Massively Parallel Editing and Post-Processing of Unstructured Tetrahedral Meshes for Virtual Prototyping

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Daniel Ströter aus Frankfurt a. M.
Tag der Einreichung: 30.07.2024, Tag der Prüfung: 10.09.2024

1. Gutachten: Prof. Dr. techn. Dr.-Ing. eh. Dieter W. Fellner
2. Gutachten: Hon. Prof. Dr.-Ing. André Stork
3. Gutachten: Prof. Dr.-Ing. Marc Alexa
Darmstädter Dissertation D17

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

GRIS

Interactive Graphics
Systems Group

Massively Parallel Editing and Post-Processing of Unstructured Tetrahedral Meshes for Virtual Prototyping

Accepted doctoral thesis by Daniel Ströter

Date of submission: 30.07.2024
Date of thesis defense: 10.09.2024

Darmstädter Dissertation D17

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 30.07.2024

D. Ströter

Zusammenfassung

In der industriellen Produktentwicklung wird heutzutage häufig auf virtuelles Prototyping zurückgegriffen, um Entwicklungszeit und Materialkosten zu sparen. Obwohl virtuelles Prototyping durch computergestütztes Design und computergestützter Simulation signifikante Einsparungen von Entwicklungszeit und Materialkosten ermöglicht, beinhaltet es mehrere aufwendige und komplexe Zwischenschritte. Üblicherweise optimieren professionelle Teams während der Produktentwicklung einen Prototypen auf eine Vielzahl an Zielsetzungen wie z. B. sparsamer Gebrauch von Ressourcen und Stabilität unter Einwirkung bestimmter physischer Belastungen. Dies erfordert in der Regel ein mehrmaliges Durchführen von virtuellem Prototyping. Deswegen sind Methoden für das Beschleunigen und das Verkürzen des Ablaufs von virtuellem Prototyping ein wichtiges Forschungsziel.

Diese Dissertation beschäftigt sich mit massiv parallelen Algorithmen, welche die faszinierende gebündelte Rechenleistung von modernen Grafikkarten ausnutzen, um virtuelles Prototyping zu beschleunigen und dessen Ablauf um einige aufwendige Zwischenschritte zu verkürzen. Das virtuelle Prototyping involviert häufig die Erstellung, Optimierung und Anpassung hochauflösender volumetrischer Netze zum Zweck einer numerischen Simulation, weswegen die vorliegende Dissertation ihren Fokus auf die effiziente Verarbeitung volumetrischer Netze legt. Eine zur numerischen Simulation häufig eingesetzte Art von volumetrischen Netzen sind unstrukturierte Tetraedernetze, weil sie eine robuste Netzgenerierung bieten und die Geometrie eines Tetraeders eine feine Diskretisierung zur Approximation einer Oberfläche ermöglicht. Aufgrund dieser Vorteile konzentriert sich die vorliegende Dissertation auf unstrukturierte Tetraedernetze.

Für das virtuelle Prototyping gilt es allerdings eine Vielzahl an Kriterien für eine erfolgreiche sowie genaue numerische Simulation zu beachten. Besonders maßgebliche Kriterien bestehen in der Auflösung des Netzes und der Qualität der geometrischen Form der Tetraederelemente. Daher finden Optimierung sowie lokale Neuvernetzung des Tetraedernetzes häufig Anwendung im virtuellem Prototyping. Um eine Beschleunigung der entsprechenden Verfahren zu erwirken, beschäftigt sich die vorliegende Dissertation mit Strategien zur massiven Parallelisierung grundlegender Operationen, welche Bestandteil von Verfahren zur Netzoptimierung und lokaler Neuvernetzung sind. Darüber hinaus ist die Robustheit der parallelen Algorithmen ein Forschungsschwerpunkt, weil Algorithmen nur dann erfolgreich in dem virtuellem Prototyping eingesetzt werden können, wenn die resultierenden Netze die Kriterien der nachfolgenden numerischen Simulation erfüllen.

Einer der zeit intensivsten Aufwendungen beim virtuellen Prototyping besteht darin, dass die Anpassung eines Prototypentwurfes eine Änderung der mittels computergestütztem Designs erzeugten Freiformflächen und eine erneute Diskretisierung in ein volumetrisches Netz erfordert. Folglich können virtuelle Prototyping-Prozesse durch Methoden zur Vermeidung der wiederholten Anpassung der Freiformflächen des Prototyps und der anschließenden Netzgenerierung erheblich verkürzt werden. Um die Möglichkeiten zur Verkürzung virtueller Prototyping-Iterationen zu erweitern, werden in dieser Dissertation interaktive Methoden zur direkten Bearbeitung der Tetraedernetze ohne direkte Anpassung der Freiformflächen untersucht. Die schnelle Rechenleistung von massiv-parallelen Grafikkarten hat das Potenzial die Anpassung von hochauflösenden Tetraedernetzen auf interaktive Weise zu realisieren.

Jeder Durchlauf eines virtuellen Prototyping-Prozesses benötigt eine Methode, um die Simulationsergebnisse visuell zu analysieren. Bei der visuellen Analyse wendet das Produktionsteam in der Regel Post-

Processing an, welches das Tetraedernetz und die zugehörigen Simulationsergebnisse visualisiert. Für das Post-Processing volumetrischer Netze ist es oft entscheidend auch die inneren Strukturen des Volumens zu visualisieren. Eine häufig angewendete Methode zur Visualisierung volumetrischer Netze ist das sogenannte direkte Rendering von Volumen. Das direkte Rendering von hochauflösenden Netzen benötigt feingranulare räumliche Datenstrukturen, welche den Raum fein aufteilen, damit die räumliche Suche nach Elementen des Netzes effektiv beschleunigt wird. Dementsprechend kann eine räumliche Datenstruktur einen beträchtlichen Anteil des auf einer Grafikkarte verfügbaren Speichers belegen. Daher wird in dieser Dissertation speicher-effizientes Post-Processing von unstrukturierten Tetraedernetzen angestrebt.

Die vorliegende Dissertation beinhaltet eine Vielfalt an Methoden für schnelleres virtuelles Prototyping. Sie präsentiert Verfahren zur Berechnung voneinander unabhängiger Teilnetze eines volumetrischen Netzes, welche das konfliktfreie Ausführen von lokaler Neuvernetzung der Teilnetze auf Grafikkarten ermöglicht. Des Weiteren beinhaltet die vorliegende Dissertation eine robuste sowie massiv parallele Methode zur Optimierung der Knotenpositionen basierend auf dem Verfahren des steilsten Abstiegs. Durch das Anwenden dieser Methoden kann die Optimierung und die Neuvernetzung von unstrukturierten Tetraedernetzen um ein bis zwei Größenordnungen beschleunigt werden. Um den Aufwand virtueller Prototyping-Prozesse zu reduzieren, präsentiert die vorliegende Dissertation Methoden, welche es einem Nutzer auf Basis von semantischen Flächengruppen oder Kontroll-Käfigen ermöglichen ein unstrukturiertes Tetraedernetz zu editieren. Aufgrund von geschickter Ausnutzung massive paralleler Grafikkarten, kann ein interaktives Editieren der Netze realisiert werden. Bei der Gestaltung dieser Methoden zur Netzeditierung wurden Maßnahmen ergriffen, um eine Tetraedernetze zu erzeugen, welche die Kriterien von numerischen Simulationen erfüllen. Darüber hinaus stellt die vorliegende Dissertation eine speicher-effiziente räumliche Datenstruktur vor, welche es mittels parametrisierter Konstruktion dem Nutzer ermöglicht den Speicherverbrauch zu steuern. Außerdem beschreibt die vorliegende Dissertation eine Methode zur Vereinfachung von unstrukturierten Tetradernetzen, welche die Anzahl der Tetraeder in einem Netz um drei Viertel reduzieren kann, ohne dass die Wahrheitsgetreue des Post-Processings signifikant beeinträchtigt wird.

Abstract

Today, many tasks in industrial product development rely on virtual prototyping to reduce development time and resource costs. Although virtual prototyping provides significant simplification of product development through the use of computer-aided design and computer-aided engineering, it remains a laborious and time consuming process that involves a number of complex steps. Typically, product development teams optimize their prototypes for many design goals, e.g., economical use of material and stability under forces, which demands many iterations of virtual prototyping. Therefore, methods for the acceleration and shortening of virtual prototyping processes are important technological advances.

This thesis presents massively parallel algorithms that exploit the impressive aggregated processing power of present-day general purpose graphics processing units to accelerate and shorten virtual prototyping. As virtual prototyping oftentimes involves the generation, optimization and adaptation of high-resolution volumetric meshes for numerical simulation, this thesis focuses on efficient processing of volumetric meshes. Unstructured tetrahedral meshes are a commonly used type of volumetric meshes, because they provide robust meshing and tetrahedra allow for good discretized approximation of surface features. Therefore, this thesis narrows its scope to unstructured tetrahedral meshes.

In virtual prototyping, a number of properties of the tetrahedral mesh concerns the success of a numerical simulation. Important properties are the resolution of the mesh and the shape quality of the tetrahedral elements. Consequently, the optimization and re-meshing of tetrahedral meshes are common tasks in virtual prototyping. This thesis investigates parallelization strategies for tetrahedral mesh editing operations that are fundamental for mesh optimization and re-meshing. In addition, the robustness of the presented methods is a research objective, because successful acceleration of virtual prototyping is only achieved, if the presented methods function properly and produce meshes that are suitable for downstream numerical simulation.

One of the primary overheads in virtual prototyping is that new prototype designs demand new discretization of boundary representations to a volumetric mesh. For this reason, virtual prototyping processes can be significantly shortened by methods for avoiding the repeated modeling of the prototype's boundary representations and subsequent mesh generation. In order to extend the facilities of shorter virtual prototyping iterations, this thesis explores user-interactive methods for directly editing the tetrahedral mesh without adjusting the boundary representations in a computer-aided design environment. The fast run time performance of massively parallel processing provides promising potential to achieve editing of high-resolution meshes at interactive rates.

Every virtual prototyping process requires a method that allows the development team the visual analysis of the simulation results. In the visual analysis step, the development team typically applies post-processing to the mesh and its annotated simulation results. Since accurate numerical simulations might require high-resolution meshes, the use of graphics processing units is common for post-processing. For post-processing volumetric meshes, it is important to visualize the inner structures of the mesh to enable a complete analysis of the prototype. A common method for post-processing volumetric meshes is direct volume rendering. The direct volume rendering of high-resolution meshes requires comprehensive acceleration data structures for fast spatial search of mesh elements, which can lead to large memory consumption. Therefore, this thesis investigates memory-efficient post-processing of unstructured tetrahedral meshes for better management of the available memory capacity.

This thesis presents a multitude of contributions for faster virtual prototyping. It presents conflict detection methods to determine dense sub-meshes for massively parallel edge/face flips and re-meshing. In addition, this thesis contributes a robust massively parallel method to relocate mesh vertices for first-order optimization methods. With the use of the presented methods, optimization and re-meshing of unstructured tetrahedral meshes can be accelerated by one or two orders of magnitude. For shortening virtual prototyping, this thesis presents user-interactive editing by user-selected face groups as well as deformation control to edit unstructured tetrahedral meshes. Due to massively parallel processing, these methods enable interactive mesh editing. The mesh editing includes measures for producing tetrahedral meshes of sufficient quality for downstream numerical simulations. For post-processing of unstructured tetrahedral meshes, this thesis presents a memory-efficient spatial data structure along with a method to coarsen meshes for direct volume rendering. The spatial data structure enables control over memory consumption by a tuning parameter. The coarsening can reduce high-resolution tetrahedral meshes to a quarter of the initial size while well-preserving most visual features.

Acknowledgment

Many people have supported me throughout this thesis project. Without the help of these people, I would have had neither the opportunity nor the courage to write this thesis. Therefore, I dedicated this section to these people to express how thankful I am for their support.

First of all, I want to thank my family members for their continuous encouragement. I am deeply grateful to my mother Birgit for her advice as a retired math teacher and her sensitive as well as empathetic support. Throughout my lifetime, she always believed in me and knew how to build me back up, whenever I faced setbacks. When people doubted me because of my hearing condition, she always encouraged me to believe in my abilities. She taught me to never give up, to be diligent in life, and that honesty is the best policy. Cherishing these values has enabled me to develop my personality and to succeed in life. So, I am very thankful for all my mother has done for me.

I am thankful to my father Berthold for his advice as a passionate mathematician and for always encouraging me to follow my curiosity. As a person with interest in science, he was always eager to discuss the topics that I was concerned with. He always pointed out the importance of an accurate argumentation and that it is worthwhile to give my best to achieve good results. He advocated for trying to obtain "intellectually complete" results, which is an adjective for well-thought-out and comprehensible results. Having high work-ethics and the will to give it my all has enabled me to stay focused and motivated throughout my PhD journey.

I am thankful to my elder sister Laura, as she is an inspiring role model to look up to and at the same time an empathetic sister. When she has the time, she is always open for arranging a family meeting, where the family comes together to exchange their latest experiences. Family is an important part in a human's lifetime. I am thankful that she is always very caring for the family.

Subsequent to my family, I want to thank several people that I have worked with at the Graphics Interactive Systems Group (GRIS) and Fraunhofer IGD. I am thankful to Prof. D.W. Fellner for giving me the opportunity to do scientific research and teaching students in computer graphics. He entrusted me with a lot of responsibility, when he offered me the student supervision of his computer graphics courses. The teaching experience has helped me to refine my didactic abilities and grow as a person. In addition, he granted me a lot of freedom in my research, as he allowed me to follow my own ideas while always being available for an enlightening discussion. Trying my best to provide good research results has enabled me to improve my technical abilities as a computer scientist.

I am thankful to Prof. André Stork for his supervision of my thesis project. After three wonderful years of work experience at the Interactive Engineering Technologies (IET) department of Fraunhofer IGD, he advised to pursue my research goals at GRIS. As the head of IET, he enabled me to do collaborative research projects with the much valued colleagues from IET. Throughout my research I felt not just like a collaborator with the IET, but rather like a member of the IET-family. The continuous engagement with IET helped me to deepen my knowledge in the fields of my study. The continuous and critical feedback of Prof. André Stork has facilitated me to improve my abilities in scientific writing, critical thinking, and scientific research.

For his continuous advice, I am thankful to Dr. Daniel Weber. He always had an open door for a discussion about which topics I should address in my research. His advice in scientific research and his knowledge about the engineering industry helped me, when I was trying to navigate through the unbounded landscape of the research domain of geometry processing for virtual prototyping. I also want to thank Dr. Daniel Weber for the continuation of the collaboration with IET after he took up the position as head of IET.

I am thankful to Dr. Johannes Mueller-Roemer for teaching me all the skills of scientific writing and research. After the supervision of my Master's thesis, he has never stopped his tireless feedback, which has enabled me to refine my scientific writing style and my abilities to give a presentation. He was always open for a discussion about my research topics and pinpointed at important technical details and related research work.

Finally, I want to thank all of my colleagues at GRIS and IGD including the secretaries for being friendly, funny, and trustworthy. Working with the above people has made my PhD journey an experience that I enjoyed and now like to remember.

