1 + \beta\boldsymbol{v}_3$. Since the local polynomials are all smooth, so both $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{x})$ and $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{y})$ are smooth. Since we enforce $\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2}(\boldsymbol{v}_1) = \boldsymbol{v}_1$ and $\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3}(\boldsymbol{v}_1) = \boldsymbol{v}_1$, then

$$\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) = \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1) = \boldsymbol{v}_1. \tag{5.5.3}$$

Thus the reconstruction is continuous at ridge vertex $\boldsymbol{v}_1$. It's easy to see that the reconstruction is continuous everywhere. For accuracy, the weighted average preserves the accuracy of local fittings.

However, when the coordinates of vertices in initial mesh are noisy, it's reasonable to apply least square fittings rather then to enforce local fittings to pass the centers. In such situation, then Equation 5.5.3 no longer holds which means we may lose G^0 continuity at ridge vertices. Since we have two different estimations of location of $\boldsymbol{v}_1$, $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1)$ and $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1)$, one fair estimation for $\boldsymbol{v}_1$ is

$$\bar{\boldsymbol{v}}_1 = \frac{1}{2}(\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2}(\boldsymbol{v}_1) + \boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3}(\boldsymbol{v}_1)). \tag{5.5.4}$$

Although a more general form of $\bar{\boldsymbol{v}}_1$ is $\bar{\boldsymbol{v}}_1 = (1-\omega_1-\omega_2)\boldsymbol{v}_1 + \omega_1 \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) + \omega_2 \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1))$, for simplicity we use the above form. For regular vertex like $\boldsymbol{v}_2$ in Figure 5.6(a), then we use the following estimation

$$\bar{\boldsymbol{v}}_2 = \boldsymbol{p}_{\boldsymbol{v}_2,\boldsymbol{v}_1}(\boldsymbol{v}_2) \tag{5.5.5}$$

Assume $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) = \bar{\boldsymbol{v}}_1 - \boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2}(\boldsymbol{v}_1)$, $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_2) = \bar{\boldsymbol{v}}_2 - \boldsymbol{p}_{\boldsymbol{v}_2,\boldsymbol{v}_1}(\boldsymbol{v}_2)$, then we have the following correction for $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}$:

$$\begin{aligned} \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{x}) &= (1-\alpha)\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_2} + \alpha \boldsymbol{p}_{\boldsymbol{v}_2,\boldsymbol{v}_1} + \\ &\quad (1-\alpha)\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) + \alpha\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_2). \end{aligned}$$

Apparently $\bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}$ is continuous along edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle$ since $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{x})$ is continuous, $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1)$ and $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_2)$ are constant. Similarly we have the correction of $\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}$:

$$\begin{aligned} \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{x}) &= (1-\beta)\boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3} + \beta \boldsymbol{p}_{\boldsymbol{v}_3,\boldsymbol{v}_1} + \\ &\quad (1-\beta)\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1) + \beta\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_3) \end{aligned}$$

where $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1) = \bar{\boldsymbol{v}}_1 - \boldsymbol{p}_{\boldsymbol{v}_1,\boldsymbol{v}_3}(\boldsymbol{v}_1)$, $\delta\boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_3) = \bar{\boldsymbol{v}}_3 - \boldsymbol{p}_{\boldsymbol{v}_3,\boldsymbol{v}_1}(\boldsymbol{v}_3)$.

Proposition 8. *The ridge aware correction is continuous and preserves accuracy when non-interpolatory local fittings are used.*

Proof. We could tell

$$\bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) = \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_3\rangle}(\boldsymbol{v}_1) = \bar{\boldsymbol{v}}_1. \tag{5.5.6}$$

Therefore the corrected reconstruction is continuous at ridge vertex $\boldsymbol{v}_1$. Since for regular node $\boldsymbol{v}_2$, $\bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{v}_2) = \bar{\boldsymbol{v}}_2 = \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_1}(\boldsymbol{v}_2)$ the continuity preserves at $\boldsymbol{v}_2$, similarly for $\boldsymbol{v}_3$. Even if $\bar{\boldsymbol{v}}_2$ is also a ridge vertex, the continuity is still preserved at $\boldsymbol{v}_2$ since we apply correction $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{v}_2)$. The accuracy is preserved since the correction part is the difference of high-order estimations. □

Remark 9. In general, for a ridge vertex $\boldsymbol{v}_0$, there could be multiple (>2) incident smooth curves (see Figure 5.6(b)). In such case, one correction for $\boldsymbol{v}_0$ is

$$\bar{\boldsymbol{v}}_0 = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{p}_{\boldsymbol{v}_0, \boldsymbol{v}_i}(\boldsymbol{v}_0). \tag{5.5.7}$$

For each edge $\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle$, we should apply the following k-order correction at $\boldsymbol{v}_0$, that is

$$\begin{aligned} \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle}(\boldsymbol{x}) &= (1-\alpha^i)\boldsymbol{p}_{\boldsymbol{v}_0, \boldsymbol{v}_i} + \alpha^i \boldsymbol{p}_{\boldsymbol{v}_i, \boldsymbol{v}_0} + \\ &\quad (1-\alpha^i)^k \delta \boldsymbol{p}_{\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle}(\boldsymbol{v}_0) + \alpha^{ik} \delta \boldsymbol{p}_{\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle}(\boldsymbol{v}_i) \end{aligned}$$

where $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle}(\boldsymbol{v}_0) = \bar{\boldsymbol{v}}_0 - \boldsymbol{p}_{\boldsymbol{v}_0, \boldsymbol{v}_i}(\boldsymbol{v}_0)$, and $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_0, \boldsymbol{v}_i \rangle}(\boldsymbol{v}_i) = 0$ if $\boldsymbol{v}_i$ is regular vertex.

5.5.2 Feature-aware Surface Reconstruction

To recover continuity in surface reconstruction, it will be more challenging since both ridge vertices and feature curves could be present in surface. Here we first introduce correction along features to recover G^0 continuity under the assumption that the input vertices coordinates are accurate. We will reduce the continuity recovery problem to this situation when the input coordinates are noisy.

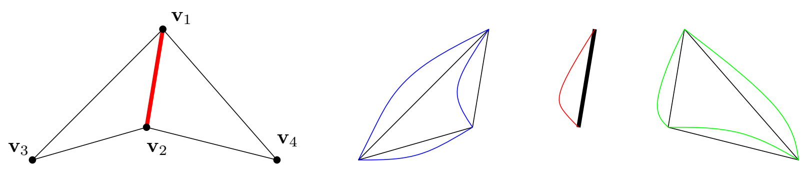


Figure 5.7: The lack of continuity across feature edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$

Interpolatory fittings

When the input coordinates are accurate, it's necessary to enforce the local curve and surface polynomial fittings to pass the center. Thus as abovementioned, the reconstructed feature curves are continuous. As proved in Section , the surface reconstruction is continuous both inside each triangle and along the non-feature edges. However, the continuity might not be guaranteed along feature edges. As illustrated in Figure 5.7, along feature edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$, if we take the fitting from feature curve $\boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x})$ as reconstruction of $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$, then the reconstruction is no longer continuous since reconstruction of $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$ and $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4$ could give different result than $\boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x})$ along $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$.

Assume that $\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ is the local fitting centered at $\boldsymbol{v}_1$ which fits the mesh on the side of $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$. Similarly we define $\boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_3 \boldsymbol{v}_1}(\boldsymbol{x})$, $\boldsymbol{p}_{\boldsymbol{v}_3, \boldsymbol{v}_1 \boldsymbol{v}_2}(\boldsymbol{x})$. Then the reconstruction on $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$ is

$$\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) = \xi \boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) + \eta \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_3 \boldsymbol{v}_1}(\boldsymbol{x}) + (1 - \xi - \eta) \boldsymbol{p}_{\boldsymbol{v}_3, \boldsymbol{v}_1 \boldsymbol{v}_2}(\boldsymbol{x}) \qquad (5.5.8)$$

where $\boldsymbol{x} = \xi \boldsymbol{v}_1 + \eta \boldsymbol{v}_2 + (1 - \xi - \eta) \boldsymbol{v}_3$ and the reconstruction on $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4$ is

$$\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{y}) = \xi \boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{y}) + \eta \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_4 \boldsymbol{v}_1}(\boldsymbol{y}) + (1 - \xi - \eta) \boldsymbol{p}_{\boldsymbol{v}_4, \boldsymbol{v}_1 \boldsymbol{v}_2}(\boldsymbol{y}). \qquad (5.5.9)$$

This means we have three estimations for points along edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$. For simplicity, one fair reconstruction for $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$ is

$$\bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x}) = \frac{1}{3}(\boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x}) + \boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) + \boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{x})). \tag{5.5.10}$$

Here $\boldsymbol{x} = (1-\alpha)\boldsymbol{v}_1 + \alpha \boldsymbol{v}_2$, and

$$\begin{aligned}
\boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x}) &= (1-\alpha)\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2}(\boldsymbol{x}) + \alpha \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_1}(\boldsymbol{x}) \\
\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) &= (1-\alpha)\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) + \alpha \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_3 \boldsymbol{v}_1}(\boldsymbol{x}) \\
\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{x}) &= (1-\alpha)\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{x}) + \alpha \boldsymbol{p}_{\boldsymbol{v}_2, \boldsymbol{v}_4 \boldsymbol{v}_1}(\boldsymbol{x}).
\end{aligned}$$

For the fitting on triangle $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$, the following correction along edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$ is required:

$$\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{z}) = \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{z}) - \boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{z}) \tag{5.5.11}$$

where $\boldsymbol{z} = (1-\alpha)\boldsymbol{v}_1 + \alpha \boldsymbol{v}_2$. Since we enforce all local fittings to pass the center, we have $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{v}_1) = \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{v}_2) = 0$. However, the correction defined above has compact support only on $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$. We need to extend the correction on $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$ by the following:

$$\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle, \boldsymbol{v}_3}(\boldsymbol{x}) = (\xi + \eta) \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\frac{\xi}{\xi + \eta}\boldsymbol{v}_1 + \frac{\eta}{\xi + \eta}\boldsymbol{v}_2) \tag{5.5.12}$$

where $\boldsymbol{x} = \xi \boldsymbol{v}_1 + \eta \boldsymbol{v}_2 + (1 - \xi - \eta)\boldsymbol{v}_3$ in $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$. It's easy to see $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle, \boldsymbol{v}_3} = \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}$ on edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$ since $\boldsymbol{x} = \xi \boldsymbol{v}_1 + \eta \boldsymbol{v}_2$ and $\xi + \eta = 1$. Moreover,