Contents

1. Introduction **1**

1.1. Motivation 1

1.2. Approach 4

1.3. Research Questions 4

1.4. Publications and Contributions 5

1.5. Outline 8

2. Preliminaries and Notation **9**

2.1. Unstructured Triangulations 9

2.1.1. Simplicial Meshes 9

2.1.2. Delaunay Triangulation 11

2.1.3. Harmonic Triangulation 13

2.2. APIs, Algorithms and Data Structures for Massively Parallel Processing 15

2.2.1. Massively Parallel GPU Computation and CUDA 15

2.2.2. Common Algorithms for GPU Programming 16

2.2.3. TCSR Simplicial Mesh Data Structure 19

2.3. Manipulating Geometry with Cage-based Deformation 20

2.3.1. Matrix Notation for Cage-based Deformation 21

2.3.2. Generalized Barycentric Coordinates 21

2.3.3. Linear Blend Skinning 23

2.4. Direct Volume Rendering of Unstructured Grids 24

3. Related Work **27**

3.1. The Two Scopes of Mesh Editing 27

3.2. Mesh Optimization and Re-meshing of Unstructured Tetrahedral Meshes 28

3.2.1. From Laplacian Smoothing to Optimizing Element Quality Metrics 28

3.2.2. Distortion Energies for Mesh Optimization 29

3.2.3. Edge Collapse in Tetrahedral Meshes 30

3.2.4. Applications of Tetrahedral Mesh Coarsening 30

3.2.5. Parallel Vertex Relocation 31

3.2.6. Parallel Re-meshing 31

3.2.7. Boundary Treatment in Mesh Optimization 33

3.3. High-level Volumetric Mesh Editing 33

3.3.1. Volumetric Mesh Editing on the Basis of Semantic Features 33

3.3.2. Volumetric Mesh Editing by Shape Deformation 34

3.4. Interactive Mesh Deformation for Model Manipulation 35

3.4.1. Free-form Deformation using Lattices 35

3.4.2. Skeletal-based Deformation 35

3.4.3. Linear Subspaces 36

3.4.4. Radial-Basis-Function-based Deformation 36

3.5. Generation of Cages for Deformation Control . . . 36
3.5.1. Cage Quality . . . 37
3.5.2. Offset Surface Simplification Methods . . . 38
3.5.3. Voxelization-based Methods . . . 38
3.5.4. Template-based Methods . . . 39
3.5.5. Interactive Cage Generation Methods . . . 39
3.5.6. Embedding the Cage . . . 40
3.6. Construction of Cage Coordinates for Deformation Control . . . 41
3.6.1. Barycentric Coordinates with Explicit Formula . . . 41
3.6.2. Energy Minimization-based Barycentric Coordinates . . . 44
3.6.3. Probability-based Coordinates . . . 46
3.6.4. Coordinates with Normal Control . . . 49
3.7. Isogeometric Analysis . . . 53
3.8. Spatial Datastructures for GPU-parallel Construction and Rendering . . . 54
3.8.1. Linear Spatial Datastructures . . . 54
3.8.2. Massively Parallel Traversal . . . 55
3.8.3. Hardware Accelerated Spatial Data Structure . . . 55
3.9. Direct Volume Rendering of Unstructured Meshes . . . 56

4. Fast Harmonic Mesh Optimization 59
4.1. Harmonic Gradient . . . 60
4.2. Boundary Features Extraction for Gradient Descent . . . 62
4.3. Vertex Relocation . . . 63
4.3.1. Relocating Interior Vertices . . . 63
4.3.2. Directional Derivatives for Boundary Treatment . . . 66
4.3.3. Preserving the Boundary . . . 68
4.3.4. Approximate Boundary Preservation to Relocate Every Vertex . . . 69
4.4. Parallel Harmonic Flipping . . . 71
4.5. Combined Vertex Relocation and Flipping . . . 74
4.6. Evaluation of Harmonic Mesh Optimization . . . 74
4.6.1. Implementation of the Original Harmonic Mesh Optimization Algorithm . . . 75
4.6.2. Run Time Performance of Parallel Flipping . . . 75
4.6.3. Robustness . . . 75
4.6.4. Element Quality and Convergence . . . 76
4.6.5. Run Time Performance of Harmonic Mesh Optimization . . . 78
4.7. Summary . . . 80

5. Massively Parallel Collapsing of Edges of Unstructured Tetrahedral Meshes 81
5.1. Collapsing Algorithm . . . 82
5.1.1. Algorithm Design . . . 82
5.1.2. Finding Admissible Edges for Collapsing . . . 84
5.1.3. Finding Independent Sub-Meshes . . . 84
5.1.4. Collapsing Edges . . . 85
5.1.5. Determinism of Conflict Detection . . . 86
5.2. Collapsing for Mesh Improvement . . . 86
5.3. A Method for Massively Parallel Mesh Adaptation using Error Estimation . . . 87

5.4. Evaluation 89
5.4.1. Robustness 90
5.4.2. Conflict Detection 90
5.4.3. Run Time Performance 92
5.4.4. Skipping Iterations that Collapse only Few Edges 94
5.4.5. Mesh Quality Improvement of Harmonic Optimization with Collapsing 95
5.5. Summary 97

6. User-guided Unstructured Tetrahedral Mesh Editing 99
6.1. Face Groups for Interactive Mesh Modification 100
6.1.1. Assigning Face Groups using Annotation from CAD 101
6.1.2. Extracting Face Groups from the Surface Geometry 101
6.1.3. Finding Feature Edges between Face Groups 103
6.2. Volumetric Mesh Editing Operations 105
6.2.1. Volumetric Hole Filling 105
6.2.2. Volumetric Mesh Erosion 106
6.3. Evaluation of Tetrahedral Mesh Editing based on Face Groups 107
6.3.1. Evaluation of Face Group Detection 108
6.3.2. Run Time Performance and Element Quality for Hole Closing 110
6.3.3. Run Time Performance and Element Quality for Erosion 113
6.4. Evaluating the Capabilities of Cage-based Deformation 114
6.4.1. Comparing Cage Generation Methods 115
6.4.2. Locality of Coordinate Types 117
6.4.3. Cage Connectivity Support of Coordinate Types 117
6.4.4. Shape Preservation of Coordinate Types 118
6.4.5. Computational Cost for Coordinate Computation 119
6.4.6. Element Quality of Deformed Tetrahedral Meshes using Cage-based Deformation . . . 120
6.4.7. Comparing Coordinate Types 122
6.5. Summary 123

7. Massively Parallel Post Processing of Unstructured Tetrahedral Meshes for Analysis 125
7.1. Octree Linear Bounding Volume Hierarchy 126
7.1.1. Quantization of AABBs along the Morton Curve 126
7.1.2. Data Structure Layout 127
7.1.3. Construction 128
7.1.4. Traversal 131
7.2. Direct Volume Rendering using the OLBVH 132
7.2.1. Empty Space Skipping 133
7.2.2. Sample the Simulation Results along View Rays 134
7.3. Coarsening Meshes for Direct Volume Rendering 135
7.4. Evaluation of Post-Processing Performance 136
7.4.1. Evaluation Meshes 136
7.4.2. Performance of OLBVH Construction 137
7.4.3. Performance of Sampling Throughput for DVR 138
7.4.4. Profiling DVR Performance 140
7.4.5. Performance of Coarsening for DVR 143

7.5. Conservative Slicing 145
7.5.1. Algorithm for Conservative Slicing using the OLBVH 145
7.5.2. Runtime Performance of Conservative Slicing 146
7.6. Summary 147

8. Conclusion and Future Work 149
8.1. Answering the Research Questions 149
8.2. Key Conclusions 151
8.3. Limitations 153
8.4. Future Work 154
8.4.1. Avenues for Faster Generation and Optimization of Unstructured Tetrahedral Meshes . 154
8.4.2. Avenues for Shorter Design Cycles in Virtual Prototyping 155
8.4.3. Avenues to Advance Rendering Performance 156
8.4.4. Avenues for Massively Parallel Processing of Unstructured Hexahedral Meshes 156

References 157

Appendices 185

A. Glossary 187

B. Supervisory Activities 191
B.1. Lectures 191
B.2. Practicals 191
B.3. Seminars 191
B.4. Bachelor's Theses 191
B.5. Master's Theses 191

C. Conference Presentations of Published Papers 192

1. Introduction

1.1. Motivation

Interactive design of geometry with computers is a widely-used technique in product development. Whenever the idea of a product comes to mind, people can create a digital mock-up or to be more precise a virtual prototype, in order to design and analyze the product virtually. Therefore, the use of a virtual prototype can relieve people from the labor and material cost to construct a physical prototype for testing as well as design purposes. The process of constructing and testing a virtual prototype that represents a real physical model is called virtual prototyping (VP) [Wan02]. Today, the use of VP is ubiquitous in many industries.

Since VP is not only concerned with the model's shape but also with an in-depth analysis of the intended functions of the model, VP involves many consecutive tasks. As this thesis focuses on volumetric meshes, fig. 1.1 provides an overview on the typical tasks of a VP process that involves volumetric meshing. For visualizing and manipulating the shape of a prototype, it is sufficient to work on the boundary structures. Therefore, the initial task of VP is to specify the shape of the prototype using computer-aided design (CAD). Many CAD applications rely on boundary representations (B-Reps) such as non-uniform rational B-splines (NURBS) [Far02]. Oftentimes, the B-Reps describe a continuous surface, in order to meet the potentially high precision demands of modeling smooth or high-fidelity surface features. However, as surface meshes are the data structure of choice for fast rendering using massively parallel graphics processing units (GPU)s, time critical applications such as interactive visualizations rely on surface meshes. Therefore, the use of continuous B-Reps in CAD typically imposes the overhead of discretizing continuous curves into surface meshes for visualization [LB16; SF19].

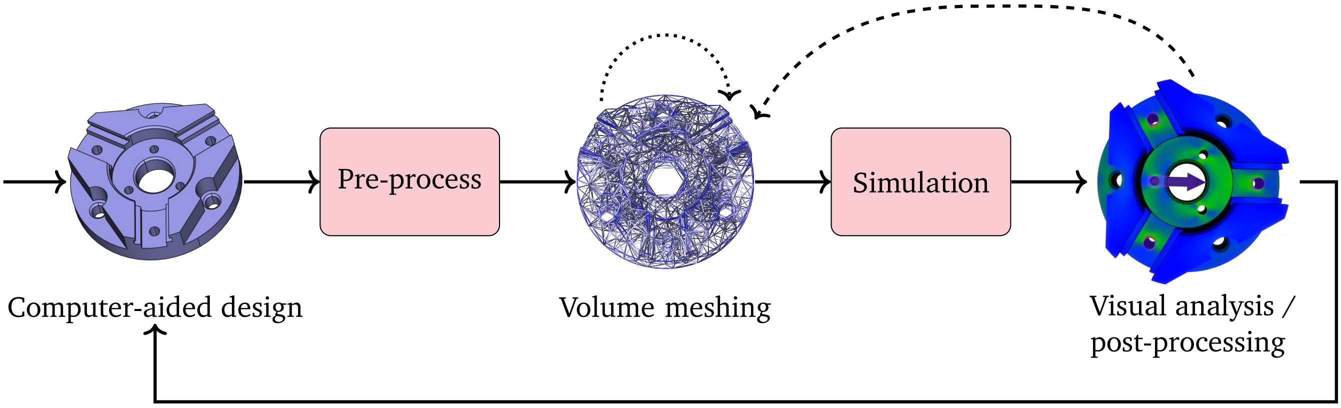


Figure 1.1.: The VP cycle starts with CAD defining B-Reps. Pre-processing prepares a surface mesh for downstream volumetric meshing. A meshing tool then generates a volumetric mesh for numerical simulation. Post-processing tools allow for the exploration of simulation results. After exploration of the simulation results, the development team decides whether the design should be changed or not. For every design change, the steps of the cycle repeat.

A crucial limitation of B-Reps and surface meshes is their restriction to the boundary of the model. As the boundary cannot be used to describe physical properties such as elasticity or thermal conductivity in the interior of the model, many analysis applications rely on a volumetric representation of the model. Numerical methods such as the finite element method (FEM) are frequently used to implement applications working with a volumetric representation. A commonly applied method to obtain a volumetric representation is the Delaunay triangulation (see section 2.1.2). Although boundary conforming Delaunay mesh generation for models with complex boundaries is a challenging problem, because some boundaries cannot be preserved without the insertion of additional vertices, the geometry processing community has devised practical methods for meshing these boundaries while only inserting few vertices [She02a; Dia+23]. As a result, unstructured tetrahedral meshing for the FEM is nowadays well-established for stability reasons.

After the generation of an unstructured tetrahedral mesh, engineers can specify a multitude of simulation scenarios to simulate the prototype. With the configuration of material properties such as Young's modulus, engineers can specify the material of a prototype and experiment with different materials. In addition, engineers specify boundary conditions constraining where the state of the model remains fixed and which parts of the model are associated with loads such as forces or heat inputs. Using the specification of a simulation scenario, engineers can apply physically based simulation to numerically simulate the prototype under real world circumstances. A common choice for numerical simulation is finite element analysis (FEA) using the FEM. Quick FEA is possible using the massively-parallel algorithms and GPU-optimized data structures of Weber et al. [Web+13; Web+15]. In order to obtain an accurate FEA, three properties of the unstructured tetrahedral mesh are of major importance [Sch+18]:

- mesh resolution, i.e., the number of elements
- order of elements, i.e., polynomial degree of elements
- tetrahedral element shape quality

As increasing the number of tetrahedral elements in the mesh typically improves the accuracy of the numerical solution of the partial differential equation (PDE), mesh resolution is an important property when creating a mesh for numerical simulation. In addition, the use of a finer mesh resolution, enables the meshing tool to better approximate continuous B-Reps, which reduces the discretization error. The order of tetrahedral elements corresponds to the order of the polynomial basis functions that describes the geometry of the elements [Jia+21]. Using higher-order elements improves the discretization accuracy but requires the use of the higher-order FEM, which imposes significantly more computation time. The shape of a tetrahedral element should be close to an equilateral element, because numerical simulation algorithms may provide inaccurate results or even crash for ill-shaped tetrahedral elements [She02b]. Various shape quality metrics have been proposed over the years [She02b; Rab+17; Ale19; Sor+23], which enable improvement of the shape of tetrahedral elements by optimization of the quality metric. However, shape quality metrics are not consistent with the Delaunay criterion, leading Delaunay triangulation algorithms to produce flat and ill-shaped tetrahedra [Lo15]. For this reason, meshing tools perform an element shape optimization step after mesh generation. While increasing mesh resolution or the order of elements are common tasks in present days, avoiding ill-shaped elements in a mesh is a difficult task.

Subsequent to simulating the prototype with a suitable unstructured tetrahedral mesh, the engineers inspect the simulation results to evaluate if the current prototype is a viable design. For visualizing the results of FEA, a post-processing tool allows for exploration of the simulation results. In order to allow for interactive exploration in 3D, present-day post-processors such as ParaView [Aya15] rely on parallel processing. If users are interested in data related to the surface only, e.g., displacement, visualizing the boundary of the model is sufficient. However, visualization of the simulation results in the interior of the model requires visualization of the inner structures. One of the common approaches for visualizing data associated with the inner structures of a model is direct volume rendering (DVR) of the volumetric mesh. With the use of DVR, engineers can interactively explore the simulation results by configuring the rendering to highlight the simulation results of interest.

When engineers have investigated the simulation results using the post-processing tool of choice, they decide whether the current design is viable or not. Oftentimes the design does not exhibit the desired properties or other optimization goals such as economical use of material are not met sufficiently. Therefore, engineers modify the prototype many times. Repeated iterations of the design cycle result in substantial development times hindering the fast creation of product designs meeting customer demands. The issue of development overheads of design cycles becomes even more significant, because design cycles occur more frequently and involve an increasing number of iterations due to the trend towards more customized and individualized products [Sto15]. An additional overhead for each design cycle is the conversion of model data between individual tasks of the cycle. In order to reduce transitions in design cycles, the engineering industry has invented methods to shorten design cycles. Popular examples are editing techniques for manipulating the discrete mesh (see section 3.3) or isogeometric analysis for simulating on CAD geometries (see section 3.7). Nevertheless, there still is a need for methods to reduce development times and shorten design cycles in VP [Bis17; LAR20; Por+21; Li+21].

A major overhead in VP cycles is the meshing step, because the shape quality of the elements needs to be sufficient for simulation. As the commonly used Delaunay mesh generation is not consistent with shape quality metrics [Lo15], this frequently requires optimization and re-meshing of the volumetric mesh (see dotted arrow in fig. 1.1). Therefore, many methods aim at relieving the development team from looping back to CAD, in order to save the development time required for repeated generation of simulation-suitable meshes (see dashed arrow in fig. 1.1). In particular, methods that alter the unstructured mesh to accelerate VP cycles, e.g., shape optimization, suffer from additional computational overheads due to repeated re-meshing for avoiding insufficient element quality [OSW23]. Recently, Alexa [Ale19] proposed the harmonic triangulation (see section 2.1.3) that is related to the Delaunay triangulation but is generated by minimization of an element quality metric. Therefore, harmonic triangulation is an interesting alternative, as its use could mitigate the element quality issues for numerical simulation applications.

In order to facilitate meshing in VP processes, many tools for optimizing and re-meshing unstructured meshes are available [KS07; Iba+17; Arp+22]. However, the bulk of these tools still relies on predominantly serial processing on the central processing unit (CPU). As the scale and complexity of engineering tasks are ever increasing, the numerical calculation of PDEs has been a driving factor for high-performance computing (HPC) [RGD22]. While HPC typically orchestrates many machines as nodes of a cluster for coarse-grained parallelism, the fine-grained parallelism of present-day GPUs is highly effective in accelerating a comprehensive set of tasks on a single machine [Ped23]. Since the performance benefit of using the GPU is suboptimal for coarse-grained parallelism, apt parallelization strategies are required to achieve fine-grained parallelization of re-meshing and mesh optimization. In the context of VP, several methods enable the exploitation of the GPU to accelerate the generation of high-resolution Delaunay meshes [Els+24]. For faster re-meshing, current mesh adaptation tools such as NASA's refine [Par22] or Sandia National Laboratories' Omega_h [Iba22] adopt CUDA (see section 2.2) to boost on-node parallelism using the GPU. While the mesh data structure of Mueller-Roemer et al. [MS18; Mue20] enables massively parallel processing of simplicial meshes for VP, the apt usage of massively parallel hardware for robust mesh optimization and re-meshing methods remains as an open problem [Pan22].

As numerical simulations are potentially executed on high-resolution volumetric meshes for accuracy reasons, post-processing applications should be able to cope with high memory demands. Especially, GPU-based rendering applications that visualize unstructured meshes are prone to occupy a large amount of memory [WMZ21]. This can lead to issues with large meshes as the available memory on a GPU is limited. Therefore, there is a need for memory-efficient post-processing.

1.2. Approach

As an overall goal, this thesis aims at facilitating the usage of GPUs in VP to reduce development time. In order to accelerate mesh optimization and re-meshing tasks, this thesis builds upon the mesh data structure of Mueller-Roemer et al. [MS18; Mue20] to devise massively parallel algorithms that enable the use of the GPU for changing the geometry and connectivity of volumetric meshes. Due to the ubiquity of unstructured tetrahedral meshes, the focus lies on this type of volumetric mesh. Since the methods for the constrained generation of unstructured tetrahedral meshes are well-established and involve many steps that are inherently serial, this thesis focuses on the methods to optimize and re-mesh unstructured tetrahedral meshes so that they are suitable for numerical simulations. As the harmonic triangulation is a promising Delaunay-related method to obtain good element shape quality, it is interesting to use this triangulation type for massively parallel mesh optimization. The efficiency of massively parallel algorithms facilitates interactive mesh editing, which enables short VP cycles without changing the geometry in CAD. Thus, this thesis also investigates the use of massively parallel mesh optimization algorithms for mesh editing. These optimization algorithms can improve mesh quality so that edited tetrahedral meshes can be used for numerical simulation. Fast simulation feedback might be possible immediately after design changes through coupling quick mesh editing with available methods for massively parallel FEA [Web+13; Web+15]. Which methods enable convenient mesh editing is also an important question for improving development overheads in VP. Thus, this thesis explores methods for interactively editing unstructured tetrahedral meshes. Since the investigation of simulation results is indispensable for VP, this thesis is concerned with efficient post-processing. Due to the increased memory demands of maintaining meshes on the GPU, memory efficient post-processing is also one of the goals of this thesis.

1.3. Research Questions

Exploring the approach of this thesis is subject to investigating the following research questions (RQ)s:

- **RQ1:** ***How can the use of the GPU accelerate mesh optimization and re-meshing tasks for unstructured tetrahedral meshes?***

  Massively parallel processing is only efficient for performing many tasks of little work that can be executed independently. The use of unstructured tetrahedral meshes for numerical simulation requires the mesh to be valid and of good shape quality. Changing the connectivity of mesh elements in parallel can lead to issues, if the connectivity of adjacent elements is changed simultaneously. Therefore, massively parallel re-meshing needs to ensure a valid mesh by using sophisticated parallelization strategies that prevent conflicting changes to element connectivity. In addition, good run time performance is not guaranteed through the usage of the GPU, because the fastest GPU will provide slow run times, if the convergence of an optimization or re-meshing algorithm is inadequate.

- **RQ2:** ***How can massively parallel optimization and re-meshing of unstructured tetrahedral meshes robustly produce meshes of sufficient quality for numerical simulations?***

  The utility of meshing algorithms not only depends on run time performance but also on the reliability of the algorithms to produce meshes that are suitable for downstream applications. Therefore, this thesis not only investigates run time performance but also the robustness of the proposed algorithms. While robustness usually refers to the ability of a meshing algorithm to produce valid results, the shape quality of the tetrahedral elements is also of major importance for downstream numerical methods. For this reason, the shape quality of the resulting meshes needs to be investigated as well.

- **RQ3:** ***How can massively parallel mesh processing be used for quick editing of unstructured tetrahedral meshes to accelerate VP cycles?***

  Methods to accelerate the VP cycle are needed and fast optimization and re-meshing of unstructured tetrahedral meshes can be a good foundation for interactive mesh editing. In particular, the combination with massively parallel FEA [Web+13; Web+15] seems attractive, because it admits fast feedback after changes to the prototype. Of course, this question depends on RQ1 and RQ2, since efficiency of the algorithm and quality of their results need to be sufficient for fast mesh editing coupled with numerical simulation. In addition, this question depends on convenient methods to interact with the mesh so that users are able to specify the mesh editing operations intuitively, because fast mesh editing is useless, if its usability is inadequate.

- **RQ4:** ***How can the massively parallel post-processing of unstructured tetrahedral meshes be implemented with efficient memory usage?***

  Reducing the memory occupation of post-processing volumetric meshes is an important and continual issue for large meshes. As the approach outlined in section 1.2 proposes to perform not only the visualization but also the re-meshing and optimization of unstructured meshes on the GPU, efficient memory usage is even more desirable, because additional computations are performed that impose additional memory allocations. As one of the most significant memory overheads for post-processing is the spatial data structure [WMZ21], this thesis explores, if it is possible to obtain a spatial data structure for memory efficient as well as massively parallel post-processing. This spatial data structure should be practical for DVR and other post-processing applications. While a memory efficient spatial data structure for massively parallel post-processing is interesting, the compression of the mesh using massively parallel re-meshing is also an interesting avenue that is explored in this thesis.

1.4. Publications and Contributions

While I investigated these research questions, I published the results of my scientific research in several papers and presented the contributions of my work at several conferences. The following list provides an overview on the contributions for each previously published paper in chronological order:

Paper	**Contributions**
[Str+20] **D. Ströter**, J. S. Mueller-Roemer, A. Stork, D. W. Fellner, "OLBVH: octree linear bounding volume hierarchy for volumetric meshes". In: *The Visual Computer* 36.10-12 (July 2020). **Honorable mention from Fraunhofer IGD for best papers in the category "Impact on Science"**, Presented at Computer Graphics International 2020, pp. 2327–2340. DOI: 10.1007/s00371-020-01886-6 (addressing RQ4)	This paper contributes a spatial data structure for unstructured volumetric meshes. As this data structure organizes an octree in a compact offset-based linear data layout it is denoted as octree linear bounding volume hierarchy (OLBVH). For fast run time performance, the OLBVH can be constructed and traversed in massively parallel manner on the GPU. In addition, the memory consumption of this spatial data structure can be controlled by a tuning parameter that governs tree depth. This enables control over the memory consumption to fit the available memory of the used GPU. For analysis purposes, this paper describes an approach for interactive DVR using the novel data structure. As a result of marking boundary tree nodes, a method for skipping empty space along view rays further improves rendering performance.

[Str+21] **D. Ströter**, U. Krispel, J. Mueller-Roemer, D. Fellner, “TEdit: A Distributed Tetrahedral Mesh Editor with Immediate Simulation Feedback”. In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Presented at SIMULTECH 2021. SciTePress 2013. SCITEPRESS - Science and Technology Publications, 2021, pp. 271–277. DOI: 10.5220/0010544402710277 (addressing RQ3)	This paper contributes interactive editing operations for customizing models using only unstructured tetrahedral meshes. An application provides implementations of editing operations. For user interaction, the application relies on a set of face groups that can be selected by the user to specify editing operations. For faster VP, the application triggers immediate simulation feedback to the user, whenever the user customizes the model. The proposed editing operations are hole closing and erosion. The hole closing operation enables to remove planar drill holes from a model, e.g., to improve the stability of the model. The erosion operation allows for the removal of model parts, e.g., to save material and production cost.
[Str+22] **D. Ströter**, J. Mueller-Roemer, D. Weber, D. W. Fellner, “Fast harmonic tetrahedral mesh optimization”. In: *The Visual Computer* (June 2022). **Visual Computer Best Paper Award at Computer Graphics International 2022**. DOI: 10.1007/s00371-022-02547-6 (addressing RQ1, RQ2, and RQ3)	This paper contributes a fast algorithm for optimizing unstructured tetrahedral meshes for numerical methods. This algorithm builds upon the framework of harmonic triangulation that provide efficient Delaunay-based optimization and low element counts. This publication extends the harmonic triangulation framework by a new way to compute the energy gradient for individual mesh vertices and a robustly converging Gauss-Seidel iteration scheme. Furthermore, the optimization of vertex positions on the boundary using directional derivatives is explored in depth. As an additional contribution, a robustly converging method to relocate vertices along a given linear mesh boundary is proposed and evaluated. Furthermore, a method for massively parallel optimization-driven flipping is proposed that performs locally most beneficial flips in terms of the to-be-optimized energy functional.
[Str+23] **D. Ströter**, A. Halm, U. Krispel, J. S. Mueller-Roemer, D. W. Fellner, “Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation”. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Gerd Wagner, Frank Werner, Tuncer Oren, and Floriano De Rango. Springer International Publishing, 2023, pp. 24–42. DOI: 10.1007/978-3-031-23149-0_2 (addressing RQ1 and RQ3)	This paper is an extended version of the second paper [Str+21] in the list. Performance evaluation of the editing operations from that paper illuminates their most significant performance overheads that can be reduced by parallel processing. The extended paper incorporates some of the massively parallel algorithms developed throughout my research into the unstructured mesh editing operations to boost run time performance. Moreover, the extended paper presents a massively parallel method to extract the boundary of an unstructured tetrahedral mesh and to generate a mapping of surface mesh facets to tetrahedral mesh facets. Furthermore, this paper addresses the maintenance of face groups throughout model customization.

[SSF23] **D. Ströter**, A. Stork, D. W. Fellner, “Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes”. In: *High-Performance Graphics - Symposium Papers*. Ed. by Jacco Bikker and Christiaan Gribble. Presented at High-Performance Graphics 2023. The Eurographics Association, 2023. DOI: 10.2312/hpg.20231139 (addressing RQ1, RQ2, RQ3, and RQ4)	This paper contributes a fast method to find non-overlapping sub-meshes in a simplicial mesh. The set of sub-meshes includes many small and compact sub-meshes, which allows for for massively parallel re-meshing. As the sub-mesh determination prioritizes sub-meshes by a pre-determined functional, the method is configurable, which provides applicability to many different re-meshing tasks. An additional contribution is an algorithm for massively parallel collapsing of edges. This algorithm first applies the method to find non-overlapping sub-meshes and subsequently collapses interior edges of the determined sub-meshes. Due to the application of rules for boundary treatment, this algorithm preserves the boundary of the unstructured mesh well.
[Bue+24] M. v. Buelow, **D. Ströter**, A. Rak, D. W. Fellner, “A Visual Profiling System for Direct Volume Rendering”. In: *Eurographics 2024 - Short Papers*. Ed. by Ruizhen Hu and Panayiotis Charalambous. The Eurographics Association, 2024. DOI: 10.2312/egs.20241030 (addressing RQ4)	This paper contributes a profiling system specifically for DVR, which allows for visual analysis of recorded performance data. The system enables to measure many performance metrics such as cache hit rates or branching and interpolates these metrics on a color scale for each pixel of the DVR image. The application of this profiling system to the OLBVH-based DVR enables an in-depth investigation of the performance bottlenecks. As a result, the analysis allows to devise approaches to improve the performance of DVR. The development of the visual profiling system was primarily the work of the corresponding author. I adjusted my DVR application for the analysis with the profiling system and collaborated in the configuration of the profiling and analysis of the results.
[Str+24] **D. Ströter**, J. M. Thiery, K. Hormann, J. Chen, Q. Chang, S. Besler, J. S. Mueller-Roemer, T. Boubekeur, A. Stork, D. W. Fellner, “A Survey on Cage-based Deformation of 3D Models”. In: *Computer Graphics Forum* 43.2 (May 2024). Presented at EUROGRAPHICS 2024. DOI: 10.1111/cgf.15060 (addressing RQ3)	This survey paper reviews the state of the art of cage-based deformation, which is an interactive technique to manipulate geometry. In a systematic way, the survey identifies the advantages and disadvantages of individual methods for cage-based deformation. The survey organizes the available methods for generating cages and computing coordinates for cage-based deformation control into categories based on their commonalities. As a result, the survey assists the reader in choosing suitable methods for the use case in mind. Thus, readers can use this survey, in order to build their own applications. In addition, the systematic evaluation of the state of the art illuminates current shortcomings in the field of cage-based deformation. Thus, the survey provides means to devise future avenues for research to improve the capabilities of interactive manipulation of geometry.