$\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x})$ has the following properties:

$$\begin{aligned}\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x}) = 0 \quad \text{where } \boldsymbol{x} \text{ in } \langle \boldsymbol{v}_1, \boldsymbol{v}_3\rangle \\ \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x}) = 0 \quad \text{where } \boldsymbol{x} \text{ in } \langle \boldsymbol{v}_1, \boldsymbol{v}_4\rangle\end{aligned} \tag{5.5.13}$$

since for point in $\langle \boldsymbol{v}_1, \boldsymbol{v}_3\rangle$, $\boldsymbol{x} = \xi \boldsymbol{v}_1 + (1-\xi)\boldsymbol{v}_3$, $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x}) = \xi \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\boldsymbol{v}_1) = 0$, similarly for edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_4\rangle$. Meanwhile, the correction is continuous and graded: when $\boldsymbol{x}$ is close to $\boldsymbol{v}_3$, $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x})$ goes to zero. Actually we could apply *k*-order correction:

$$\delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x}) = (\xi+\eta)^k \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle}(\frac{\xi}{\xi+\eta}\boldsymbol{v}_1 + \frac{\eta}{\xi+\eta}\boldsymbol{v}_2) \tag{5.5.14}$$

to allow less correction inside $\Delta \boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3$. Similarly we could define $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle}$ and $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_3,\boldsymbol{v}_1\rangle}$, then we have

$$\begin{aligned}\delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle,\boldsymbol{v}_1}(\boldsymbol{x}) &= (1-\xi)\delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle}(\frac{\eta}{1-\xi}\boldsymbol{v}_2 + \frac{1-\xi-\eta}{1-\xi}\boldsymbol{v}_3) \\ \delta \boldsymbol{p}_{\langle \boldsymbol{v}_3,\boldsymbol{v}_1\rangle,\boldsymbol{v}_2}(\boldsymbol{x}) &= (1-\eta)\delta \boldsymbol{p}_{\langle \boldsymbol{v}_3,\boldsymbol{v}_1\rangle}(\frac{1-\xi-\eta}{1-\eta}\boldsymbol{v}_3 + \frac{\xi}{1-\eta}\boldsymbol{v}_1).\end{aligned}$$

These edge-based corrections share the same type of properties: continuous, graded, and don't affect corrections on other edges. Notice that if $\langle \boldsymbol{v}_2, \boldsymbol{v}_3\rangle$ is not a feature edge, then $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle} = 0$ and thus $\delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle,\boldsymbol{v}_1}(\boldsymbol{x}) = 0$ for any $\boldsymbol{x}$ in $\Delta \boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3$. Therefore we could have the following corrected reconstruction for $\Delta \boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3$:

$$\bar{\boldsymbol{p}}_{\Delta \boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3}(\boldsymbol{x}) = \boldsymbol{p}_{\Delta \boldsymbol{v}_1\boldsymbol{v}_2\boldsymbol{v}_3}(\boldsymbol{x}) + \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1,\boldsymbol{v}_2\rangle,\boldsymbol{v}_3}(\boldsymbol{x}) + \delta \boldsymbol{p}_{\langle \boldsymbol{v}_2,\boldsymbol{v}_3\rangle,\boldsymbol{v}_1}(\boldsymbol{x}) + \delta \boldsymbol{p}_{\langle \boldsymbol{v}_3,\boldsymbol{v}_1\rangle,\boldsymbol{v}_2}(\boldsymbol{x}). \tag{5.5.15}$$

Proposition 10. *The feature aware correction for surface mesh is continuous and* *preserves accuracy when interpolatory local fittings are used.*

Proof: $\bar{\boldsymbol{p}}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ is continuous inside $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$ since $\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ and all corrections are continuous inside $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3$. The continuity along edge $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$ is apparent also since $\bar{\boldsymbol{p}}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) = \boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) + \delta \boldsymbol{p}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle, \boldsymbol{v}_3}(\boldsymbol{x}) = \bar{\boldsymbol{p}}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}(\boldsymbol{x})$ for $\boldsymbol{x}$ in $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle$. For regular edges like $\langle \boldsymbol{v}_2, \boldsymbol{v}_3 \rangle$, since all the corrections are zero along the regular edges, then $\bar{\boldsymbol{p}}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x}) = \boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ then the continuity is obvious. Since $\boldsymbol{p}_{\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ is high-order accurate and the corrections are difference between high-order estimations, the corrected reconstruction is of high-order accuracy. The above argument could be applied to $\Delta \boldsymbol{v}_1 \boldsymbol{v}_2 \boldsymbol{v}_4$ in a similar way.

Non-interpolatory fittings

When the input coordinates are inaccurate, it's better to infer the coordinates of input vertices from least square fittings. For vertices on feature curve, we not only get the fitting for curve, but also a couple of one sided surface fittings centered at these vertices, like $\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2}(\boldsymbol{x})$, $\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_3}(\boldsymbol{x})$ and $\boldsymbol{p}_{\boldsymbol{v}_1, \boldsymbol{v}_2 \boldsymbol{v}_4}(\boldsymbol{y})$ for $\boldsymbol{v}_1$ in Figure 5.7. For ridge vertices, we could get several one-sided curve fittings, and probably also a couple of one sided surface fittings. One straightforward way to estimate the coordinates of vertices on feature curves and ridge vertices is by averaging:

$$\bar{\boldsymbol{v}} = \frac{1}{n}\left(\sum_{i=1}^{k} \boldsymbol{p}_{\boldsymbol{v}, \boldsymbol{v}_2^{c_i}}(\boldsymbol{v}) + \sum_{j=1}^{n-k} \boldsymbol{p}_{\boldsymbol{v}, \boldsymbol{v}_2^{s_j} \boldsymbol{v}_3^{s_j}}(\boldsymbol{v})\right) \tag{5.5.16}$$

where c_i is the *i*-th incident smooth curve to $\boldsymbol{v}$ ($k = 0$ if $\boldsymbol{v}$ is not a ridge vertex) and s_j is the *j*-th incident surface to $\boldsymbol{v}$ (for a manifold surface, we could have two such incident surfaces). For regular vertex, we use the least square fitting result of $\boldsymbol{v}$ as estimation:

$$\bar{\boldsymbol{v}} = \boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{v}). \tag{5.5.17}$$

After the above correction, even the local polynomial fittings (curve and/or surface) are no longer continuous at vertex $\boldsymbol{v}$. To fix this, we apply the following corrections for local fittings:

- for each smooth curve c incident to $\boldsymbol{v}$, we have

$$\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{x}) = \boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{x}) + \delta\boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{x}) \tag{5.5.18}$$

  where $\delta\boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{x}) = (1-\alpha)(\bar{\boldsymbol{v}} - \boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{v}))$ for any $\boldsymbol{x}$ in $\langle\boldsymbol{v}, \boldsymbol{v}_2^c\rangle$ and $\boldsymbol{x} = (1-\alpha)\boldsymbol{v} + \alpha\boldsymbol{v}_2^c$. Notice that $\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{v}) = \bar{\boldsymbol{v}}$, $\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^c}(\boldsymbol{x})$ is continuous along $\langle\boldsymbol{v}, \boldsymbol{v}_2^c\rangle$ and preserves accuracy.

- for each smooth surface s incident to $\boldsymbol{v}$, we have

$$\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{x}) = \boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{x}) + \delta\boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{x}) \tag{5.5.19}$$

  where $\delta\boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{x}) = \xi(\bar{\boldsymbol{v}} - \boldsymbol{p}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{v}))$ for any $\boldsymbol{x}$ in $\Delta\boldsymbol{v}\boldsymbol{v}_2^s\boldsymbol{v}_3^s$ and $\boldsymbol{x} = \xi\boldsymbol{v} + \eta\boldsymbol{v}_2^s + (1-\xi-\eta)\boldsymbol{v}_3^s$. Notice that $\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{v}) = \bar{\boldsymbol{v}}$, $\tilde{\boldsymbol{p}}_{\boldsymbol{v},\boldsymbol{v}_2^s\boldsymbol{v}_3^s}(\boldsymbol{x})$ is continuous along $\Delta\boldsymbol{v}\boldsymbol{v}_2^s\boldsymbol{v}_3^s$ and preserves accuracy.

From now on, all the local fittings centered at the same vertex $\boldsymbol{v}$ have the same estimation $\bar{\boldsymbol{v}}$. In addition, all the corrected local fittings are continuous and preserves high-order accuracy. This reduces the problem of recovering continuity to the same one as in the interpolation section. We could apply the same strategy mentioned above to recover G^0 continuity while preserving accuracy.

Proposition 11. *The feature aware correction for surface mesh is continuous and preserves accuracy when non-interpolatory local fittings are used.*

Proof: After corrections at ridge vertices and vertices on feature curves, all the local fittings are continuous along their compact support and pass the corrected center vertices. In addition, they all preserve the high-order accuracy of original local fittings. Therefore we could reduce the G^0 recovery problem to the one when the local fittings are interpolatory. After the same type of corrections as above for interpolatory fittings, the reconstruction is continuous and preserves accuracy according to Proposition 10.

Chapter 6

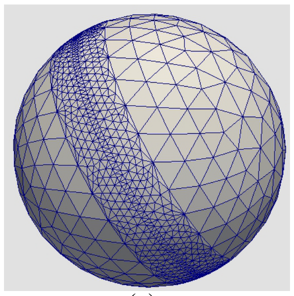
Numerical Results and Applications

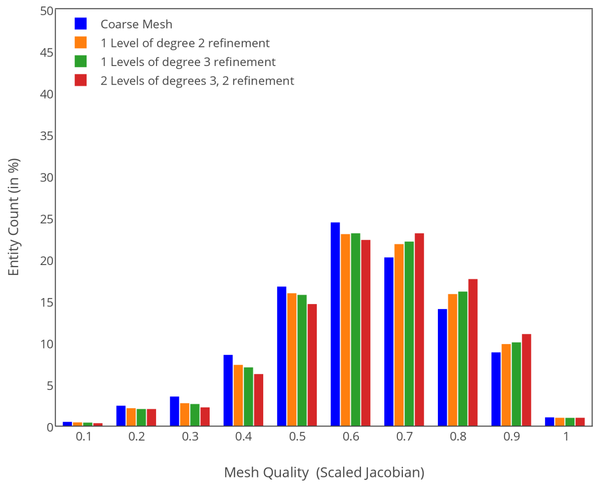
6.1 Numerical Results of Uniform Mesh Refinement

We present numerical results to demonstrate the effectiveness of uniform mesh refinement, in terms of computational efficiency of the parallel framework and its effect on mesh quality. We also demonstrate the application of the developed capability to study convergence properties of different point projection schemes for various mesh hierarchies as well as its application to multigrid method.

6.1.1 Mesh Quality Under Uniform Mesh Refinement

We first study the effect of uniform mesh refinement on the mesh quality. Since uniform refinement of tetrahedral meshes do not produce congruent sub-tetrehedra, we use it as our test case. An initial coarse tetrahedral sphere mesh with 23636 tets and 4793 vertices (shown in Fig. 6.1(a)) was refined using three strategies all starting with the coarse mesh:


(a)

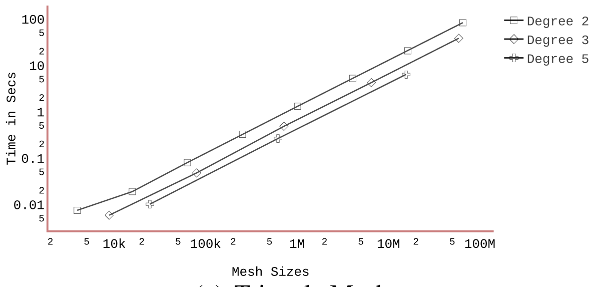


(b)

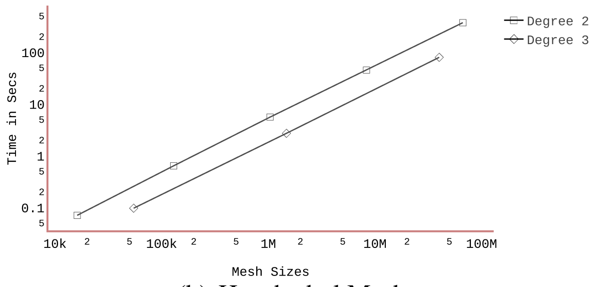
Figure 6.1: Left: A tetrahedral mesh. Right: Distribution of mesh quality for an initial coarse mesh and various degree of refinements.