1.5. Outline

This thesis is structured into eight chapters so that each chapter relies on the chapters before. In this way, readers can focus on the particular topics of interest, while cross references link the readers to the necessary information to understand the text.

After this introductory chapter, chapter 2 briefly describes the preliminary concepts of this thesis. It defines the notation that is consistently applied in this thesis, in order to prevent confusion due to unexpected usage of notation. In addition, the descriptions in chapter 2 further elaborate important aspects of the core concepts so that readers are able to gain a deep understanding of many design choices for the presented massively parallel algorithms.

Chapter 3 highlights previous work on the issues addressed in this thesis. The purpose of this chapter is twofold. First, the reader can obtain a comprehensive overview on the state of the art and obtain the literature references that contain the information of interest. Second, the descriptions in chapter 3 clarify the differences of the methods in this thesis to the methods presented by other authors, which should stress the necessity of the methods proposed in this thesis.

As the first chapter presenting the proposed methods, chapter 4 describes a massively parallel algorithm for harmonic optimization of unstructured tetrahedral meshes. This chapter presents optimization and parallel re-meshing techniques that are amenable to the GPU. An evaluation critically investigates the run time performance, robustness, convergence and resulting element quality. This chapter is based on the paper about fast harmonic mesh optimization[Str+22] and the boundary extraction presented in the extended paper about user-guided editing of unstructured tetrahedral meshes using face groups [Str+23].

Subsequently, chapter 5 extends the massively parallel re-meshing capabilities by edge collapse operations, which enables coarsening of unstructured tetrahedral meshes. It describes how to adapt meshes to numerical simulations exploiting the GPU. Again, a critical evaluation investigates the proposed algorithms. The content of chapter 5 is primarily based on the methods presented in the paper about massively parallel collapsing edges in an unstructured tetrahedral mesh [SSF23].

After introducing basic massively parallel optimization and re-meshing methods, this thesis proceeds with chapter 6 that describes user-guided methods for editing unstructured tetrahedral meshes. Besides presenting interactive mesh editing techniques, chapter 6 describes further extensions to the massively parallel re-meshing methods specifically for interactive mesh editing. In addition, chapter 6 provides a comprehensive evaluation of cage-based deformation for interactive mesh deformation, featuring advantages and disadvantages of individual methods. The content is primarily based on the papers about editing tetrahedral meshes with face groups [Str+21; Str+23], while the evaluation has been published in a survey [Str+24].

Since post-processing is also a central topic of this thesis, chapter 7 presents methods for memory efficient and massively parallel post-processing. It primarily shows algorithms for fast and memory efficient DVR on the basis of the spatial data structure presented in the thesis authors master thesis [Str19]. A critical evaluation investigates run time performance and memory consumption. The methods in chapter 7 primarily are based on [Str+20], while the evaluation includes the observations in [Bue+24].

Finally, chapter 8 draws the key conclusions of this thesis. In order to follow up on the central questions of this thesis, chapter 8 provides concise replies to the RQs posed in section 1.3. An overview on significant limitations of the proposed methods and possible avenues for future work close this thesis in a critical way.

2. Preliminaries and Notation

To facilitate clearness and understandability, this chapter presents the preliminary concepts and definitions on notation. As this thesis focuses on the processing and modification of tetrahedral meshes for numerical methods, section 2.1 provides an overview on unstructured triangulations. Processing meshes with GPUs in a massively parallel manner typically relies on common application programming interfaces (API)s, data structures and foundational algorithms that are explained in section 2.2. A quick and versatile approach for user-interactive mesh deformation is presented in section 2.3. Since this thesis addresses DVR of unstructured tetrahedral meshes for post-processing, section 2.4 presents the foundations of volumetric rendering of unstructured grids.

2.1. Unstructured Triangulations

Many tasks in computer graphics or engineering involve the use of unstructured meshing, because it is a robust, versatile and efficient way to discretize geometry. While structured meshes typically organize elements in a uniform pattern, unstructured meshes are not restricted to a pre-determined structure and provide more freedom to organize the connectivity of the mesh. One special case of unstructured simplicial meshes are unstructured tetrahedral meshes. Section 2.1.1 presents the foundational concepts and notations on simplicial meshes. The generation of unstructured simplicial meshes requires sophisticated schemes to efficiently compute and alter the mesh in a robust manner. One well established approach to generate a simplicial mesh is the Delaunay triangulation, which is outlined in section 2.1.2. A novel and interesting notion is the harmonic triangulation that appears in section 2.1.3.

2.1.1. Simplicial Meshes

Simplicial meshes are used to discretize a domain Ω. A d-dimensional simplicial mesh $\mathcal{M} = (\mathrm{V}, \mathrm{T})$ consists of a tuple of vertices V and simplices T. For convenience, the non-zero integers $N_\mathrm{V} \in (\mathbb{N}\backslash\{0\})$ and $N_\mathrm{T} \in (\mathbb{N}\backslash\{0\})$ hold the numbers of vertices $|\mathrm{V}| = N_\mathrm{V}$ and simplices $|\mathrm{T}| = N_\mathrm{T}$, respectively. In order to model the typical array structure of vertices contiguous in memory, the structure of V represents a uniquely ordered sequence of vertices:

$$\mathrm{V} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{N_\mathrm{V}-1}), \quad \text{where} \quad \mathbf{v}_i \in \mathbb{R}^d, \forall i = 0, 1, \ldots, N_\mathrm{V} - 1. \tag{2.1}$$

For each vertex $\mathbf{v}_i \in \mathrm{V}$, the index of $\mathbf{v}_i$ refers to the position i of the vertex in V, e.g., the index of $\mathbf{v}_1$ is 1. Thus, any vertex of the mesh $\mathcal{M}$ is associated with a unique index. A k-simplex $\tau \in \mathrm{T}$ is a tuple of $k + 1 \geq 1$ vertex indices:

$$\tau = (t_0, t_1, \ldots, t_k), \quad \text{where} \quad t_i \in \{j \mid 0 \leq j \leq N_\mathrm{V} - 1\}, \forall i = 0, 1, \ldots, k. \tag{2.2}$$

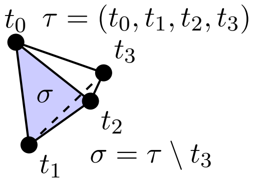
For convenience, the dimension $\dim(\tau) = k$ denotes the dimension of a k-simplex τ. Simplices are provided with a local order of points, because the order is relevant for many quantities such as the face normals or the

Jacobian (see eq. (2.7)). The simplices T are also ordered contiguous in memory. To model the structure of T according to the memory layout T is like V a uniquely ordered sequence:

$$\mathrm{T} = (\tau_0, \tau_1, \dots, \tau_{N_\mathrm{T}-1}). \tag{2.3}$$

Thus, each simplex τ_i is associated with an index i. While all the simplices $\tau \in \mathrm{T}$ are d-simplices, a k-simplex σ of a lower dimension $\dim(\sigma) = k < d$ is called a face of τ, if each vertex of τ is also a vertex of σ. Additionally, τ is a coface of σ. The relationship $\leq$ denotes the inclusion of σ in τ:

$$\sigma \leq \tau \iff \forall t \in \tau, t \in \sigma. \tag{2.4}$$



For any k-simplex τ, the specific face σ of τ opposite to the vertex of index i may be obtained with the set difference $\tau \backslash i = \sigma$. A visualization of the set difference operation appears in the inset to the right. For a 3-simplex $\tau = (t_0, t_1, t_2, t_3)$ the face σ opposite to t_3 is shaded blue. Performing the set difference $\tau \setminus t_3$ provides $\sigma = (t_0, t_1, t_2)$. The notation covers some convenience definitions for typical cases of simplices. If τ is of dimension 3, it is a tetrahedron with volume v_τ. If τ is of dimension 2, it is a triangle with area a_τ. If σ is of dimension $k \leq d-1$ and a face of the d-simplex $\tau \in \mathrm{T}$, then the vertices of σ define a normal $\mathbf{n}_\sigma$ pointing outwards of τ. For instance, $\mathbf{n}_{\tau \setminus t_3}$ denotes the normal of σ in the inset pointing outwards of τ. As it is elaborate to provide set differences for normals and areas of particular faces of τ, the notation provides convenience definitions for areas and normals. For any simplex τ, the normals and areas can be identified as $\mathbf{n}_i = \mathbf{n}_{\tau \setminus t_i}$ and $a_i = a_{\tau \setminus t_i}$.

Besides the topological relationships of simplicial complexes, the notation covers the important geometric quantities of simplices. As matrix formulations are useful for expressing geometric quantities of simplices, $\mathbf{X}_\tau$ denotes the matrix arranging the vertices columnwise in the local order of the k-simplex τ:

$$\mathbf{X}_\tau = (\mathbf{v}_{t_0}, \mathbf{v}_{t_1}, \dots, \mathbf{v}_{t_k}) \in \mathbb{R}^{d \times (k+1)}. \tag{2.5}$$

In order to express the vertex set of a d-simplex τ in relation to its first vertex, the notation provides the matrix $\mathbf{M}_d$:

$$\mathbf{M}_d = \begin{pmatrix} -1 & -1 & \dots & -1 \\ \mathbf{e}_1 & \mathbf{e}_2 & \dots & \mathbf{e}_k \end{pmatrix} \in \mathbb{R}^{(d+1) \times d}, \tag{2.6}$$

where $\mathbf{e}_i$ denotes the ith canonical unit vector of dimension d. With the use of $\mathbf{X}_\tau$ and $\mathbf{M}_d$, it is convenient to calculate the Jacobian $\mathbf{J}_\tau$ of a d-simplex τ:

$$\mathbf{J}_\tau = (\mathbf{v}_{t_1} - \mathbf{v}_{t_0}, \dots, \mathbf{v}_{t_d} - \mathbf{v}_{t_0}) = \mathbf{X}_\tau \mathbf{M}_d \in \mathbb{R}^{d \times d}. \tag{2.7}$$

The evaluation of the Jacobian allows for calculation of the volume or area of a top-level simplex in T. If $\mathcal{M}$ is a 3-dimensional simplicial mesh, then the volume of the tetrahedron $\tau \in \mathrm{T}$ evaluates to $\det(\mathbf{J}_\tau)/6 = v_\tau$. If $\mathcal{M}$ is 2-dimensional simplicial mesh, then the area of the triangle $\tau \in \mathrm{T}$ evaluates to $\det(\mathbf{J}_\tau)/2 = a_\tau$. For all simplices $\tau \in \mathrm{T}$, the local order of vertices shall be arranged such that the determinant of the Jacobian is positive. If for some k-simplex τ the evaluation of $\det(\mathbf{J}_\tau)$ is negative, τ is inverted. A special case occurs, if the vertex positions of a k-simplex τ are linearly dependent. In this case, the Jacobian is not invertible and its determinant vanishes. A k-simplex τ with $\det(\mathbf{J}_\tau) = 0$ is degenerate.

Many numerical methods such as the FEM require the mesh to satisfy certain conditions. For this reason, the characteristic “valid” denotes a mesh that satisfies a basic set of these conditions. A mesh is valid, if it satisfies the following conditions:

1. For any two vertices $v_i, v_j \in \mathrm{V}$ with $i \neq j$, it holds that $v_i \neq v_j$.

2. All the simplices $\tau \in \mathrm{T}$ satisfy $\dim(\tau) = d$.

3. All the simplices $\tau \in \mathrm{T}$ satisfy $\det(\mathbf{J}_\tau) > 0$.

4. For two distinct $\tau, \tau' \in \mathrm{T}$, the intersection $\tau \cap \tau' = \sigma$ is either $\emptyset$ or a simplex with $\dim(\sigma) < d$.

5. For a face $\sigma \leq \tau$ with $\dim(\sigma) = d - 1$ and $\dim(\tau) = d$, there is at most one $\tau' \in \mathrm{T}$ such that $\tau \cap \tau' = \sigma$.

For a numerical method to produce plausible results with a mesh $\mathcal{M}$, it is typically not sufficient for $\mathcal{M}$ to be valid. The quality in terms of element shape is also crucial for numerical methods. As the requirements on element quality depend on the numerical method of choice, this condition is not part of the explicit specifications.

2.1.2. Delaunay Triangulation

For geometry processing, many applications construct a Delaunay triangulation. A Delaunay triangulation is typically a simplicial mesh (cf. section 2.1.1), whose elements $\tau \in \mathrm{T}$ respect the circumball criterion [Che+13a]:

Circumball criterion. *Let $\mathcal{M}$ be a d-dimensional simplicial mesh and σ a k-dimensional simplex for any $k \leq d$. Further, let σ be defined by $k + 1$ vertices in* V. *The simplex σ respects the circumball criterion iff. the open circumball of σ contains none of the vertices in* V.

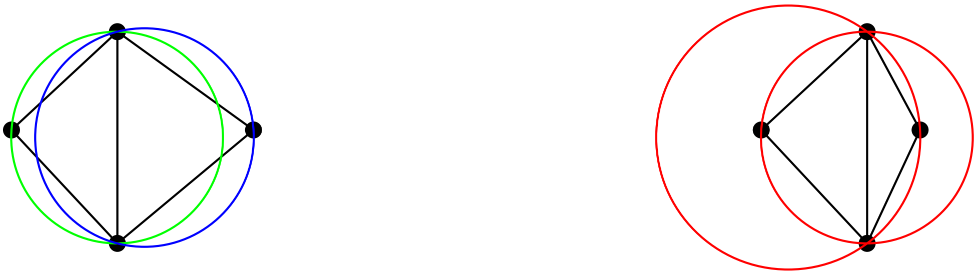


Figure 2.1.: To the left: The two triangles respect the circumball criterion, because the green and blue circles only intersect with the vertices of their corresponding simplices but do not contain any other vertex. Thus, the two triangles are Delauany.
To the right: The two triangles violate the circumball criterion, because their circumcircles contain the vertex of their respective opposite simplex. Thus the two triangles are not Delaunay.

If a simplex satisfies the circumball criterion, the simplex is Delaunay. A Delaunay simplex with a closed circumball containing only the vertices of the simplex is strongly Delaunay. Figure 2.1 shows Delaunay and non-Delaunay simplices. For constructing the Delaunay triangulation, it is mandatory to repair configurations violating the circumball criterion such as shown on the right of fig. 2.1. A frequent operation of use to fix non-Delaunay simplices is performing a bistellar flip. Figure 2.2 visualizes 2-3 and 3-2 bistellar flips that are commonly applied for generating a Delaunay triangulation. Each bistellar flip retriangulates the convex region of adjacent simplices. For every bistellar flip, an inverse flip operation exists, e.g., the 2-3 flip is the inverse operation to the 3-2 flip and vice versa. Thus, bistellar flips enable choosing between two connectivity configurations. For one of the two configurations, the simplices affected by the flip satisfy the circumball criterion. While bistellar flips can certainly ensure the adherence to the circumball criterion for all triangles of a two-dimensional simplicial mesh, it is still an open question, if bistellar flips can guarantee the circumball criterion for all tetrahedra of a three-dimensional simplicial mesh [Che+13b].