1. one degree-2 refinement,
2. one degree-3 refinement, and
3. first a degree-3 refinement followed by a degree-2 refinement.

We use scaled Jacobian as the mesh quality measure. Figure 6.1(b) shows the distribution of the mesh quality measure for the meshes obtained using the three strategies along with the initial mesh. In all the three cases, the overall mesh quality improved with the shortest-diagonal approach [112], and degree-3 refinement delivers similar and even slightly better quality improvement because there are more intermediate octahedra in degree-3 refinement.



(a) Triangle Meshes



(b) Hexahedral Meshes

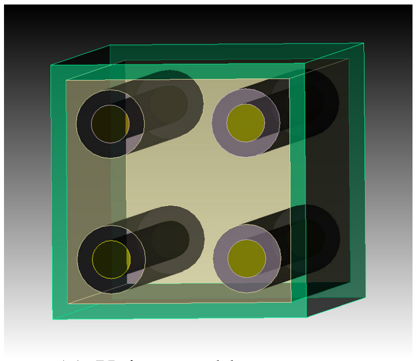
Figure 6.2: Time taken to generate hierarchies in serial for different degree of refinements.

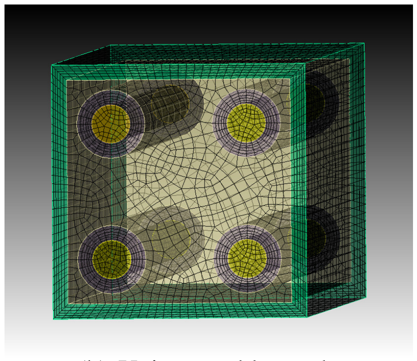
6.1.2 Computational Efficiency

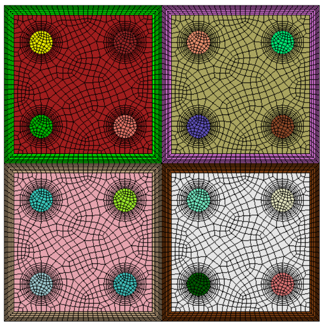
We now report the computational efficiency results of the mesh hierarchy generation algorithm. In Figure 6.2, the serial run times for generating hierarchies using different supported degree of refinements on two representative meshes for two and three dimensions are shown. In Figure 6.2(a), we start with a triangle mesh with 1000 entities and generate hierarchies with degrees 2, 3 and 5. Clearly, the results confirm that higher-degree of refinement reach greater resolution much faster. Similarly, in Figure 6.2(b), an initial hexahedral mesh with 2048 entities is used to generate hierarchies with degrees 2 and 3. We conclude that if a deep hierarchy is required, a degree 2 refinement per level would give a gradually increasing mesh with more levels. On the other hand, high resolution can be reached very quickly with small hierarchies using high-degree refinement. These serial tests were performed on a Mac computer with 2.3 GHz Intel Core i7 processor and 16GB RAM.

In order to perform the weak scaling studies of the uniform refinement algorithm, we use the Reactor Geometry (and mesh) Generator (**RGG** [58]) tool developed as part of the generic mesh generation framework **MeshKit** [109], which encapsulates the workflow of creating a nuclear reactor core geometry and meshes ready for computational analysis in state-of-art physics solvers. Designed with the nuclear engineer in mind, RGG guides the engineer through the process of designing fuel pins, ducts, and assemblies, and then the layout of the reactor core and mesh generation process. We designed a simple unit rectangular lattice based assembly consisting of four fuel pins with six boundary layers around them as shown in Figures 6.3(a) and 6.3(b). The total assembly generation including the geometry and mesh generation was done only once, which took about 29 seconds. The mesh generation was performed using Cubit. These unit assemblies were used to create composite assemblies increasing linearly with the number of processors in a pre-decided pattern as shown in Figure 6.3(c) and Figure 6.3(d) with two levels of degree 2 of further refinement to increase the resolution. The initial mesh contained about 19,800 hexahedrons with approximately 158K and 1.27M hexes in the subsequent refinement levels. The tests were performed on the Blues cluster at Argonne National Laboratory, which has 310 nodes, 16 cores/node with Intel Sandy Bridge processor and 64GB RAM per node.

The weak scalability results are shown in Figure 6.4. The merge based shared interface resolution algorithm for UMR scales similar to the ideal case for up to 512 processors. However, we see a drop in the efficiency around 1024 processors which might be due to an overloaded test platform and needs further investigation. The time taken to resolve the shared entities for the initial mesh is almost an order of magnitude faster than those taken for the two levels of refinement. This is because the time for shared interface resolution for UMR involves resolution of approximately 4 and 16 times of the initial number of entities on the shared entities. The mesh refinement by itself scales perfectly.


(a) Unit assembly geometry.


(b) Unit assembly mesh.


(c) 2x2 configuration.

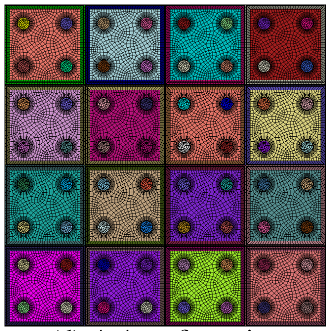
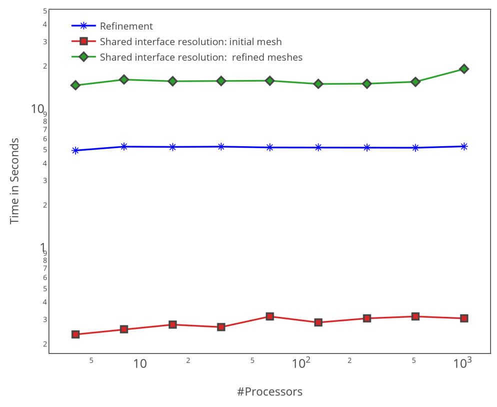

(d) 4x4 configuration

Figure 6.3: The initial mesh on each processor and creation of the whole assemblies from the unit.



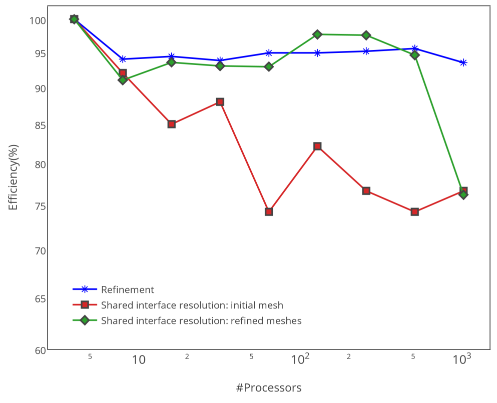


Figure 6.4: Weak scaling studies for the mesh hierarchy generation algorithm using the RGG tool with two levels of uniform refinement on Blues. The left and right figure shows the time in seconds and efficiency as the number of processors increases with fixed problem size.

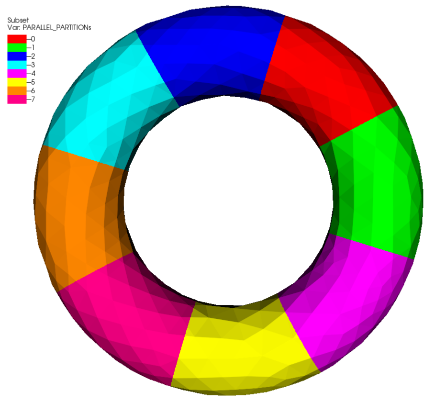


Figure 6.5: An example torus mesh with eight partitions used for the scalability study of mesh hierarchy generation.

Table 6.1: The times in seconds for the refinement, resolution of shared entities using the merge and optimized algorithms as the number of processors increases.

	Times in Seconds		
#Partitions	Refine	Merge	Optimized
1	9.98	0	0
2	5.31	14.13	0.078
4	2.45	7.4	0.063
8	1.3	3.8	0.073

Next, we present preliminary results comparing the merge-based and optimized resolve shared interface algorithms. We use a torus with approximately 2.6M vertices and 5.3M triangles as an initial mesh with one level of degree-2 refinement. Figure 6.5 shows an example mesh with eight partitions. Table 6.1 lists the times taken in seconds by the refinement and the two interface resolution algorithms for up to eight cores. Clearly, the optimized resolve shared algorithm shows a performance improvement of almost two orders of magnitude. However, we see no scaling as the core counts are increased as observed for the merge-based algorithm. The underlying reason behind is that the communication pattern (2 shared processors) and the total number of entities (approx. 2K edges) on the shared interface does not change for these lower core counts. As a result, for the optimized algorithm, the amount of work remains exactly the same compared to the merge-based algorithm. Currently, more rigorous tests are being performed for different distribution patterns on more number of processors on Blues.

6.1.3 Demonstration of UMR Capabilities in Discrete Solvers

The parallel UMR capability is useful in generating hierarchy of meshes to perform convergence studies and for creating optimal multigrid preconditioners for elliptic PDE solvers. A multigrid Poisson solver written using the PETSc-MOAB (DMMoab) interface that leverages the scalability of both the codes and specifically utilizing UMR is presented here for computing order of accuracy efficiently. Figure 6.6 shows that the Poisson solver with an inhomogeneous source term in 2-D and using a Method of Manufactured Solution (sinusoidal exact solution $\sin(\pi x)\cdot\sin(\pi y)$) yields the expected second order convergence (linear Lagrange continuous Galerkin FEM). These results were generated by successively refining the mesh through UMR and solved with Geometric Multigrid (GMG). As a proof of principle, we have performed some preliminary experimental study on the impact of high-order projection on the accuracy of the PDE solutions on curved geometries. From our experimental using MATLAB (Figure 6.14, 6.15), we have observed significant improvements in the overall accuracy with high-order reconstructions compared to using only piecewise linear reconstructions during mesh refinement.

In terms of computational efficiency, the iteration convergence of geometric multigrid-based preconditioner for a Poisson solver using a Generalized Finite Difference (GFD) method is also shown in Figure 6.7 and comparison to standard black-box algebraic preconditioners including AMG, shows optimal reduction in iteration error independent of the mesh resolution or degrees-of-freedom when using Geometric multigrid preconditioners. Figure 6.7 also shows that GMG with 3 and 5 levels are comparable to a Full Multigrid (FMG) scheme, which is theoretically optimal for solving such elliptic PDE systems.

6.2 Numerical Results of Adaptive Mesh Refinement

6.2.1 Adaptive Finite Element Method (AFEM)

We implemented AMR for the re-entrant corner problem from [83] with the Kelly error indicator. The equation is

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0 \tag{6.2.1}$$

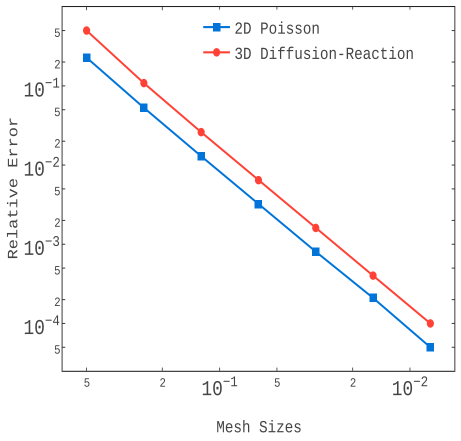


Figure 6.6: Mesh convergence of a 2D Poisson and 3D Diffusion-Reaction problem.