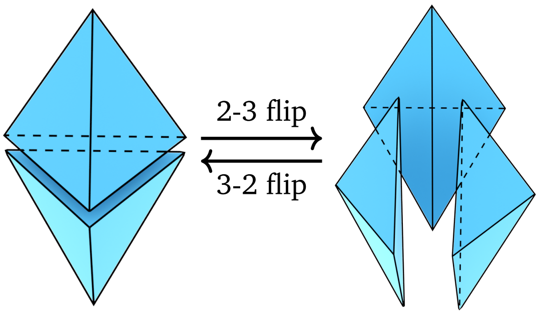


Figure 2.2.: The use of a bistellar 2-3 or 3-2 flip retriangulates a convex region defined by five points. The 2-3 flip replaces two tetrahedra with a shared face by three tetrahedra around an interior edge. The 3-2 flip replaces three tetrahedra with a common interior edge by two tetrahedra with a shared interior face.

Since bistellar flips can at least locally ensure the circumball criterion, they are a useful tool in Delaunay based meshing. In the following, the local view on simplicial meshes serves as a basis to define the Delaunay triangulation. A $(d-1)$-simplex σ of a valid mesh $\mathcal{M}$ (cf. section 2.1.1) either is part of a single d-simplex τ or is shared by two distinct d-simplices τ and τ'. The face σ is locally Delaunay iff. the open circumball of σ neither contains a vertex from τ nor, if $\sigma \leq \tau'$, contains a vertex from the opposite τ'. For instance, the edge connecting the two triangles to the left of fig. 2.1 is locally Delaunay, while the edge on the right of fig. 2.1 is not. This locality is a useful criteria for checking a Delaunay triangulation, as it is part of the following Delaunay lemma [Che+13a]:

Delaunay Lemma. *The valid d-dimensional simplicial mesh $\mathcal{M}$ is Delaunay iff. the following equivalent statements are true:*

1. *Every d-simplex $\tau \in \mathrm{T}$ is Delaunay (i.e. fulfills the circumball criterion).*
2. *Every $(d-1)$-simplex σ with $\sigma \leq \tau$ for any $\tau \in \mathrm{T}$ is Delaunay.*
3. *Every $(d-1)$-simplex σ with $\sigma \leq \tau$ for any $\tau \in \mathrm{T}$ is locally Delaunay.*

As geometry processing is not only concerned with the properties of the bare simplices but also with the conformance of a triangulation to a given model shape, this section also covers the definition of triangulations respecting user specified surfaces. In the meshing domain, a common way to specify a discrete input surface is the piecewise linear complex [Che+13c]:

Piecewise linear complex. *Let $\mathcal{P}$ be a finite set of linear cells. $\mathcal{P}$ is a piecewise linear complex (PLC) iff. it satisfies the following conditions:*

1. *All the vertices and edges in $\mathcal{P}$ belong to a simplicial complex.*
2. *The boundary of each linear cell $\iota \in \mathcal{P}$ consists of the union of linear cells from $\mathcal{P}$.*
3. *If two distinct $\iota, \kappa \in \mathcal{P}$ intersect, the union of linear cells in $\mathcal{P}$ forms the intersection, where all the cells forming the union are of lower dimension than at least one of ι or κ.*

The definition of PLCs covers different input geometries such as triangular surface meshes or point clouds and even allows for the existence of holes. In fact, a PLC in $\mathbb{R}^3$ can include triangles, which are not part of any polyhedron of the PLC. The expression dangling triangle denotes these triangles not properly connected on the face level. Meshing tools such as TetGen [Si20] receive as input a PLC, typically without dangling triangles, and produce a Delaunay mesh respecting the PLC. A Delaunay triangulation respecting a given PLC $\mathcal{P}$ is denoted as the constrained Delaunay triangulation of $\mathcal{P}$. For the constrained Delaunay property, the local view of simplices again serves as a basis for the definition of the triangulation. Thus, the following definition of a constrained Delaunay triangulation in $\mathbb{R}^3$ relies on the constrained Delaunay definition for a single simplex [Che+13d]:

Constrained Delaunay. *For a given PLC $\mathcal{P}$, the k-simplex σ with $k \leq d$ is constrained Delaunay, if the vertices of σ are also in $\mathcal{P}$, σ respects $\mathcal{P}$, and the open circumball of σ contains no vertex from $\mathcal{P}$, which is visible from any point of the relative interior of σ.*

Constrained Delaunay Lemma. *Let $\mathcal{P}$ be a PLC without dangling triangles and $\mathcal{M}$ be a 3-dimensional simplicial mesh. $\mathcal{M}$ is a constrained Delaunay triangulation respecting $\mathcal{P}$ iff. the following equivalent statements are true:*

1. *Every tetrahedron $\tau \in \mathrm{T}$ is constrained Delaunay.*
2. *Every facet σ with $\sigma \leq \tau$ for any $\tau \in \mathrm{T}$ not included in a polygon in $\mathcal{P}$ is constrained Delaunay.*
3. *Every facet σ with $\sigma \leq \tau$ for any $\tau \in \mathrm{T}$ not included in a polygon in $\mathcal{P}$ is locally Delaunay.*

While the constrained Delaunay triangulation is a vital tool in geometry processing, it is important that for some PLCs it is impossible to generate a constrained Delaunay triangulation without creating additional vertices. A prominent example for such a PLC is Schönhardt's polyhedron [Sch28], which can be obtained by rotating one end of a triangular prism. As the argumentation of Schönhardt's polyhedron generalizes to prisms over n-gons with $n \geq 3$ [Ram03], there are many polyhedra that do not admit a triangulation without adding new vertices. In order to efficiently treat PLCs that do not directly admit a constrained Delaunay triangulation, a common strategy is boundary recovery. The boundary is recovered by adding new vertices to the PLC until each segment is strongly Delaunay and inserting facets of the PLC incrementally into the Delaunay triangulation of the vertices in the PLC [She02a]. The remaining tetrahedra on the outside can then be removed by propagating a marking from the outside, if the PLC is closed.

2.1.3. Harmonic Triangulation

Similar to the Delaunay triangulation, the harmonic triangulation is constructed with criteria that shall be satisfied by the simplicial mesh. Alexa [Ale19] posed the notion of harmonic triangulation, which is related to the Delaunay triangulation. While constructing the Delaunay triangulation involves adherence to the Delaunay criterion, the construction of the harmonic triangulation means to solve a minimization problem. For each element $\tau \in \mathrm{T}$ an energy function can be evaluated for the minimization problem. The foundation of the energy function E is Dirichlet energy. Dirichlet energy is typically defined over a function g mapping from a continuous domain Ω:

$$E(g) = \frac{1}{2}\int_{\Omega} \|\nabla g(x)\|^2 \mathrm{d}x, \quad \text{where} \quad g\colon \Omega \to \mathbb{R}. \tag{2.8}$$

In order to formulate harmonic triangulations, Alexa [Ale19] uses a discretized formulation of Dirichlet energy. As a valid d-dimensional triangulation approximates Ω with a set of simplices, the discretized formulation is a sum of the function values $\mathbf{f}_\tau$ for each simplex $\tau \in \mathrm{T}$:

$$E(f) = \frac{1}{2}\sum_{\tau\in\mathrm{T}} v_\tau \|\mathbf{f}_\tau^{\mathsf{T}}\mathbf{M}_d(\mathbf{X}_\tau\mathbf{M}_d)^{-1}\|^2 = \frac{1}{2}\sum_{\tau\in\mathrm{T}} \mathbf{f}_\tau^{\mathsf{T}}\mathbf{L}_\tau\mathbf{f}_\tau = \frac{1}{2}\mathbf{f}^{\mathsf{T}}\mathbf{L}_{\mathrm{T}}\mathbf{f}, \tag{2.9}$$

where $\mathbf{L}_\tau$ and $\mathbf{L}_\mathrm{T}$ are the discrete Laplace-Beltrami operators of τ and T, respectively. The specific entries of the matrix $\mathbf{L}_\tau \in \mathbb{R}^{(d+1)\times(d+1)}$ can be expressed as the common cotan weights:

$$(\mathbf{L}_\tau)_{ij} = \frac{1}{d^2 v_\tau} a_i a_j \mathbf{n}_i^\mathsf{T} \mathbf{n}_j. \tag{2.10}$$

As the trace of the Laplace-Beltrami operator $\mathrm{tr}(\mathbf{L}_\tau)$ is related to the harmonic index [BS07] used for characterization of two-dimensional Delaunay triangulations, minimization aims at reducing the trace. The trace is the sum of diagonal entries given by eq. (2.10):

$$\mathrm{tr}(\mathbf{L}_\tau) = \frac{1}{d^2} \frac{\sum_{i=0}^{d} a_i^2}{|v_\tau|} \tag{2.11}$$

$$\mathrm{tr}(\mathbf{L}_\mathrm{T}) = \sum_{\tau \in \mathrm{T}} \mathrm{tr}(\mathbf{L}_\tau). \tag{2.12}$$

As a result, the trace $\mathrm{tr}(\mathbf{L}_\tau)$ provides a scale dependent shape measure for a simplex τ. Alexa [Ale19] observed that minimizing $\mathrm{tr}(\mathbf{L}_\tau)$ tends to produce equilateral simplices and leads to good shape quality. Dropping the dimensional constant for $\mathrm{tr}(\mathbf{L}_\tau)$ (see eq. (2.11)) leads to the harmonic index η:

$$\eta(\tau) = \frac{\sum_{i=0}^{d} a_i^2}{|v_\tau|}. \tag{2.13}$$

Since the aim for harmonic triangulations is to minimize $\mathrm{tr}(\mathbf{L}_\mathrm{T})$, it is reasonable to use η for operations reducing the trace. One operation to reduce the trace is performing bistellar flips (cf. section 2.1.2). A bistellar flip is a harmonic flip, if it reduces $\mathrm{tr}(\mathbf{L}_\mathrm{T})$. Due to the minimal roughness theorem for Delaunay triangulations posed by Rippa [Rip90], every Delaunay edge flip in a two-dimensional simplicial mesh is also a harmonic flip. Alexa [Ale19] explores the 3-2 and 2-3 flips for the interrelation of the circumball criterion to harmonic flips and finds that

either the harmonic flip is also a Delaunay flip,

or the harmonic flip creates a local triangulation of two tetrahedra, where the Delaunay flip would create a local triangulation of three tetrahedra.

Thus, the harmonic triangulation tends to consist of fewer elements than the Delaunay triangulation. This is an interesting property for numerical methods, because each element costs computationally. In particular, the relationship of the trace, i.e. the minimization function, to element shape is a beneficial property, because ill-shaped elements are avoided so that numerical methods can successfully be applied on harmonic triangulations. For the construction of the harmonic triangulation, Alexa [Ale19] finds that prioritizing harmonic flips by their reduction of the trace leads to goods shape quality. Therefore, it is a good strategy to always perform the harmonic flip, which is most beneficial for the reduction of the trace until no more harmonic flips can be performed.

For the optimization of vertex positions, Alexa [Ale19] proposes a gradient descent scheme. Given a simplex $\tau \in \mathrm{T}$, its gradient can be computed as:

$$\frac{\partial\, \mathrm{tr}(\mathbf{L}_\tau)}{\partial \mathbf{X}_\tau} = \frac{1}{d!} (\mathbf{X}_\tau \mathbf{M}_d)^{-\mathsf{T}} \mathbf{M}_d^\mathsf{T} (\mathrm{tr}(\mathbf{L}_\tau)\mathbf{I} - 2\mathbf{L}_\tau). \tag{2.14}$$

In order to perform gradient descent iterations, the gradient can be evaluated for each simplex, assembling a vector of update directions for the vertices V of the mesh. As the mesh shall be valid after the gradient descent step, a binary search along the direction of a steepest descent finds an inversion free step size λ for vertex relocation. Brent's method then determines an optimal step size for the minimization of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$ within the interval $[0, \lambda]$.

2.2. APIs, Algorithms and Data Structures for Massively Parallel Processing

Over the years, massively parallel processors have gained substantially in their importance. Today, GPUs are one of the most famous accelerators for various time-critical or compute intensive tasks. A popular choice to program GPUs is NVIDIA's CUDA API. Therefore, section 2.2.1 provides a concise introduction to GPU computation and CUDA. As some primitive algorithms are well-established building blocks for more sophisticated massively parallel algorithms, section 2.2.2 presents a collection of these foundational algorithms that is relevant to this thesis. Processing unstructured meshes on a GPU should rely on a massively parallel method to compute the connectivity of facets. For this reason, section 2.2.3 describes a mesh data structure for the purpose of massively parallel processing.

2.2.1. Massively Parallel GPU Computation and CUDA

As one of the core goals of computer graphic applications is realistically visualizing complex 3D scenes in real-time, a lot of processing power is required to render several images per second that display a scene with geometrically detailed objects and realistic lighting. The more complex the geometric detail or the lighting of a scene becomes the more processing power is required for real-time rendering, which imposes a tradeoff between realism and interactivity. The intention to mitigate this tradeoff through increasing the available processing power for rendering, is the driving factor in the invention and steadily improvement of GPUs. Massively parallel GPUs are an apt accelerator for 3D rendering, because the shading of fragments and colors of pixels can be computed performing many small independent calculations that map well to the GPU architecture coupling many simple processors.

While GPUs initially served as a specific-function hardware to accelerate the 3D graphics pipeline, their impressive aggregated processing power was much sought-after in other fields than rendering such as scientific computing. For this reason, the chip industry evolved the GPU from specific-function hardware to the general purpose graphics processing unit (GPGPU) that is programmable and nowadays a ubiquitous accelerator to be found in present-day computers. Like forecast by Nickolls and Dally [ND10] in 2010, the GPGPU architecture is now remarkably widely used and one of the key workhorses for many compute-intensive fields such as high-performance computing [RGD22] or deep learning [MV19].

High-level APIs for parallel computing such as OpenCL [Khr23] or CUDA [NVI23a] enable the convenient control of GPGPUs and the implementation of massively parallel algorithms. The algorithms presented in this thesis are prototypically implemented with CUDA due to its superior performance [Asa+21]. Whereas this choice restricts the implementation to NVIDIA GPUs only, the algorithms do not require CUDA-specific programming primitives but can be executed on most modern GPUs. Despite the convenience due to the usage of an API, one needs to respect the memory locality for data management to design efficient massively parallel algorithms. While all the high-level APIs for GPU programming provide similar models for the locality of memory, the following outlines the methodology how CUDA organizes threads and memory in a hierarchy.

The well-known taxonomy by Flynn [Fly72] classifies GPUs as single instruction multiple data (SIMD) processors. However, the CUDA API organizes processing in a single instruction multiple threads (SIMT) programming model, where a thread is the lowest level of abstraction for performing computations as well as writing to or loading from memory [NVI22b]. While in a SIMD model typically a single thread performs calculations on multiple data using vector registers and processing units, in a SIMT model many threads perform the same operations on arbitrary data. The CUDA API organizes threads in a grid of blocks, as can be seen in fig. 2.3. The CUDA API successively assigns several of the N blocks of a grid to a streaming multiprocessor, which executes the threads of the assigned block. Each block consists of up to 32 warps. One warp includes 32 parallel threads, which may follow divergent control flow paths though the best performance is achieved, when all the threads perform the same instructions as frequently as possible. Thus, branching

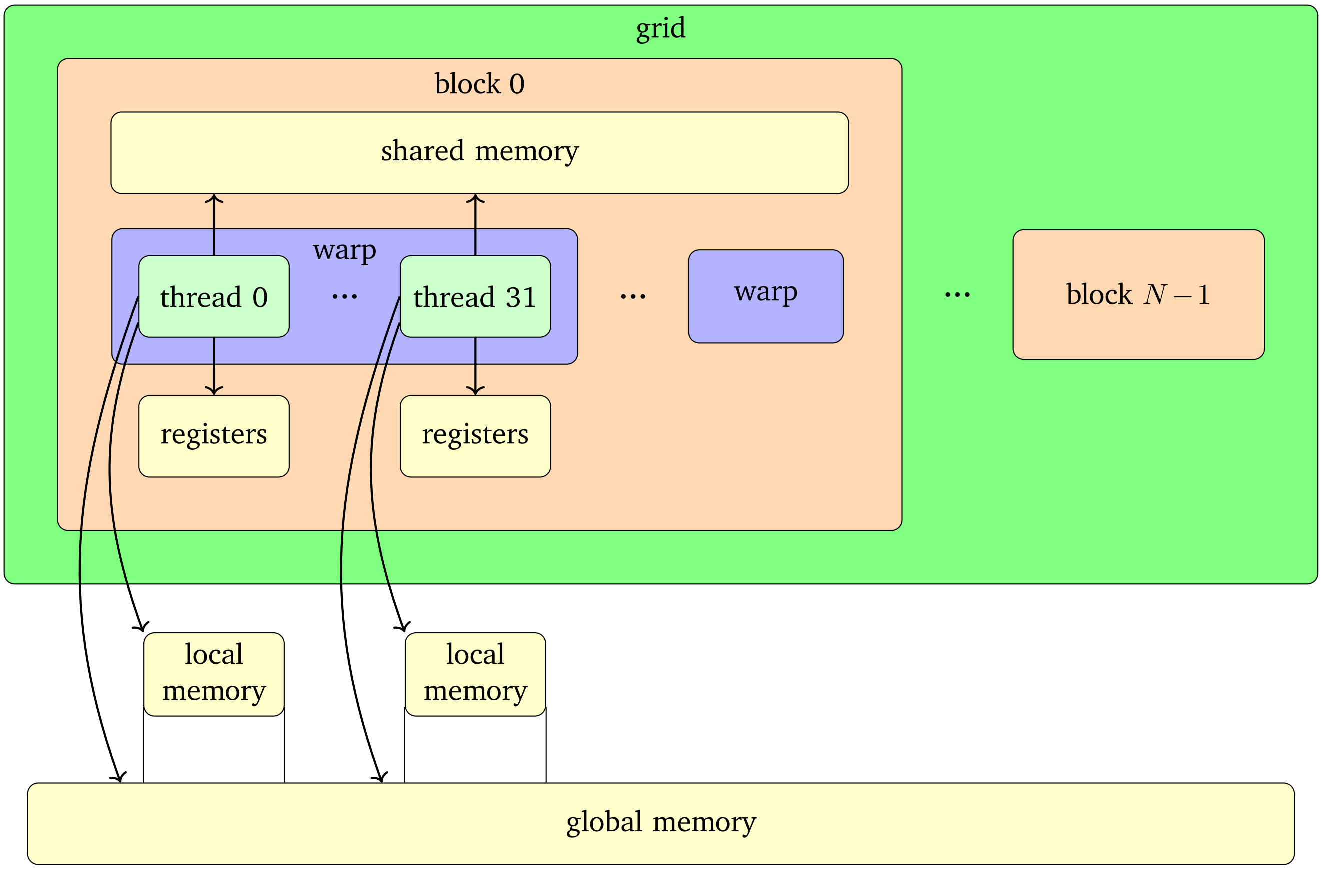


Figure 2.3.: The structure of memory access and thread execution in CUDA.

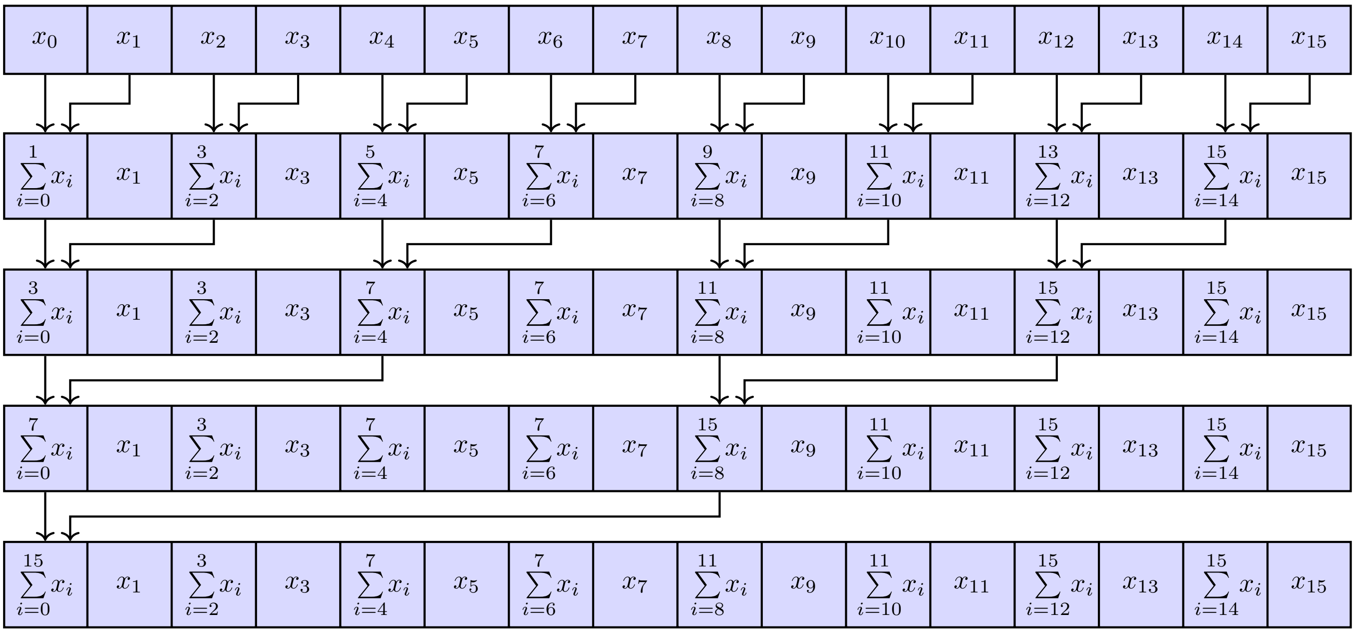
should be avoided in a GPU program. As a benefit of the SIMT model, each thread accesses its own registers for fast data management. Within a warp, threads can exchange data very efficiently using warp-level synchronization. In order to allow for data exchange within a block, each block includes shared memory that can be accessed by the individual threads associated with the block.

The CPU can allocate and copy memory to global GPU memory to provide an interface from CPU to GPU data. In addition, GPU global memory enables threads to exchange data between blocks and to store results that can be loaded by the CPU, when execution is finished. If a GPU-program requires more memory than can be maintained in the registers, each thread can allocate a small limited portion of global memory as its own private local memory that can only be used by the associated thread. The local memory incurs slower access performance but provides much more space than registers. In general, the access to registers and shared memory is fast and can be performed in only a few cycles, whereas the access to global memory potentially requires hundreds of cycles and is much slower. Therefore, a CUDA program should be designed so that its memory is maintained on the storage locations with fast access to well exploit the processing power of a GPU.

2.2.2. Common Algorithms for GPU Programming

Achieving good exploitation of the processing power of GPUs not only depends on smart programming design choices such as the careful consideration of the local proximity of data access or the avoidance of diverging control flows, but also requires the implementation of suitable algorithms. As many problems in computer science involve typical sub-problems such as sorting, some primitive massively-parallel algorithms are established standard methods, today. One of the early attempts to formulate data parallel algorithms for SIMD devices dates back to 1986, when Hillis and Steele [HS86] specified algorithms for solving common problems in a SIMD manner and concluded that these algorithms provide a broad applicability. While many massively parallel algorithms for problems such as sorting saw continuous improvement over the years, other algorithms are now well-established standards. Libraries such as Thrust [NVI23c] or Cub [NVI22a] provide efficient implementations of the well-established algorithms. The implementations for the evaluations in this thesis rely on Thrust. This section covers the massively-parallel algorithms for reduction, prefix scan, and stream reduction, because they are frequently performed by the geometry processing algorithms contributed by this thesis.

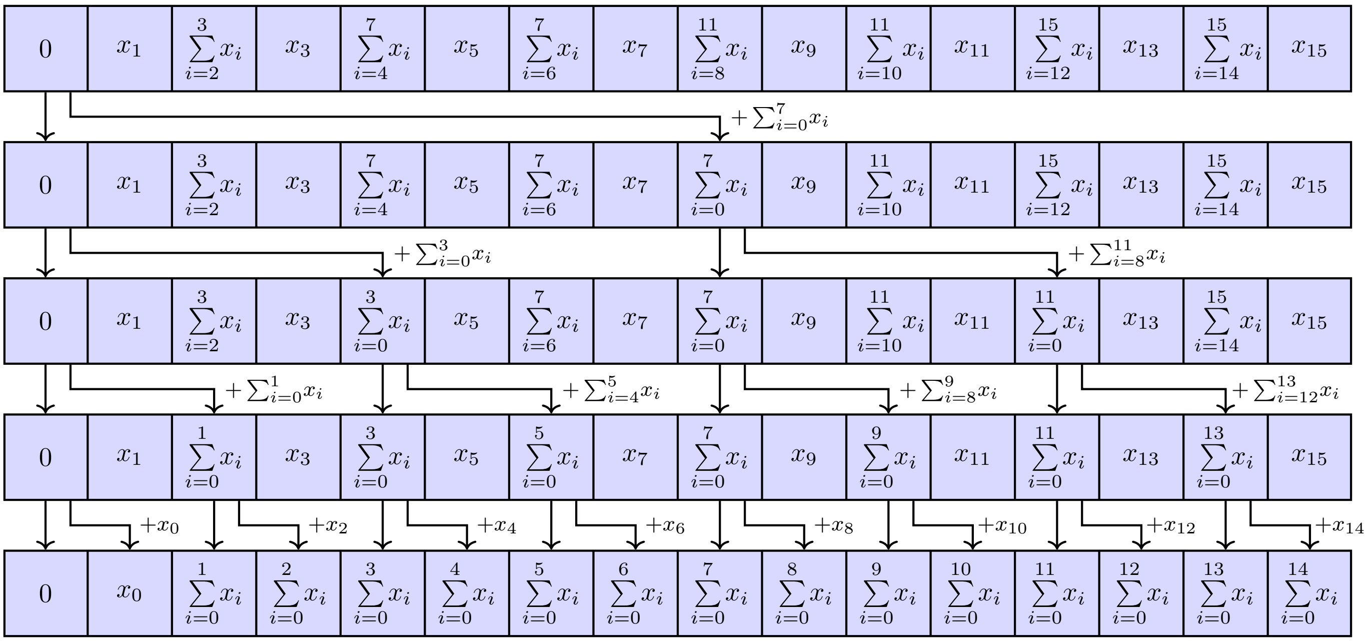
Reduction If an algorithm requires the computation of one value on an entire array of values, a reduction scheme can be used to execute this computation is parallel. Hillis and Steele [HS86] describe a parallel reduction scheme that is frequently used, today. Figure 2.4 shows an example of the reduction scheme for computing the sum of an array of 16 entries $x_0, \dots, x_{15}$. Their scheme organizes the operator application to the elements in a binary tree so that each level of the tree can be executed in a parallel pass. Every processor performs computations on only few elements and writes the result to another array until one entry contains the final result. In this way, the result for an array of N entries can be obtained in $\mathcal{O}(\operatorname{ld} N)$ parallel passes. There are various ways to structure the binary tree. In the example, the structure of the binary tree is such that the final result is in the first entry x_0.



x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
$\sum_{i=0}^{1} x_i$ x_1 $\sum_{i=2}^{3} x_i$ x_3 $\sum_{i=4}^{5} x_i$ x_5 $\sum_{i=6}^{7} x_i$ x_7 $\sum_{i=8}^{9} x_i$ x_9 $\sum_{i=10}^{11} x_i$ x_{11} $\sum_{i=12}^{13} x_i$ x_{13} $\sum_{i=14}^{15} x_i$ x_{15}
$\sum_{i=0}^{3} x_i$ x_1 $\sum_{i=2}^{3} x_i$ x_3 $\sum_{i=4}^{7} x_i$ x_5 $\sum_{i=6}^{7} x_i$ x_7 $\sum_{i=8}^{11} x_i$ x_9 $\sum_{i=10}^{11} x_i$ x_{11} $\sum_{i=12}^{15} x_i$ x_{13} $\sum_{i=14}^{15} x_i$ x_{15}
$\sum_{i=0}^{7} x_i$ x_1 $\sum_{i=2}^{3} x_i$ x_3 $\sum_{i=4}^{7} x_i$ x_5 $\sum_{i=6}^{7} x_i$ x_7 $\sum_{i=8}^{15} x_i$ x_9 $\sum_{i=10}^{11} x_i$ x_{11} $\sum_{i=12}^{15} x_i$ x_{13} $\sum_{i=14}^{15} x_i$ x_{15}
$\sum_{i=0}^{15} x_i$ x_1 $\sum_{i=2}^{3} x_i$ x_3 $\sum_{i=4}^{7} x_i$ x_5 $\sum_{i=6}^{7} x_i$ x_7 $\sum_{i=8}^{11} x_i$ x_9 $\sum_{i=10}^{11} x_i$ x_{11} $\sum_{i=12}^{15} x_i$ x_{13} $\sum_{i=14}^{15} x_i$ x_{15}

Figure 2.4.: Reduction for computing the sum of an array of 16 elements.

Prefix scan A prefix scan applies an operator to all the preceding values of an array entry. If the prefix scan additionally applies the operator to the array entry itself, then the prefix scan is inclusive. Otherwise, the prefix scan is exclusive. Blelloch [Ble90] formulates an efficient parallel scheme of a prefix scan. Firstly, a parallel reduction calculates the sum of values using the binary tree structure of Hillis and Steele [HS86]. Subsequently, the prefix sum is obtained by reverse traversal of the binary tree. This requires to keep all of the intermediate results of the reduction. Figure 2.5 shows the parallel passes for the computation of the prefix of an array of 16 entries $x_0, \dots, x_{15}$. The traversal starts at the root node and replaces its entry with the identity element for the operator of choice. Throughout traversal, every left child receives the value of its parent. Every right child receives the result of applying the operator to the value of the parent and the left child. When traversal terminates, the resulting array contains the exclusive prefix scan of the input array. If an inclusive prefix scan shall be computed, it suffices to left shift the obtained array and insert the total sum into the last array entry. The efficiency of a CUDA implementation of the parallel prefix scan can be significantly improved e.g. by following the advice of Harris et al. [HSO07].