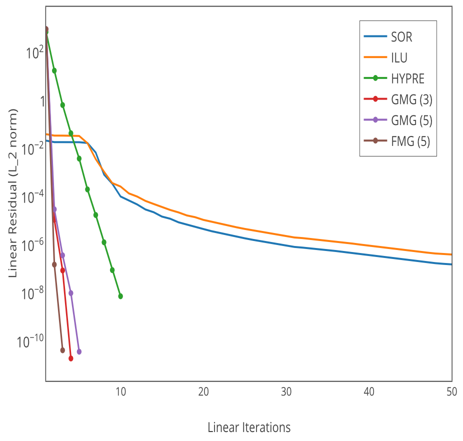


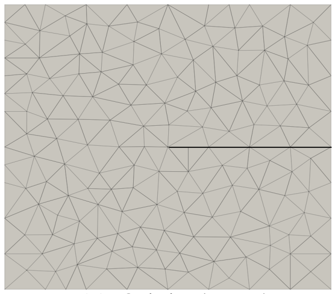
Figure 6.7: Comparison of convergence of multigrid and standard black-box preconditioners.

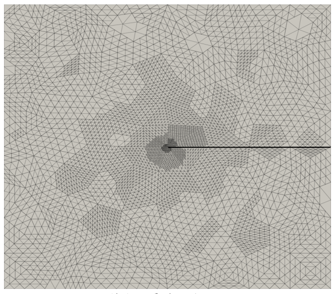
on domain $[-1,1]\times[-1,1]\backslash\{0\leq x\leq 1, y=0\}$. The boundary condition is $u=g$ and the exact solution is

$$u = r^{\frac{1}{2}}\sin\left(\frac{\theta}{2}\right) \tag{6.2.2}$$

where $r=\sqrt{x^2+y^2}$, $\theta=\tan^{-1}(y/x)\in[0,2\pi)$.

We apply 6 adaptive refinements over the original mesh and compare it with FEM on a mesh with uniform regular refinement. The results are shown in Figure 6.8, 6.9. The original mesh has a crack $\{0\leq x\leq 1, y=0\}$ along which the solution is not smooth. The refinement is centralized along the crack due to the non-smoothness of the solution, see Figure 6.8(b). The result in Figure 6.9(a) shows that the adaptive FEM approach delivers a better convergence rate than FEM over a uniformly refined mesh. The L_2 error is computed as $\int_\Omega |u-u_h|^2 dA$. The numerical results indicate that the same accuracy could be achieved with many less DOFs or number of elements.


(a) Original Mesh


(b) 6th AMR

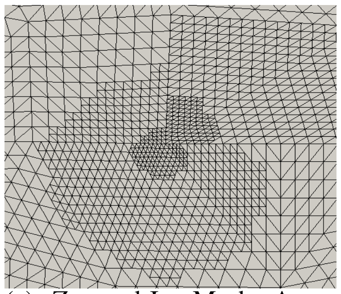
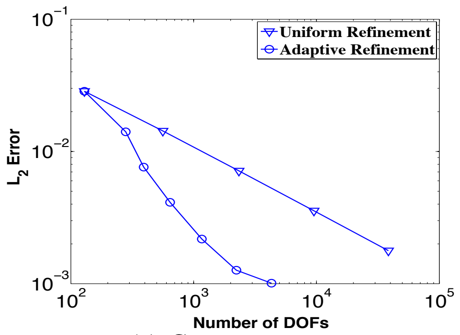
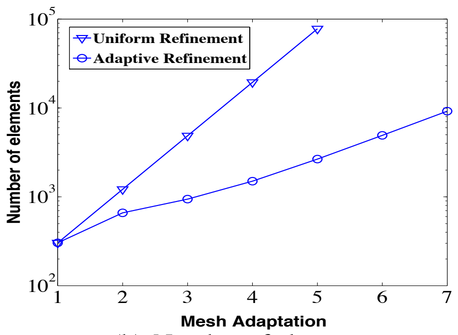

(c) Zoomed-In Mesh Around Center

Figure 6.8: AFEM: Mesh adaptation for re-entrant corner problem



(a) Convergence rate



(b) Number of elements

Figure 6.9: AFEM: Numerical results for re-entrant corner problem

6.2.2 Comparison with Pointer-based Data Structure

We will compare the storage for Hierarchical AHF with the pointer-based data structure in libMesh, as it is the most closely related to our data structure. Let C and V represent the set of cells and vertices of the given mesh, and let $|\cdot|$ denote the size of a set. For Hierarchical AHF, let C_1 and V_1 denote the cells and vertices, respectively, in the original, unrefined mesh. We will consider an implementation of Hierarchical AHF that includes the element connectivity (*elements*), vertex to parent element mapping (*v2pe*), sibling half-facet mapping (*sibhfcs*), element to parent element (*e2pe*) and element to child element mapping (*e2ce*). Since the vertex to incident half-facet mapping (*v2hf*) is optional, we will not include it in this analysis. Therefore, we have the following five maps which require the following number of entities:

element connectivity: $n_c = v_c\,|C|$
vertex to parent element map: $n_p = |V| - |V_1|$
sibling half-facet map: $n_s = f_c\,|C|$
element to parent map: $n_{ep} = |C| - |C_1|$
element to child map: $n_{ec} = |C|$

where v_c and f_c are the numbers of vertices per cell and the number of faces per cells, respectively. In general, the entities are stored as 32-bit integers. For the half-facet ID $\langle eid,\ lfid\rangle$, we encode it in a 32-bit integer. For the vertex to parent element map we store $\langle level,\ eid,\ lid\rangle$ as two 32-bit integers, one for $level$ and one for $\langle eid,\ lfid\rangle$. Thus the storage in bytes is

$$\begin{aligned} S_{\text{AHF32}} &= 4n_c + 8n_p + 4n_s + 4n_{ep} + 4n_{ec} \\ &= 4\left(2 + v_c + f_c\right)|C| - 4\,|C_1| + 8\,|V| - 8\,|V_1| \end{aligned} \tag{6.2.3}$$

If we were to store the entities as 64-bit integers, the storage would effectively double.

For each cell, libMesh stores the element connectivity and the "face neighbors" of the cells. Two cells are face neighbors if they share a side; in 1D a side is a vertex, in 2D a side is an edge, and in 3D a side is a face. Like Hierarchical AHF, adaptive mesh refinement and coarsening is central to libMesh and hence the cells and their ancestors are stored in a tree. Specifically, a pointer to the parent of an element and an array of pointers to its children (if any) are stored. In general, a d-dimensional element is refined into 2^d children of the same type except when dealing with pyramids, which are refined into pyramids and tetrahedra. For the sake of simplicity, we will use 2^d as the number of children of an element. Note that the level of an element is not stored in libMesh, since this can be found recursively from the parents. To store nodal information, libMesh has a node class. Each object in the node class stores the coordinates of the node, a unique global ID number and the degree of freedom indices. Since we are comparing the storage for the topological information of the mesh, we will consider the storage cost of the global ID number. Therefore we have 4 maps requiring the following number of entities:

element connectivity: $n_c = v_c|C|$

neighboring objects: $n_n = f_c|C|$

Table 6.2: Comparison of storage requirements in kilobytes of Hierarchical AHF and libMesh for a mesh in various stages of refinement on 32-bit and 64-bit architectures. On 64-bit architecture, one may store Hierarchical AHF with 32-bit integers or 64-bit integers.

	32-Bit Architecture		64-Bit Architecture		
	Hier AHF	libMesh	Hier AHF (32-Bit)	Hier AHF (64-Bit)	libMesh
Original Mesh	57.867	59.535	57.867	115.734	119.070
Refinement 1	85.070	86.105	85.070	170.141	172.210
Refinement 2	142.742	142.441	142.742	285.484	284.882
Refinement 3	233.516	231.266	233.516	467.031	462.531
Refinement 4	332.398	328.051	332.398	664.797	656.102
Refinement 5	446.680	440.180	446.680	893.359	880.359

hierarchical: $n_h = |C| + 2^d|C_r|$

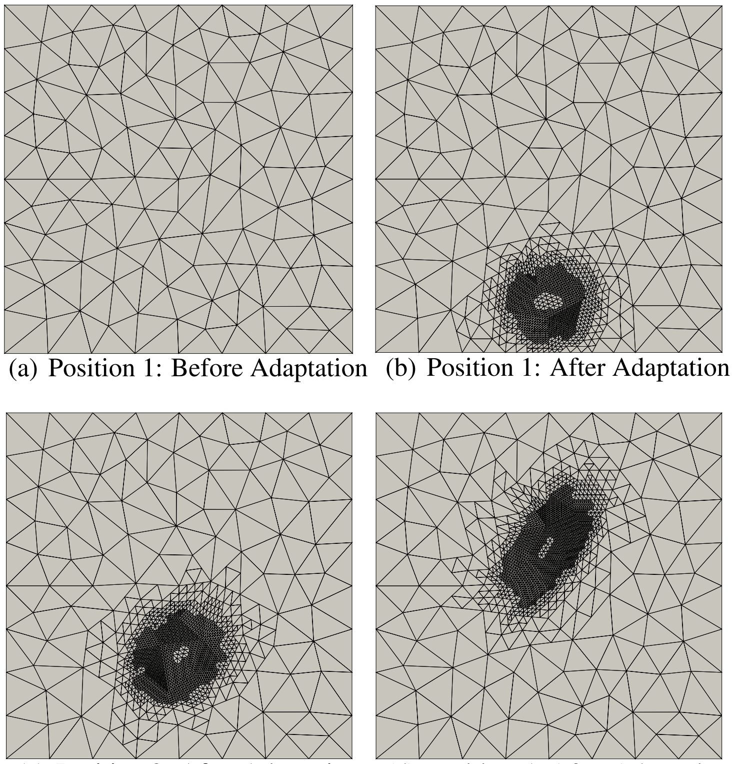
nodal: $n_v = |V|$

where C_r is the number of refined cells in the mesh. Since all these mappings are stored as pointers, if we assume 32-bit architecture, then we can estimate the storage in bytes as

$$\begin{aligned} S_{\text{libMesh32}} &= 4n_c + 4n_n + 4n_h \\ &= 4\left(1 + v_c + f_c\right)|C| + 4 \cdot 2^d\,|C_r| + 4\,|V| \end{aligned} \tag{6.2.4}$$

On 64-bit architectures, the storage would double.

As an example, Table 6.2 shows the storage required by the first six meshes used in Section 6.2.3 for Hierarchical AHF and libMesh. It can be seen that the memory cost of the two approaches is comparable if an integer has the same length as a pointer. However, Hierarchical AHF would require about half of the storage on modern 64-bit architectures for meshes with less than two billion elements per processor.


(a) Position 1: Before Adaptation (b) Position 1: After Adaptation

(c) Position 2: After Adaptation (d) Position 5: After Adaptation

Figure 6.10: Example triangular mesh during adaptive mesh refinement.