0 | x_1 | $\sum_{i=2}^{3} x_i$ | x_3 | $\sum_{i=4}^{7} x_i$ | x_5 | $\sum_{i=6}^{7} x_i$ | x_7 | $\sum_{i=8}^{11} x_i$ | x_9 | $\sum_{i=10}^{11} x_i$ | x_{11} | $\sum_{i=12}^{15} x_i$ | x_{13} | $\sum_{i=14}^{15} x_i$ | x_{15}
$+\sum_{i=0}^{7} x_i$
0 | x_1 | $\sum_{i=2}^{3} x_i$ | x_3 | $\sum_{i=4}^{7} x_i$ | x_5 | $\sum_{i=6}^{7} x_i$ | x_7 | $\sum_{i=0}^{7} x_i$ | x_9 | $\sum_{i=10}^{11} x_i$ | x_{11} | $\sum_{i=12}^{15} x_i$ | x_{13} | $\sum_{i=14}^{15} x_i$ | x_{15}
$+\sum_{i=0}^{3} x_i$
$+\sum_{i=8}^{11} x_i$
0 | x_1 | $\sum_{i=2}^{3} x_i$ | x_3 | $\sum_{i=0}^{3} x_i$ | x_5 | $\sum_{i=6}^{7} x_i$ | x_7 | $\sum_{i=0}^{7} x_i$ | x_9 | $\sum_{i=10}^{11} x_i$ | x_{11} | $\sum_{i=0}^{11} x_i$ | x_{13} | $\sum_{i=14}^{15} x_i$ | x_{15}
$+\sum_{i=0}^{1} x_i$
$+\sum_{i=4}^{5} x_i$
$+\sum_{i=8}^{9} x_i$
$+\sum_{i=12}^{13} x_i$
0 | x_1 | $\sum_{i=0}^{1} x_i$ | x_3 | $\sum_{i=0}^{3} x_i$ | x_5 | $\sum_{i=0}^{5} x_i$ | x_7 | $\sum_{i=0}^{7} x_i$ | x_9 | $\sum_{i=0}^{9} x_i$ | x_{11} | $\sum_{i=0}^{11} x_i$ | x_{13} | $\sum_{i=0}^{13} x_i$ | x_{15}
$+x_0$ $+x_2$ $+x_4$ $+x_6$ $+x_8$ $+x_{10}$ $+x_{12}$ $+x_{14}$
0 | x_0 | $\sum_{i=0}^{1} x_i$ | $\sum_{i=0}^{2} x_i$ | $\sum_{i=0}^{3} x_i$ | $\sum_{i=0}^{4} x_i$ | $\sum_{i=0}^{5} x_i$ | $\sum_{i=0}^{6} x_i$ | $\sum_{i=0}^{7} x_i$ | $\sum_{i=0}^{8} x_i$ | $\sum_{i=0}^{9} x_i$ | $\sum_{i=0}^{10} x_i$ | $\sum_{i=0}^{11} x_i$ | $\sum_{i=0}^{12} x_i$ | $\sum_{i=0}^{13} x_i$ | $\sum_{i=0}^{14} x_i$

Figure 2.5.: Prefix scan for computing the exclusive prefix sum of an array of 16 entries using the binary tree of the reduction from fig. 2.4.

Stream compaction A stream compaction shrinks an input array to all the entries that satisfy a predetermined predicate. This is useful, when an algorithm shall compact an arbitrarily large array to a significantly smaller array. While some stream compaction approaches rely on binary search [Hor05], many approaches [Hug+13; Bak+17] rely on a parallel prefix scan method. Algorithm 1 outlines the stream compaction method that relies on an exclusive prefix sum and is used in this thesis. The evaluation of the predicate can be performed in a parallel pass over array entries, saving the results in a predicate array. If the predicate is *true* for the array entry i, the i-th entry of the predicate array contains a 1. Otherwise, the i-th entry contains a 0. As this structure resembles a marking of elements, the predicate array is also called *marking array*, in this thesis. An exclusive prefix sum on the *marking array* produces offset positions, which enables to write the elements satisfying the predicate to a new array in an order preserving way. In order to obtain the number of to-be-copied elements, the exclusive prefix sum method computes the total sum e.g. by using an additional array entry at the end of the resulting array for the prefix sum. In this way, the stream compaction can allocate an array of exactly the required size for compaction. A parallel pass then writes all the marked entries of the input array to the offset positions in the compacted array. For copying mesh elements such as triangles or tetrahedra, the offset positions can be used to re-index vertices so that the resulting mesh holds a valid indexation, as presented by Wald [Wal21].

Algorithm 1 Algorithm for a massively parallel stream compaction

```
procedure STREAMCOMPACTION(Array $x$, Predicate $\mathcal{P}$)
    $N \leftarrow x.size()$
    MarkingArray $m \leftarrow allocate(N)$
    for $i \leftarrow 0, \dots, N-1$ do                              ▷ In parallel
        if $\mathcal{P}(x_i)$ then
            $m_i \leftarrow 1$
        else
            $m_i \leftarrow 0$
        end if
    end for
    offsets $\leftarrow allocate(N+1)$
    offsets $\leftarrow$ EXCLUSIVEPREFIXSUM$(m, 0, N)$
    Array dest $\leftarrow allocate(\mathtt{offsets}_N)$
    for $i \leftarrow 0, \dots, N-1$ do                              ▷ In parallel
        if $m_i = 1$ then
            $k \leftarrow \mathtt{offsets}_i$
            $\mathtt{dest}_k \leftarrow x_i$
        end if
    end for
    return dest
end procedure
```

2.2.3. TCSR Simplicial Mesh Data Structure

A geometry processing application using meshes typically relies on a data structure to maintain the mesh. Since RQ2 aims at exploring the capabilities of massively-parallel processing for geometry processing applications, it is important to choose a data structure, which supports fine-grained parallelism. One key challenge for the use of massively-parallel processors is efficient use of memory, because parallel computations of up to millions of threads oftentimes involves a considerable amount of memory occupations allocating arrays for inputs and results. Another key challenge is to provide a data layout for efficient access. Massively-parallel threads can suffer from high memory bandwidths. Thus, the used data structure shall be optimized for coalescing memory accesses is important to achieve high performance.

In order to adhere to the requirements of massively-parallel processing, the ternary compressed sparse row (TCSR) mesh data structure by Mueller-Roemer et al. [MS18; Mue20] enables maintaining a simplicial mesh (cf. section 2.1.1) in a suitable manner for massively-parallel processors such as GPUs. Henceforth, this data structure is referred to as TCSR mesh. TCSR mesh facilitates massively-parallel mesh modeling applications through the quick computation of connectivity relationships. The computation of simplicial element connectivity relationships can be classified into two categories [MAS17]:

1. Top-down: Mapping every k-simplex τ to every $(k-n)$-dimensional face σ with $\sigma \leq \tau$ and $0 < n < k$.

2. Bottom-up: Mapping every $(k-n)$-dimensional face σ to every k-simplex τ with $\sigma \leq \tau$ and $0 < n < k$.

Top-down relationships can be computed with boundary operators ∂_k, while bottom-up relationships are computed with coboundary operators d_k. If a boundary or coboundary operator computes indirect relationships with $n > 1$, the superscript of the operators ∂_k^{n-1} and d_k^{n-1} denotes the dimension of the faces. Figure 2.6 shows an overview of the boundary and coboundary operators. For instance, ∂_3^1 represents the mapping of tetrahedra to their six edges. A matrix expresses the connectivity relationships, where row i represents the k-simplex of index i and column j represents the $(k-n)$-face of index j. If the ijs entry of the matrix is equal to 1, then the $(k-n)$-face of index j is part of the k-simplex of index i. A sign is used to express the orientation of a face to a simplex. Thus, the possible values for an entry in the connectivity matrix are in $\{-1, 0, 1\}$. As a result, the connectivity matrix is ternary. In order to manage memory efficiently, both operator types (boundary and coboundary) organize the resulting connectivity relationships in the compressed sparse row array layout, which only saves non-zero entries. Combined with the ternary matrix representation, this leads to a ternary compressed sparse row layout.

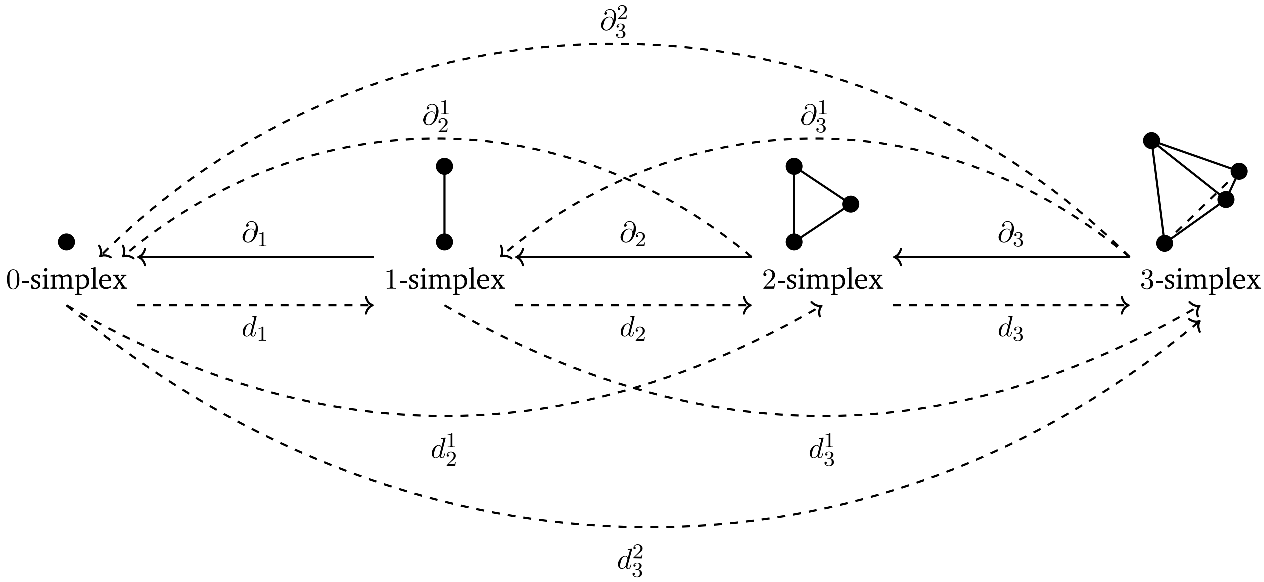


Figure 2.6.: Boundary ∂_k^n and coboundary d_k^n operators. The solid lines represent operators which are always stored, while the dashed lines represent operators that are computed and stored on demand.

As the to-be-allocated memory for boundary operators is predictable, e.g. each tetrahedron includes four vertices, a boundary operator returns an array of face indices. However, the to-be-allocated memory for coboundary operators is not a priori clear, e.g. one vertex can be part of 20, 21 or any other number of tetrahedra. For this reason, coboundary operators require to determine the number of non-zero entries for each k-simplex before allocating a fittingly-sized array. While every coboundary operator can be retrieved by transposing its corresponding boundary operator $d_k = \partial_k^\top$ [MAS17], the computation of some chained coboundary operators can be accelerated by filling larger arrays with sorted tuples and subsequently removing the duplicates [MS18]. The use of the Bin-BCSR* matrix data structure [Web+12; MS18] for coboundary operators provides efficient memory management optimized for access coalescing of massively-parallel processors such as GPUs.

Besides offering fast modeling, simulation applications benefit from massively-parallel assembly of the stiffness matrix, which can be a considerable overhead for simulations [MS18]. This is a useful benefit for providing fast simulation feedback (see RQ3), as it reduces the run time of simulation applications.

2.3. Manipulating Geometry with Cage-based Deformation

While the choice of efficient data structures and algorithms enables the fast processing of meshes, the user needs a way to interactively model the geometry, i.e., the prototype. The modeling step typically is a part of CAD. This section presents a quick and versatile approach to model meshes using an enclosing cage. The shape of the geometry is bound to the cage. As deforming the cage triggers deformation of the geometry, this approach is denoted as cage-based deformation.

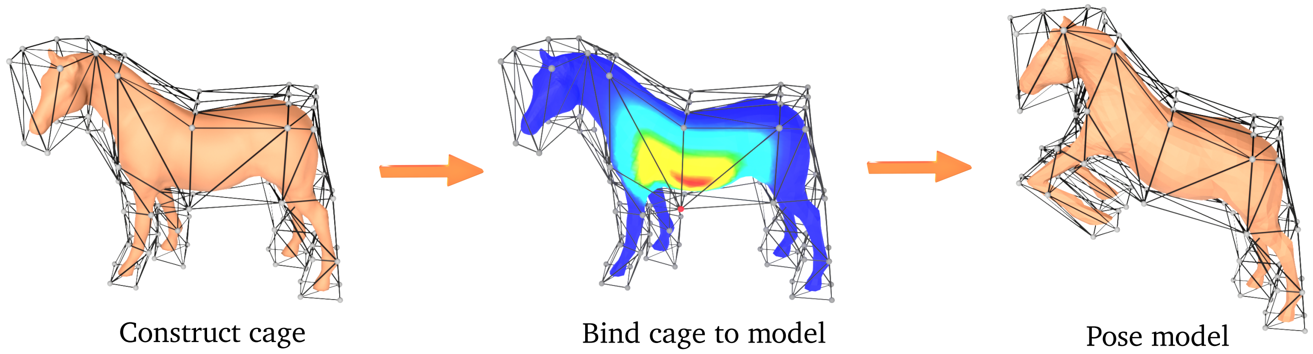


Figure 2.7.: Cage-based deformation workflow: First, designers construct a cage. Calculating coordinates for the vertices of the model (see influence of the red vertex) binds the cage to the model. As a result, each cage vertex influences a nearby part of the model and users can intuitively pose the model.

In general, the cage-based deformation workflow follows three steps (see fig. 2.7). First, the cage $\mathcal{C}$ for the geometry $\mathcal{M}$ must be constructed, which can be a laborious task. To bind the cage $\mathcal{C}$ to the geometry $\mathcal{M}$ for deformation control, cage-based deformation methods commonly use generalized barycentric coordinates. Altering the positions of the cage vertices $\mathbf{c}$ relocates the vertices $\mathbf{v}$ of the geometry $\mathcal{M}$, while its connectivity remains unchanged.

2.3.1. Matrix Notation for Cage-based Deformation

As cage-based deformation uses linear affine mappings, this thesis introduces a common matrix notation for the available methods. A cage $\mathcal{C}$ is a manifold surface mesh consisting of vertices $\mathbf{c}_i \in \mathbb{R}^3$, $i = 1, \ldots, N_\mathbf{C}$ and polygons f_i, $i = 1, \ldots, N_\mathbf{F}$. The matrix $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_{N_\mathbf{C}})^\mathsf{T} \in \mathbb{R}^{N_\mathbf{C} \times 3}$ includes all the vertices of $\mathcal{C}$. The polygonal faces $f_i \in \mathbb{N} \times \ldots \times \mathbb{N}$ include integers of cage vertices into $\mathbf{C}$, form the boundary of the cage $\partial\mathcal{C} \ni f_i$ and define the interior space of the cage $\operatorname{Int}\mathcal{C}$, where $\partial\mathcal{C} \cap \operatorname{Int}\mathcal{C} = \varnothing$. If f is a triangle or quadrilateral (quad), this thesis denotes it as t or q, respectively. The mesh $\mathcal{M}$ is the to-be-deformed geometry. As cage-based deformation can be applied to several mesh types, the notation should cover surface as well as volume meshes. Thus, $\mathcal{M}$ consists of geometrical primitives $\mathcal{P}$ and vertices $\mathbf{v}_i \in \mathbb{R}^3$, $i = 1, \ldots, N_\mathbf{V}$. The matrix $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_{N_\mathbf{V}})^\mathsf{T} \in \mathbb{R}^{N_\mathbf{V} \times 3}$ contains all the vertices of $\mathcal{M}$. The prime symbol indicates deformed vertices $\mathbf{v}'$. Deformations are limited to a certain domain Ω. Typically, it holds that $\Omega = \mathcal{C}$ but some deformation methods allow for extrapolation from the cage. A point in the domain is denoted as $\mathbf{x} \in \Omega$.