6.2.3 Mesh Adaptation

Hierarchical AHF supports both efficient refinement and derefinement. Here we illustrate the results in Figure 6.10. A function from [63], i.e. $x(x-1)y(y-1)e^{-100((x-0.5)^2+(y-0.117)^2)}$ over the domain $[0,1] \times [0,1]$ and its counter-clockwise rotations serve as a series of numerical solutions. The Kelly error estimator is used to drive the AMR algorithm to mark and adapt the mesh. The function is rotated 4 times, thus it has 5 positions, referred to as position 1 (i.e. original function), position 2, and so on. Starting from position 1, the original mesh (Figure 6.10(a)) is adapted by the solution at this position. Then the solution is rotated to position 2 and AMR is applied over the mesh in position 1 (Figure 6.10(b)), and a new mesh

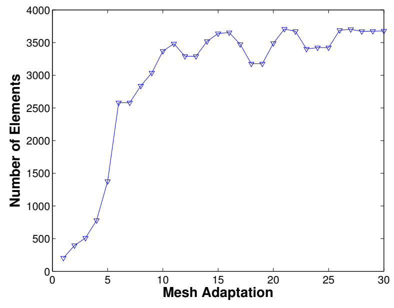


Figure 6.11: AMR for 2D triangular mesh: number of active cells

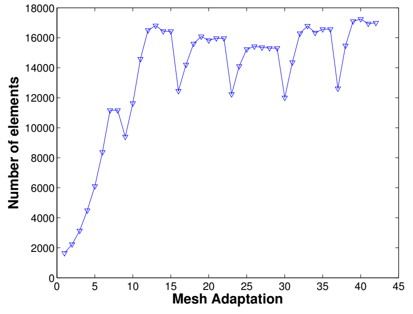
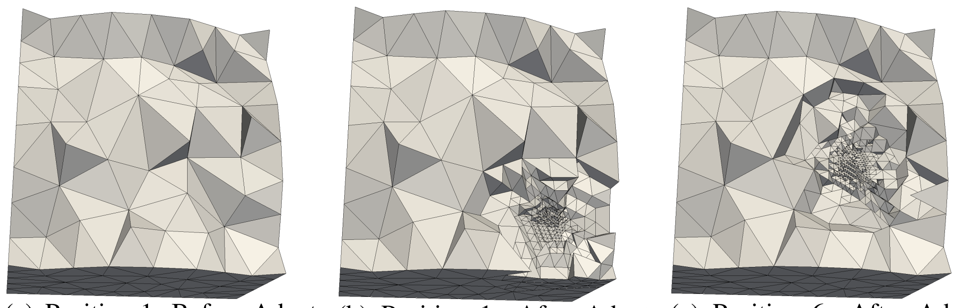


Figure 6.12: AMR for 3D tetrahedra mesh: number of active cells



(a) Position 1: Before Adaptation (b) Position 1: After Adaptation (c) Position 6: After Adaptation

Figure 6.13: Example AMR in 3D: cross-sections of tetrahedral mesh.

(Figure 6.10(c)) is obtained in position 2, so on and so forth. At each position we make 5 adaptations. The number of active elements at each adaptation can be found in Figure 6.11.

To demonstrate the mesh adaptation in 3D, we define a function $e^{-1000((x-x_c)^2+(y-y_c)^2+(z-z_c)^2)}$ over the unit cube $[0,1]^3$, and rotate its center (x_c, y_c, z_c) along the plane $z_c = 0.5$ for five times. We perform AMR based on the approximation errors to this series of functions. Figure 6.13 shows the cross-sections of the initial mesh and the mesh at three different stages. Similar to the 2D results, the number of elements remained approximately constant during the adaptation process; see Figure 6.12.

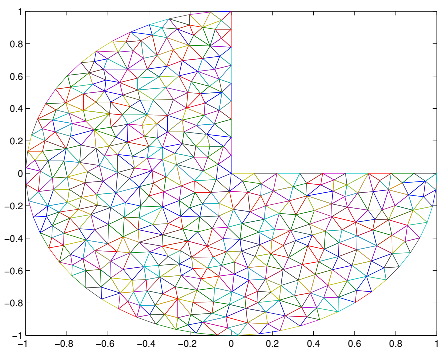
6.3 Numerical Results of High-order Reconstruction

In this section, we explore the importance of geometry in the accuracy of numerical PDE under mesh refinement, see related numerical result in Section 6.3.1. In addition, we present some preliminary numerical results with our parallel implementation of WALF in MOAB. We also conduct numerical experiment with FAH-WALF, especially in comparison with point-based WALF.

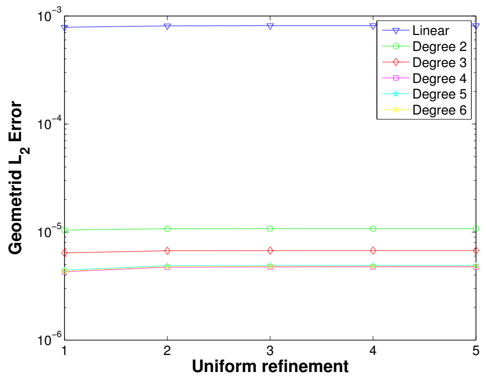
6.3.1 Geometry and PDE

Geometry plays an important role in solving PDEs. When one tries to achieve high-order accuracy via mesh refinement, the triangulation must be refined along the curved boundary, otherwise the geometric error will counteract the effect of mesh refinement. High-order reconstruction of the boundary geometry is effective in preserving the accuracy of solution under refinement. The numerical results indicate that geometric error matters in solving PDE.

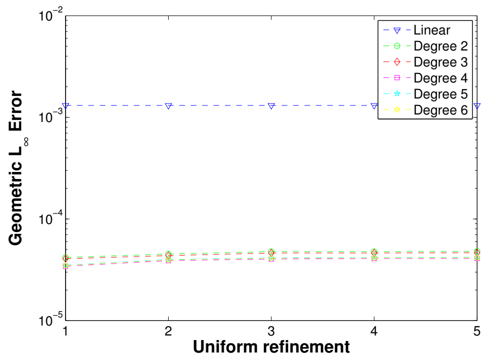
To measure the effect of geometry in solving PDEs, we conduct the following experiment. The geometry is two intersecting sphere as illustrated in Figure 6.15(a). The initial mesh 7551 vertices and 47086 tetrahedra while the surface has 668 triangles, 22 feature edges. The mesh is refined 3 times recursively with final mesh has 4024977 vertices and 24108032 tetrahedra. The new vertices introduce by refinement are first placed linearly interpolated boundary and then repositioned via high-order reconstruction of the boundary, or the exact geometry. Here, we adopt FAH-WALF method with degree 1 to 6 to reconstruct geometry from the original mesh. For each new mesh, the geometric error is measured along the boundary in L_∞ norm and plotted in Figure 6.15(b). Under uniform refinement, the reconstruc-



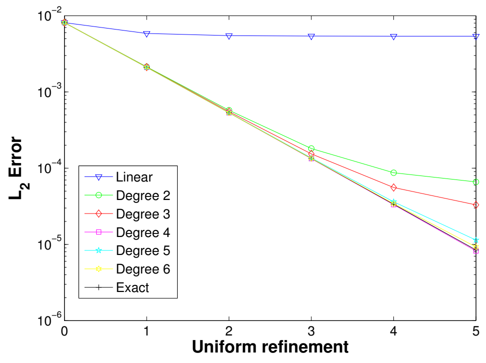
(a) Original mesh



(b) L_2 errors of reconstruction with various degrees of WALF



(c) L_∞ errors of WALF reconstruction

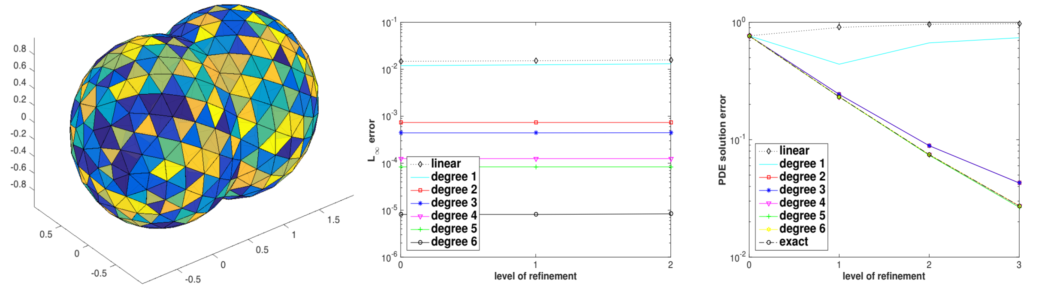


(d) L_2 error of Poisson equation solution under various reconstructed boundary and exact boundary

Figure 6.14: Convergence studies for Poisson equation with linear interpolated boundary, WALF reconstructed boundaries and exact boundary

tion error stays the same since we could only estimate geometry from original mesh. We have 8 sets of mesh sequences, corresponding to linear interpolation, degree 1 to 6 reconstruction and exact geometry.

On each mesh sequence, Poisson equation with analytical solution $u(x, y, z) = e^{x^2+y^2+z^2}$ is solved on the successively refined mesh and Dirichlet boundary condition is assumed. Since in practice, boundary condition is only available on exact surface geometry, so for the cases of inexact geometry the boundary condition is set as $u(\boldsymbol{x}) = f(\hat{\boldsymbol{x}})$. Here $\boldsymbol{x}$ is on the boundary of inexact geometry, $\hat{\boldsymbol{x}}$ is its projection



(a) Initial mesh (b) L_∞ geometric error (c) Error of PDE solution under various geometries

Figure 6.15: Geometry and PDE: geometric error affects the accuracy of PDE solution

onto exact geometry and f is the Dirichlet boundary condition. The solution error is calculated in terms of energy norm $\sqrt{\int_\Omega |u - \tilde{u}|^2 \, dx}$ where $\tilde{u}$ is the numerical solution and is plotted in Figure 6.15(c). We could observe that if the approximation of geometry is good enough, the convergence rate of FEM is recovered even if exact geometry is not known beforehand.

Remark. For completeness and readers' interest, we also conducted the same test for Poisson equation with solution $e^{x^2+y^2}$ on a 2-D mesh for 3-quarters disk; see Figure 6.14.

6.3.2 Parallel Surface Reconstruction in MOAB

The reconstruction algorithm WALF is implemented under submodule *Discrete Geometry* in MOAB and it supports high-order surface reconstruction in serial and parallel. We performed our experiment using a sequence of refined sphere meshes and torus meshes (mesh sizes are listed in Table 6.3). In this test, we randomly generate 10 points on each element of the mesh and then project them onto the high-order surface using WALF. We compute the error as the shortest distance from each ap-

Table 6.3: Mesh sizes of the sequence of sphere and torus meshes used for convergence studies.

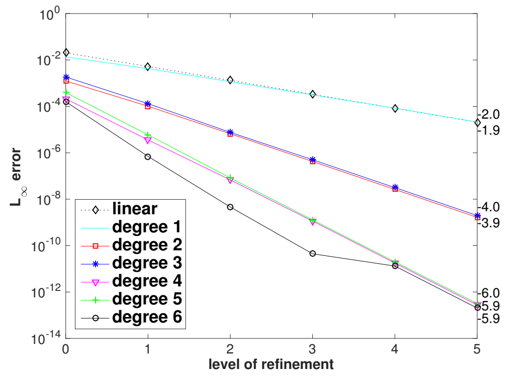
	#Vertices	#Triangles
Sphere Meshes	210	416
	834	1664
	3330	6656
	13314	26624
	53250	106496
	212994	425984

Torus Meshes	168	336
	672	1344
	2688	5376
	10752	21504
	43008	86016
	172032	344064

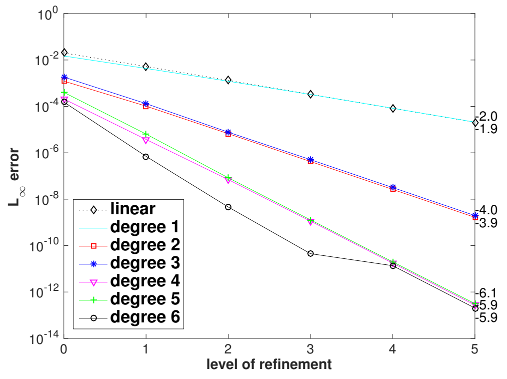
proximation to the exact geometry.

Figures 6.16(a), 6.16(b), 6.16(c) and 6.16(d) show the L_∞ errors of WALF for the sequence of sphere and torus meshes in serial and parallel. In the legend, the degree indicates the degree of polynomial fittings used by WALF, and "linear" indicates the error of linear interpolation. The average convergence rates are shown along the right of the plots, which was calculated as $\log(\text{error}_5/\text{error}_0)/\log(h_5/h_0)$, where error_i denotes the L_∞ error on ith mesh and h_i is the maximum edge length of the corresponding mesh.

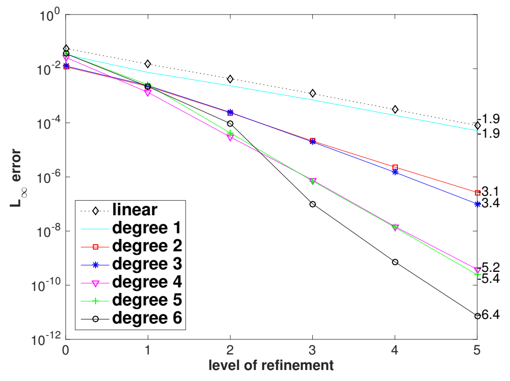
It is obvious that WALF could achieve high-order accurate geometry approximation, especially compared with the linear interpolation. WALF utilizes local polynomial fittings thereby the reconstruction algorithm could be easily parallelized. As long as sufficient ghost layers are provided, the reconstruction algorithm needs



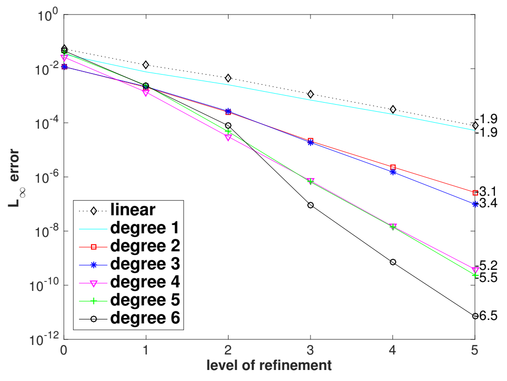
(a) Sphere-Serial



(b) Sphere-8 processors



(c) Torus-Serial



(d) Torus-8 processors

Figure 6.16: Convergence studies for point projection using high-order surface reconstruction using a sphere and torus mesh in serial and parallel.

Table 6.4: Numbers of ghost layers for WALF in parallel.

Degree of WALF	1	2	3	4	5	6
#Ghost Layers	2	2	3	3	4	4

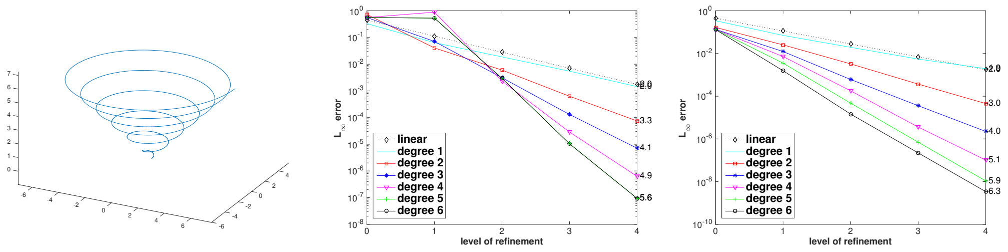
only local computation without any parallel communication. The required numbers of ghost layers for various degrees of WALF are listed in Table 6.4.

6.3.3 Accuracy of Hermite-style High-order Reconstructions

We report the convergence study of FAH-WALF and its comparison with WALF. The study is performed on a sequence of unstructured meshes of the same object. The sequence is obtained via uniformly refining original mesh recursively, and a *k* levels of refinement delivers *k+1* levels of mesh. Each mesh has exact geometry and normals (tangent vectors if the geometry is curve). In the test, 10 points on each facet of the mesh are randomly generated and then projected onto a high-order surface constructed using FAH-WALF or WALF.

The error of projection is measured as the distance from the projection onto estimated geometry to the projection onto exact geometry. The geometric error of high-order reconstruction is defined in L_∞ norm as $\max_{i=1}^{v} \|\tilde{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_i\|_2$, where $\tilde{\boldsymbol{x}}_i$ is the projection by FAH-WALF or WALF, and $\hat{\boldsymbol{x}}_i$ is the projection onto the exact geometry, and v is the number of test vertices. We show the average convergence rates along the right of the plot of reconstruction errors. The rate was calculated as $\log(\text{error}_{end}/\text{error}_{base})/\log(h_{end}/h_{base})$. Here *base* stands for the initial mesh and *end* stands for the finest mesh in sequence.

In current tests, exact vertex position and normal/tangent direction are given as



(a) Example conical helix (b) WALF reconstruction (c) Hermite-style reconstruction

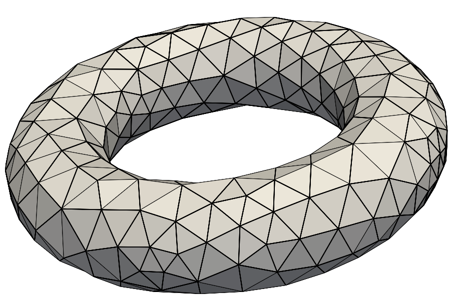
Figure 6.17: Curve Reconstruction

input for FAH-WALF. In practice, WALF does not require exact normal/tangent direction. Here, in our comparison, WALF also utilizes the exact normal direction.

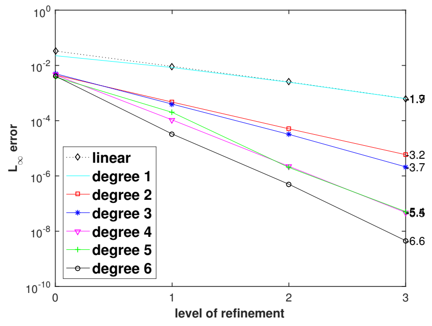
The first study is performed using a conical helix as in Figure 6.17(a) as geometry. The sequence of meshes in the test is obtained via 4 levels of refinement of a helix mesh with 50 vertices. From Figure 6.17(b),6.17(c), we could tell that both WALF and FAH-WALF could achieve the asymptotical convergence rate of (*d*+*1*). However, Hermite-style reconstruction tends to be more accurate. Especially, Hermite-style reconstruction is more robust for coarse meshes.

The following test is performed using a torus with in-radius of 0.7 and outer-radius 1.3 as geometry. Figure 6.18 shows the L_∞ error of FAH-WALF for a torus mesh which has 304 vertices and 608 triangles, and 3 levels of refinement which gives the finest mesh which has 19456 vertices and 38912 triangles. From the figure, it is obvious that quadratic and higher-degree fittings produce far more accurate results than linear interpolation.

Both WALF and FAH-WALF could achieve the convergence rate of (*d*+*1*) if the mesh is well resolved. However, for coarse mesh, FAH-WALF give smaller errors than WALF, as shown in Figure 6.17(b),6.17(c),6.19, although asymptotically they delivered similar convergence rate. From our observation, the reason for the better

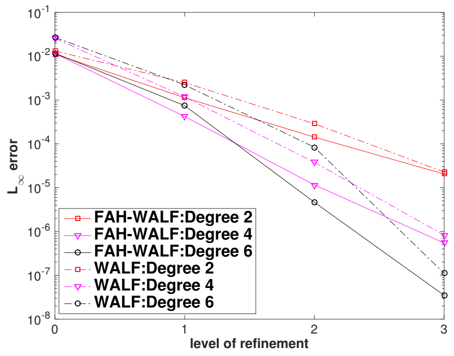


(a) Initial torus mesh: 304 vertices, 608 triangles

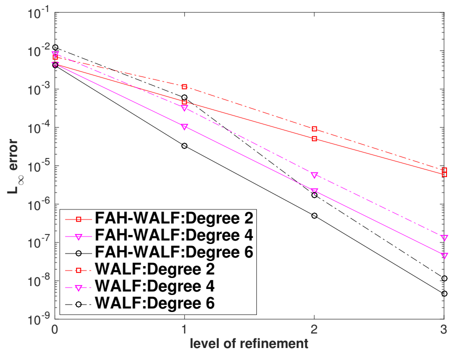


(b) L_∞ errors of FAH-WALF under mesh refinement for torus. FAH-WALF achieves $(d+1)$st accuracy for degree-d polynomials.

Figure 6.18: Convergence study of FAH-WALF



(a) L_2 error of two methods for initial torus mesh with 336 triangles



(b) L_2 error of two methods for initial torus mesh with 608 triangles

Figure 6.19: Comparison with WALF

accuracy and robustness for FAH-WALF is that it requires a more compact stencil for local fitting which reduces the risk of overfitting.

Chapter 7

Conclusion and Further Directions

In this dissertation, we presented a method for generating a hierarchical unstructured meshes in parallel for efficient solution of PDE's using finite element methods and multigrid solvers. A multi-degree, multi-dimensional and multilevel framework is designed to generate the nested hierarchies from an initial mesh that can be used for a number of purposes from multi-level methods to generating large meshes. Two parallel communication algorithms are designed to aid in resolution of shared interface. We presented numerical results for computational efficiency of the refinement strategy in a parallel set up as well as the effect on mesh quality. We also demonstrated the applicability of the developed capability for multilevel and finite element methods as well as provide access to different point projection strategies that can effect the solution accuracy.

One of our primary contribution is that we developed a simple but general array-based half-facet mesh data structure, called Hierarchical AHF, for hierarchical meshes under adaptive mesh refinement. We described the algorithms and a prototype implementation in MATLAB for both refinement and derefinement for

2D triangular and 3D tetrahedral meshes. We demonstrate that Hierarchical AHF is efficient in terms of both storage and computational costs. Our framework could be easily integrated with finite element codes that support nonconformal meshes. The numerical results indicate the effectiveness of the adaptive procedures. In addition, our data structure is easily extended to support red-green refinement, so that it can also be used with finite element codes that require conformal meshes.

Hierarchical AHF stores all the information using arrays instead of pointers. Due to its array-based nature, it is well suited for parallel computations and is relatively easier to port onto GPUs. In addition, it facilitates easier interoperability with application codes. This tree hierarchy could be further utilized by multigrid or multilevel methods, which are often used as solvers of the arising linear systems for large scale simulations. We plan to explore these aspects in our future research.

In addition, we considered the problem of high-order surface reconstruction from surface meshes, which is important for meshing and geometric modeling. We introduced the *FAH-WALF* method, which extended the WALF method [61] in two ways. First, we introduced a Hermite-style least-squares approximation, which leverages both points and normals of the input mesh. Second, we described the high-order reconstruction of feature curves, as well as a blending strategy to ensure G^0 continuity of the reconstructed surface at sharp features, while preserving the order of accuracy. We proved the high-order of convergence of our proposed techniques. The experimental results verified the high-order convergence rates FAH-WALF and its robustness, especially for coarse input meshes.

By design, FAH-WALF is useful when an accurate, instead of "exact" geometry is needed, and when accessing the CAD software may be inconvenient. In particular, it is especially attractive for high-order finite elements, mesh refinement, mesh smoothing and mesh adaptivity, both in serial and parallel, especially for the solutions of PDEs. In such applications, it is desirable for the discretization errors to be balanced for those from the PDE discretizations and from the geometric representations. In this aspect, FAH-WALF is similar to WALF. However, an important advantage of the FAH-WALF methodology, compared to WALF, is that we can use high-order reconstruction, such as fifth- or sixth-order reconstructions, instead of only second or third-order reconstructions, on a relatively coarse mesh, while ensuring the accuracy and stability of the method. Therefore, with FAH-WALF, an application can start from a mesh that may be two to four times coarser in each direction while achieving the same accuracy of the reconstruction compared to using WALF. This leads to a reduction by a factor of 4 to 16 of the input mesh for a surface mesh, and a factor of 8 to 64 for a 3-D volume mesh. This advantages broadly expand the scope of applicability of high-order surface reconstruction to practical applications.

For geometric modeling applications, FAH-WALF methodology provides a valuable technique to complement the CAD techniques, such as NURBS and T-splines. Even though FAH-WALF enforces only G^0 continuity, the jumps in the normals and curvatures can be guaranteed to be high-order for smooth surfaces, which may suffice for many applications. In addition, our treatment of feature curves also makes the technique more convenient, and potentially also more robust, than traditional CAD techniques from the applications' point of view. However, FAH-WALF is by no means a replacement of traditional CAD techniques. In particular, if G^1 or G^2 continuity is needed, such as geometric modeling with just few control points, then the traditional G^1 or G^2 continuous CAD models, such as NURBS and T-splines may be advantageous. In addition, for high accuracy, the input points and normals must be accurate, preferably obtained from an analytic description of the surface, such as a CAD model. As a future research direction, we will investigate the noise-resistance of the proposed technique when the normals are imperfect, such as when they are estimated from the input mesh, similar to the techniques presented in [62, 119].

Bibliography

[1] Wikipedia: list of finite element software packages. `http://en.wikipedia.org/wiki/List_of_finite_element_software_packages`. Accessed: 2015-01-26.

[2] B. A Szabó. Mesh design for the p-version of the finite element method. *Computer Methods in Applied Mechanics and Engineering*, 55(1):181–197, 1986.

[3] M. Ainsworth and J. T. Oden. A posteriori error estimation in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 142(1):1–88, 1997.

[4] M. Ainsworth and J. T. Oden. *A posteriori error estimation in finite element analysis*, volume 37. John Wiley & Sons, 2011.

[5] M. Ainsworth and B. Senior. Aspects of an adaptive hp-finite element method: Adaptive strategy, conforming approximation and efficient solvers. *Computer Methods in Applied Mechanics and Engineering*, 150(1):65–87, 1997.

[6] M. Ainsworth and B. Senior. An adaptive refinement strategy for hp-finite element computations. *Appl. Numer. Math.*, 26(1):165–178, 1998.

[7] T. Alumbaugh and X. Jiao. Compact array-based mesh data structures. In *Proceedings of 14th International Meshing Roundtable*, pages 485–504, 2005.

[8] R. B. Andrew, A. H. Sherman, and A. Weiser. Some refinement algorithms and data structures for regular local mesh refinement, 1983.

[9] D. N. Arnold, A. Mukherjee, and L. Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2000.

[10] I. Babuska and W. Rheinboldt. Error estimates in adaptive finite element computations. *SIAM J. Numer. Anal.*, 15:736–754, 1978.

[11] I. Babuska and W. Rheinboldt. A posteriori error estimates for the finite element method. *Internat. J. Numer. Methods Engrg.*, 12:1597–1615, 1978.

[12] I. Babuska and W. C. Rheinboldt. A posteriori error analysis of finite element solutions for one-dimensional problems. *SIAM Journal on Numerical Analysis*, 18(3):565–589, 1981.

[13] I. Babuška, T. Strouboulis, and K. Copps. hp optimization of finite element approximations: Analysis of the optimal mesh sequences in one dimension. *Computer methods in applied mechanics and engineering*, 150(1):89–108, 1997.

[14] I. Babuška, B. A. Szabo, and I. N. Katz. The p-version of the finite element method. *SIAM J. Numer. Anal.*, 18(3):515–545, 1981.

[15] W. Bangerth, R. Hartmann, and G. Kanschat. deal. ii-a general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):24, 2007.

[16] W. Bangerth and O. Kayser-Herold. Data structures and requirements for hp finite element software. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):4, 2009.

[17] R. E. Bank. *PLTMG, a Software Package for Solving Elliptic Partial Differential Equations: Users' Guide 8.0*, volume 5. Siam, 1998.

[18] R. E. Bank, T. F. Dupont, and H. Yserentant. The hierarchical basis multigrid method. *Numerische Mathematik*, 52(4):427–458, 1988.

[19] R. E. Bank and A. Weiser. Some a posteriori error estimators for elliptic partial differential equations. *Mathematics of Computation*, 44(170):283–301, 1985.

[20] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering*, 3(3):181–191, 1991.

[21] E. Bänsch, P. Morin, and R. H. Nochetto. An adaptive uzawa fem for the stokes problem: Convergence without the inf-sup condition. *SIAM Journal on Numerical Analysis*, 40(4):1207–1229, 2002.

[22] P. Binev, W. Dahmen, and R. DeVore. Adaptive finite element methods with convergence rates. *Numerische Mathematik*, 97(2):219–268, 2004.

[23] B. S. Bischoff, M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, and R. Aachen. OpenMesh – a generic and efficient polygon mesh data structure. In *In OpenSG Symposium*, 2002.

[24] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Mathematics of Computation*, 55(191):1–22, 1990.

[25] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[26] C. Canuto, M. Y. Hussaini, A. M. Quarteroni, A. Thomas Jr, et al. *Spectral methods in fluid dynamics*. Springer Science & Business Media, Berlin Heidelberg, 2012.

[27] C. Carstensen and R. Hoppe. Error reduction and convergence for an adaptive mixed finite element method. *Mathematics of computation*, 75(255):1033–1042, 2006.

[28] C. Carstensen and R. H. Hoppe. Convergence analysis of an adaptive nonconforming finite element method. *Numerische Mathematik*, 103(2):251–266, 2006.

[29] J. M. Cascon, C. Kreuzer, R. H. Nochetto, and K. G. Siebert. Quasi-optimal convergence rate for an adaptive finite element method. *SIAM Journal on Numerical Analysis*, 46(5):2524–2550, 2008.

[30] L. Chen, M. Holst, and J. Xu. Convergence and optimality of adaptive mixed finite element methods. *Mathematics of Computation*, 78(265):35–53, 2009.

[31] L. Chen, R. H. Nochetto, and J. Xu. Optimal multilevel methods for graded bisection grids. *Numerische Mathematik*, 120(1):1–34, 2012.

[32] K. C. Chitale, O. Sahni, M. S. Shephard, S. Tendulkar, and K. E. Jansen. Anisotropic adaptation for transonic flows with turbulent boundary layers. *AIAA Journal*, 53(2):367–378, 2014.

[33] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *The Development of Discontinuous Galerkin Methods*. Springer, Berlin Heidelberg, 2000.

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

[35] L. Demkowicz. *Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume 1 One and Two Dimensional Elliptic and Maxwell problems*. CRC Press, 2006.

[36] L. Demkowicz, J. Oden, and T. Strouboulis. An adaptive p-version finite element method for transient flow problems with moving boundaries. In *Finite elements in fluids*, volume 1, pages 291–305, 1985.

[37] L. Demkowicz, W. Rachowicz, and P. Devloo. A fully automatic hp-adaptivity. *SIAM J. Sci. Comput.*, 17(1):117–142, 2002.

[38] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodriguez-Ferran. Arbitrary Lagrangian-Eulerian methods. In E. Stein, R. de Borst, and T. J. Hughes, editors, *Encyclopedia of Computational Mechanics*, chapter 14. Wiley, 2004.

[39] W. Dörfler. A convergent adaptive algorithm for poisson's equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, 1996.

[40] V. Dyedov, N. Ray, D. Einstein, X. Jiao, and T. Tautges. Ahf: Array-based half-facet data structure for mixed-dimensional and non-manifold meshes. In J. Sarrate and M. Staten, editors, *Proceedings of the 22nd International Meshing Roundtable*, pages 445–464. Springer International Publishing, 2014.

[41] H. C. Edwards, A. B. Williams, G. D. Sjaardema, D. G. Baur, and W. K. Cochran. Sierra toolkit computational mesh conceptual model. *Sandia National Laboratories SAND Series, SAND2010-1192*, 2010.

[42] I. Ergatoudis, B. Irons, and O. Zienkiewicz. Curved, isoparametric, "quadrilateral" elements for finite element analysis. *Int. J. Solids Struct.*, 4(1):31–42, 1968.

[43] K. Eriksson and C. Johnson. Adaptive finite element methods for parabolic problems i: A linear model problem. *SIAM Journal on Numerical Analysis*, 28(1):43–77, 1991.

[44] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30:1167–1202, 2000. Special Issue on Discrete Algorithm Engineering.

[45] G. Farin. Curves and surfaces for computer aided geometric design (3rd ed.). Academic Press, San Diego, 1993.

[46] J. E. Flaherty. Csci, math 6860 finite element analysis. Lecture Notes: 2000.

[47] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Comput. Graph. (TOG)*, 24(3), 2005.

[48] P. J. Frey and P. L. George. *Mesh Generation: Application to finite elements*. Hermes, 2000.

[49] R. V. Garimella. MSTK – a flexible infrastructure library for developing mesh based applications. In *Proceedings of 13th International Meshing Roundtable*, pages 213–220, 2004.

[50] J. Goldfeather and V. Interrante. A novel cubic-order algorithm for approximating principal direction vectors. *ACM Trans. Comput. Graph. (TOG)*, 23(1):45–63, 2004.

[51] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins, 3rd edition, 1996.

[52] W. Gui and I. Babuska. The h, p and hp versions of the finite element method in 1 dimension. i-iii. *Numerische Mathematik*, 49(6):577–683, 1986.

[53] B. Guo and I. Babuška. The hp version of the finite element method. *Comput. Mech.*, 1(1):21–41, 1986.

[54] P. Houston and E. Süli. A note on the design of hp-adaptive finite element methods for elliptic partial differential equations. *Computer methods in applied mechanics and engineering*, 194(2):229–243, 2005.

[55] J. Hu and J. Xu. Convergence of adaptive conforming and nonconforming finite element methods for the perturbed stokes equation. Technical report, Research Report. School of Mathematical Sciences and Institute of Mathematics, Peking University, 2007.

[56] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Meth. Appl. Mech. Engrg.*, 194(39):4135–4195, 2005.

[57] D. A. Ibanez, E. S. Seol, C. W. Smith, and M. S. Shephard. Pumi: Parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):17, 2016.

[58] R. Jain and T. J. Tautges. Generating unstructured nuclear reactor core meshes in parallel. *Procedia Engineering*, 82(0):351 – 363, 2014. 23rd International Meshing Roundtable (IMR23).

[59] X. Jiao, A. Colombi, X. Ni, and J. Hart. Anisotropic mesh adaptation for evolving triangulated surfaces. *Engrg. Comput.*, 26:363–376, 2010.

[60] X. Jiao and D. Wang. Reconstructing High-Order Surfaces for Meshing. In S. Shontz, editor, *Proceedings of the 19th International Meshing Roundtable*, pages 143–160. Springer Berlin Heidelberg, 2010.

[61] X. Jiao and D. Wang. Reconstructing high-order surfaces for meshing. *Engineering with Computers*, 28:361–373, 2012.

[62] X. Jiao and H. Zha. Consistent computation of first- and second-order differential quantities for surface meshes. In *ACM Solid and Physical Modeling Symposium*, 2008.

[63] M. T. Jones and P. E. Plassmann. Adaptive refinement of unstructured finite-element meshes. *Finite Elements in Analysis and Design*, 25(1):41–60, 1997.

[64] D. Kelly. The self-equilibration of residuals and complementary a posteriori error estimates in the finite element method. *International Journal for Numerical Methods in Engineering*, 20(8):1491–1506, 1984.

[65] D. Kelly, D. S. Gago, O. Zienkiewicz, I. Babuska, et al. A posteriori error analysis and adaptive processes in the finite element method: Part i-error analysis. *International Journal for Numerical Methods in Engineering*, 19(11):1593–1619, 1983.

[66] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theo. Appl.*, 13:65–90, 1999.

[67] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22(3-4):237–254, 2006.

[68] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: A c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22:237–254, 2006.

[69] Y. Kondratyuk. Adaptive finite element algorithms for the stokes problem: Convergence rates and optimal computational complexity. *Preprint*, 1346, 2006.

[70] Y. Kondratyuk and R. Stevenson. An optimal adaptive finite element method for the stokes problem. *SIAM Journal on Numerical Analysis*, 46(2):747–775, 2008.

[71] I. Kossaczkỳ. A recursive approach to local mesh refinement in two and three dimensions. *Journal of Computational and Applied Mathematics*, 55(3):275–288, 1994.

[72] M. Kremer, D. Bommes, and L. Kobbelt. OpenVolumeMesh – a versatile index based data structure for 3D polytopal complexes. *Proceedings of 21st International Meshing Roundtable*, pages 531–548, 2012.

[73] P. Kus, P. Solin, and D. Andrs. Arbitrary-level hanging nodes for adaptive hp-fem approximations in 3D. *Journal of Computational and Applied Mathematics*, 270:121–133, 2014.

[74] P. Ladeveze and D. Leguillon. Error estimate procedure in the finite element method and applications. *SIAM Journal on Numerical Analysis*, 20(3):485–509, 1983.

[75] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting: An Introduction*. Academic Press, 1986.

[76] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67:1517–1531, 1998.

[77] X. Li, M. S. Shephard, and M. W. Beall. Accounting for curved domains in mesh adaptation. *Int. J. Numer. Meth. Engrg.*, 58(2):247–276, 2003.

[78] X. Li, M. S. Shephard, and M. W. Beall. 3d anisotropic mesh adaptation by mesh modification. *Computer methods in applied mechanics and engineering*, 194(48):4915–4950, 2005.

[79] A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing*, 16(6):1269–1291, 1995.

[80] X. Luo, M. S. Shephard, and J.-F. Remacle. The influence of geometric approximation on the accuracy of high order methods. *Rensselaer SCOREC report*, 1, 2001.

[81] C. Mavriplis. Adaptive mesh strategies for the spectral element method. *Computer methods in applied mechanics and engineering*, 116(1):77–86, 1994.

[82] K. Mekchay and R. H. Nochetto. Convergence of adaptive finite element methods for general second order linear elliptic pdes. *SIAM Journal on Numerical Analysis*, 43(5):1803–1827, 2005.

[83] W. Mitchell. Adaptive mesh refinement benchmark problems. http://math.nist.gov/amr-benchmark/index.html, 2013. Accessed:2015-04-24.

[84] W. F. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. *ACM Transactions on Mathematical Software (TOMS)*, 15(4):326–347, 1989.

[85] P. Morin, R. H. Nochetto, and K. G. Siebert. Data oscillation and convergence of adaptive fem. *SIAM Journal on Numerical Analysis*, 38(2):466–488, 2000.

[86] P. Morin, R. H. Nochetto, and K. G. Siebert. Convergence of adaptive finite element methods. *SIAM review*, 44(4):631–658, 2002.

[87] P. Morin, K. G. Siebert, and A. Veeser. A basic convergence result for conforming adaptive finite elements. *Mathematical Models and Methods in Applied Sciences*, 18(05):707–737, 2008.

[88] R. H. Nochetto, K. G. Siebert, and A. Veeser. Theory of adaptive finite element methods: an introduction. In *Multiscale, nonlinear and adaptive approximation*, pages 409–542. Springer, 2009.

[89] J. T. Oden, W. Wu, and M. Ainsworth. Three-step hp adaptive strategy for the incompressible navier-stokes equations. In *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, pages 347–366. Springer, 1995.

[90] P.-O. Persson and J. Peraire. Curved mesh generation and mesh refinement using Lagrangian solid mechanics. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting and Exhibit*, volume 204, 2009.

[91] D. Poirier, S. R. Allmaras, D. R. McCarthy, M. F. Smith, and F. Y. Enomoto. The CGNS system, 1998. AIAA Paper 98-3007.

[92] W. Rachowicz, J. T. Oden, and L. Demkowicz. Toward a universal hp adaptive finite element strategy part 3. design of hp meshes. *Computer Methods in Applied Mechanics and Engineering*, 77(1):181–212, 1989.

[93] N. Ray. *High-Order Surface Reconstruction and its Applications to Surface Integrals and Surface Remeshing*. PhD thesis, Stony Brook University, 2013.

[94] N. Ray, I. Grindeanu, X. Zhao, V. Mahadevan, and X. Jiao. Array-based, parallel hierarchical mesh refinement algorithms for unstructured meshes. *Computer-Aided Design*, 2016.

[95] M.-C. Rivara. Design and data structure of fully adaptive, multigrid, finite-element software. *ACM Transactions on Mathematical Software (TOMS)*, 10(3):242–264, 1984.

[96] M.-C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21(3):604–613, 1984.

[97] M.-C. Rivara and C. Levin. A 3-D refinement algorithm suitable for adaptive and multi-grid techniques. *Communications in Applied Numerical Methods*, 8(5):281–290, 1992.

[98] A. Schmidt and K. G. Siebert. A posteriori estimators for the h–p version of the finite element method in 1d. *Applied numerical mathematics*, 35(1):43–66, 2000.

[99] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. *ACM Trans. Graph.*, 22(3):477–484, 2003.

[100] K. Segeth. A review of some a posteriori error estimates for adaptive finite element methods. *Mathematics and Computers in Simulation*, 80(8):1589–1600, 2010.

[101] E. S. Seol. *FMDB: Flexible Distributed Mesh Database For Parallel Automated Adaptive Analysis*. PhD thesis, Rensselaer Polytechnic Institute, 2005.

[102] K. G. Siebert. A convergence proof for adaptive finite elements without lower bound. *IMA journal of numerical analysis*, 31(3):947–970, 2011.

[103] D. Sieger and M. Botsch. Design, implementation and evaluation of the surface mesh data structure. In *In Proceedings of the 20th International Meshing Roundtable*, 2011.

[104] P. Šolín, J. Červený, and I. Doležel. Arbitrary-level hanging nodes and automatic adaptivity in the hp-FEM. *Math. Comput. Simulat.*, 77(1):117–132, 2008.

[105] P. Šolín et al. Hermes - higher-order modular finite element system. `http://hpfem.org/`.

[106] R. Stevenson. Optimality of a standard adaptive finite element method. *Foundations of Computational Mathematics*, 7(2):245–269, 2007.

[107] T. Tautges, R. Meyers, and K. Merkley. MOAB: A mesh-oriented database. Technical report, Sandia National Laboratories, 2004.

[108] T. J. Tautges. Canonical numbering systems for finite-element codes. *International Journal for Numerical Methods in Biomedical Engineering*, 26(12):1559–1572, 2010.

[109] T. J. Tautges, J. Kraftcheck, J. Porter, A. Caceres, I. Grindeanu, D. Karpeev, R. Jain, H.-J. Kim, S. Cai, S. Jackson, J. Hu, B. Smith, C. Verma, S. Slattery, and P. Wilson. MeshKit: a Open-Source library for mesh generation. In *Proceedings, SIAM Conference on Computational Science & Engineering*, Reno, NV, Mar. 2011. SIAM.

[110] T. J. Tautges, J. A. Kraftcheck, N. Bertram, V. Sachdeva, and J. Magerlein. Mesh interface resolution and ghost exchange in a parallel mesh representation. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 1670–1679, Washington, DC, USA, 2012. IEEE Computer Society.

[111] The CGNS Steering Sub-committee. *The CFD General Notation System Standard Interface Data Structures*. AIAA, 2002.

[112] K. Tim and T. Preusser. Stability of the 8-tetrahedra shortest-interior-edge partitioning method. *Numerische Mathematik*, 109:435–457, 2008.

[113] L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[114] A. van der Sluis. Condition numbers and equilibration of matrices. *Numer. Math.*, 14:14–23, 1969.

[115] R. Verfürth. A review of a posteriori error estimation techniques for elasticity problems. *Computer Methods in Applied Mechanics and Engineering*, 176(1):419–440, 1999.

[116] P. Vincent and A. Jameson. Facilitating the adoption of unstructured high-order methods amongst a wider community of fluid dynamicists. *Math. Model. Nat. Phenom.*, 6(3):97–140, 2011.

[117] A. Vlachos, J. Peters, C. Boyd, and J. L. Mitchell. Curved PN triangles. In *Proc.of the 2001 Symposium on Interactive 3D graphics*, pages 159–166, 2001.

[118] D. Walton. A triangular g1 patch from boundary curves. *Comput. Aid. Des.*, 28(2):113–123, 1996.

[119] D. Wang, B. L. Clark, and X. Jiao. An analysis and comparison of parameterization-based computation of differential quantities for discrete surfaces. *Comput. Aid. Geom. Des.*, 26:510–527, 2009.

[120] Z. Wang. High-order methods for the Euler and Navier–Stokes equations on unstructured grids. *Prog. Aerosp. Sc.*, 43(1):1–41, 2007.

[121] H. Wu and Z. Chen. Uniform convergence of multigrid v-cycle on adaptively refined finite element meshes for second order elliptic problems. *Science in China Series A: Mathematics*, 49(10):1405–1429, 2006.

[122] X. Zhao, R. Conley, N. Ray, V. S. Mahadevan, and X. Jiao. Conformal and non-conformal adaptive mesh refinement with hierarchical array-based half-facet data structures. *Procedia Engineering*, 124:304–316, 2015.

[123] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for practical engineerng analysis. *International Journal for Numerical Methods in Engineering*, 24(2):337–357, 1987.