2.3.2. Generalized Barycentric Coordinates

Since Möbius [Möb27] first formulated barycentric coordinates, the concepts of barycentric coordinates and interpolation have been widely used and extended. Applications of such generalized barycentric coordinates (GBC) are manifold including color interpolation [Mey+02], Gouraud and Phong shading [HF06], rendering of quadrilaterals [HT04], texture mapping [DMA02], texture synthesis [Tak+10], image warping [HF06; War+07; SHF13; MT23], image composition [Far+09], mesh parameterization [Flo97], shape deformation [Jos+07; JSW05; LS08; Lip+07; WG10], deformation transfer [BWG09], gradient mesh simplification [LJH13], generalized Bézier surfaces [LS07; LD89], surface design [SV18], and finite element applications [AO06; MP07; SM06; TS08; WBG07]. While this thesis focuses on the essentials for cage-based deformation, the reader is referred to the survey paper by Floater [Flo15] and the book by Hormann and Sukumar [HS17] for an in-depth overview on GBC.

Early work on GBC focused on 2D and many constructions have been proposed over the last 50 years. *Wachspress coordinates* [Wac75] are rational functions that can be expressed with a simple closed form [Mey+02], but they are not well-defined for arbitrary simple polygons. Likewise, *discrete harmonic coordinates* [PP93; Eck+95] are not well-defined on finite elements, while they are constructed by a piecewise approximation of the Laplace equation. Floater [Flo03] construct *mean value coordinates*, which also have a simple closed form and are well-defined for any concave polygon [HF06]. They are positive within the kernel of star-shaped polygons, possess the Lagrange property, and are smooth everywhere in the plane, except for polygon vertices at C^0-continuities. Alternative closed-form GBC for concave polygons are *metric* [MLD05; SM06], *moving least squares* [MS10], *Poisson* [LH13], *cubic mean value* [LJH13], and *Gordon–Wixom* [GW74; Bel06]. There even exists a whole family of coordinates that are well-defined for degenerate polygons [YS19], but all these constructions can be negative inside the domain. Positive coordinates for arbitrary concave polygons include *positive mean value* [Lip+07], *positive Gordon–Wixom coordinates* [MLS11], and *power coordinates* [Bud+16], but they are not smooth. *Blended barycentric coordinates* [APH17] overcome this issue, but they require on an embedding triangulation. Non-negative coordinates with at least C^1-continuity for arbitrary polygons include *harmonic* [Jos+07], *maximum entropy* [HS08], *maximum likelihood* [CDH23], *positive and smooth Gordon–Wixom* [Wan+19], *iterative* [DCH20], and *local barycentric coordinates* [Zha+14; TDZ19], but they do not exhibit a closed form and must be approximated with a numerical method.

Some of the 2D constructions can be extended to 3D, which is required for cage-based deformation in 3D, and this thesis reviews these constructions in detail in section 6.4.7. GBC provide a set of functions $\lambda_i \colon \mathbb{R}^3 \to \mathbb{R}$, $i = 1, \ldots, N_\mathrm{C}$ to interpolate the interior of $\mathcal{C}$. With the use of these functions, one can express any point inside $\mathcal{C}$ as an affine sum of cage vertices:

$$\sum_{i=1}^{N_\mathrm{C}} \lambda_i(\mathbf{v})\mathbf{c}_i = \mathbf{v}, \quad \text{where} \quad \sum_{i=1}^{N_\mathrm{C}} \lambda_i(\mathbf{v}) = 1.$$

Cage-based deformation capitalizes on these interpolation functions, in order to determine the vertex positions of $\mathcal{M}$, whenever the user deforms $\mathcal{C}$. Thus, calculating the coefficients $\lambda_i(\mathbf{v}_j)$, $i = 1, \ldots, N_\mathrm{C}$ for every vertex $\mathbf{v}_j$, $j = 1, \ldots, N_\mathrm{V}$ of $\mathcal{M}$ is an essential pre-processing step of cage-based deformation. This stage is frequently denoted as bind time, because it binds the to-be-deformed model to $\mathcal{C}$. After bind time, the vertex positions of $\mathcal{M}$ can be adjusted to the deformed cage:

$$\sum_{i=1}^{N_\mathrm{C}} \lambda_i(\mathbf{v}_j)\mathbf{c}'_i = \mathbf{v}'_j. \tag{2.15}$$

For brevity, λ_{ij} denotes the coefficient $\lambda_i(\mathbf{v}_j)$. Several properties govern the resulting deformation quality. Non-negativity is an important property, because it guarantees that the deformed vertices are located inside $\mathcal{C}'$. In addition, the coefficients λ_{ij} should vary smoothly, in order to provide shape preservation. On the boundary of $\mathcal{C}$, the functions λ_i should be linear over the polygons f. In summary, high-quality cage-based deformation control is achieved, if the functions λ_i satisfy the following properties:

- **Reproduction:** $\sum_{i=1}^{N_\mathrm{C}} \lambda_i(\mathbf{x})\mathbf{c}_i = \mathbf{x}, \forall \mathbf{x} \in \Omega$
- **Partition of unity:** $\sum_{i=1}^{N_\mathrm{C}} \lambda_i(\mathbf{x}) = 1, \forall \mathbf{x} \in \Omega$
- **Non-negativity:** $\lambda_i(\mathbf{x}) \geq 0, \forall \mathbf{x} \in \Omega$
- **Smoothness:** $\lambda_i \in C^\infty, \forall i = 1, \ldots, N_\mathrm{C}$

- **Linearity on $\partial\mathcal{C}$:** λ_i is linear on each cage polygon f
- **Lagrange property:** $\lambda_i(\mathbf{c}_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta
- **Locality:** λ_i only influences regions nearby $\mathbf{c}_i$, i.e., $\lambda_i(\mathbf{x})$ vanishes if $\mathbf{x}$ is far away from $\mathbf{c}_i$

After a set of GBC λ_{ij} is determined, the user can pose the model by relocating the cage vertices, i.e., control handles. Let us suppose that $\mathbf{W}_{ij} = \lambda_{ij}$ so that $\mathbf{W} \in \mathbb{R}^{N_\mathrm{C} \times N_\mathrm{V}}$. Then, posing the model follows a simple deformation scheme:

$$\mathbf{V}' = \mathbf{W}^\mathsf{T}\mathbf{C}'. \tag{2.16}$$

Most deformation approaches using cages only rely on eq. (2.16) to pose a model in accordance to $\mathcal{C}$. However, if cages are combined with other control structures, other update schemes such as linear blend skinning are used.

2.3.3. Linear Blend Skinning

The linear blend skinning (LBS) method is typically used for deformation using skeletal control structures. In LBS, several transformations are applied at once as a “blend” to a point in space. Let us consider the coefficients λ_{ij} as interpolating weights. A set of N transformations is applied to a vertex $\mathbf{v}_j$ as follows:

$$\mathbf{v}'_j = \sum_{i=1}^{N} \lambda_{ij}(\mathbf{L}_i\mathbf{v}_j + \mathbf{t}_i) = \sum_{i=1}^{N} \lambda_{ij}\mathbf{T}_i \begin{pmatrix} \mathbf{v}_j \\ 1 \end{pmatrix}, \tag{2.17}$$

where $\mathbf{L}_i$ and $\mathbf{t}_i$ are the linear and translation parts of the transformation $\mathbf{T}_i$, respectively.

As Jacobson [Jac14] pointed out, cage-based deformation using GBC is a special case of LBS, if transformations applied to cage vertices are restricted to translations. Through re-arranging, eq. (2.15) for adjusting the model vertices to the deformed cage matches eq. (2.17):

$$\begin{aligned} \mathbf{v}'_j &= \sum_{i=1}^{N_\mathrm{C}} \lambda_{ij}\mathbf{c}'_i = \sum_{i=1}^{N_\mathrm{C}} \lambda_{ij}\mathbf{c}'_i + \lambda_{ij}\mathbf{c}_i - \lambda_{ij}\mathbf{c}_i = \sum_{i=1}^{N_\mathrm{C}} \mathbf{v}_j + \lambda_{ij}(\mathbf{c}'_i - \mathbf{c}_i) \\ &= \sum_{i=1}^{N_\mathrm{C}} \lambda_{ij}(\mathbf{I}\mathbf{v}_j + \mathbf{t}_i). \end{aligned}$$

With the use of LBS, a convenient update scheme for cage-based deformation can be obtained. When the coordinates λ_{ij} are determined after bind time, the LBS matrix $\mathbf{M} \in \mathbb{R}^{N_\mathrm{V} \times 4N_\mathrm{C}}$ can be assembled:

$$\mathbf{M} = \begin{bmatrix} \lambda_{11}(\mathbf{v}_1^\mathsf{T}, 1) & \dots & \lambda_{N_\mathrm{C}1}(\mathbf{v}_1^\mathsf{T}, 1) \\ \vdots & \ddots & \vdots \\ \lambda_{1N_\mathrm{V}}(\mathbf{v}_{N_\mathrm{V}}^\mathsf{T}, 1) & \dots & \lambda_{N_\mathrm{C}N_\mathrm{V}}(\mathbf{v}_{N_\mathrm{V}}^\mathsf{T}, 1) \end{bmatrix}.$$

Whenever the user deforms the cage $\mathcal{C}$ at pose time, it suffices to assemble the transformation matrix $\mathbf{T} = ((\mathbf{I}, \mathbf{t}_1)^\mathsf{T}, \dots, (\mathbf{I}, \mathbf{t}_{N_\mathrm{C}})^\mathsf{T})^\mathsf{T} \in \mathbb{R}^{4N_\mathrm{C} \times 3}$. The model deformation using LBS can then be calculated as a simple matrix product:

$$\mathbf{V}' = \mathbf{M}\mathbf{T}.$$

2.4. Direct Volume Rendering of Unstructured Grids

After a numerical method such as the FEM performed calculations on a volumetric mesh, users typically wish to explore the results on the boundary as well as the inner structures of a volumetric mesh. DVR allows for the visualization of the inner structures of a geometry. Today, many important visualization applications such a computer tomography or post-processing in rapid prototyping rely on DVR.

The most commonly used approach for DVR is ray marching (see fig. 2.8) that was introduced by Drebin et al. [DCH88] for structured voxel grids in 1988. Like in the well-known raytracing technique, the camera emits rays through the scene to render a 2D image. The key aspect of ray marching to visualize inner structures is tracing rays within the volume of an object. This allows the interpolation of scalar values that are discretely defined inside the object. The ray marching technique performs sampling at discrete spatial points along the view rays. For regular voxel grids, it is sufficient to sample the scalar field at all the voxels intersecting a ray, whereas unstructured grids such as unstructured tetrahedral meshes (cf. section 2.1.1) require more sophisticated algorithms and an acceleration structure to quickly confine the set of potentially intersecting primitives.

For unstructured grids, DVR places sampling points along view rays at a certain sampling rate that is either constant or adaptive to the scalar field. Typically, an interpolation scheme such as barycentric interpolation approximates a scalar value using the spatially discrete scalar field. A transfer function $\mathbf{t}\colon \mathbb{R} \to \mathbb{R}^4$ maps the interpolated scalar value to color (radience) $\mathbf{c}^{\text{rgba}} = (r, g, b, \alpha)^\mathsf{T}$ in red, green, blue and alpha (RGBA) format. Through specification of the transfer function, the user controls the visualization to highlight the scalar values of interest [Lju+16]. After computation of the transfer function, the resulting color value needs to be adapted to the sampling rate, because the larger the step size along the ray the more material of the object is pervaded resulting in a more opaque color. The adaption of the color value to the sampling rate is achieved with the use of alpha correction [Eng+04]:

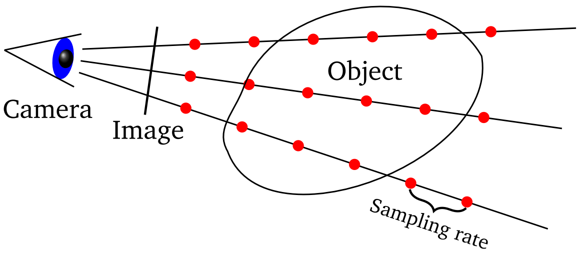


Figure 2.8.: In ray marching, the camera emits view rays through the object. The scalar field is sampled at sampling points (red) along the rays.

$$\tilde{\alpha} = 1 - (1 - \alpha)^{\frac{\Delta t}{\Delta t_{\text{min}}}},$$

where Δt is the current sampling rate and Δt_{min} is the minimal sampling rate used for computing the transfer function.

The application of alpha correction yields a corrected color value $\tilde{\mathbf{c}}^{\text{rgba}} = (r, g, b, \tilde{\alpha})^\mathsf{T}$ for a single sampling point on the ray. As DVR creates many sampling points for each ray, a scheme needs to merge the sampled color values along each ray to one color value. A color compositing scheme is the typical choice for this purpose. Typically, linear compositing based on the corrected alpha values is used [Kur11]:

$$\tilde{\mathbf{c}}^{\text{rgba}}_{\text{out}} = (1 - \tilde{\alpha})\tilde{\mathbf{c}}^{\text{rgba}}_{\text{in}} + \tilde{\alpha}\tilde{\mathbf{c}}^{\text{rgba}}_{\text{sample}},$$

where $\tilde{\mathbf{c}}^{\text{rgba}}_{\text{in}}$ is the incoming color obtained from the previous samples along the ray and $\tilde{\mathbf{c}}^{\text{rgba}}_{\text{sample}}$ is the current sample.

Typically, DVR emits one view ray per image pixel. The sampling rate governs the number of samples for each ray. Computation of the pixel colors amounts to interpolating the scalar field for each sample and compositing the corrected color values until the color becomes opaque or every sample has been processed.

Thus, the sampling rate substantially influences the rendering performance. For accurate rendering results, users benefit from a sampling rate. However, as a high sampling rate leads to many samples per frame, it leads to reduced run time performance. Thus, DVR imposes an inherent tradeoff between rendering quality and run time performance. In order to provide an interactive visualization, the DVR should be accelerated with present day GPUs. Since the computation of a color for a ray is independent from the other rays, parallelization of DVR typically processes rays simultaneously.

3. Related Work

This chapter highlights prior work on unstructured volumetric meshes, which is related to VP. As the term “mesh editing” is associated with many different geometry processing tasks, this chapter begins with a discussion of the scope of mesh editing in section 3.1. Optimizing and re-meshing a mesh for adaptation to numerical simulation is an important field of related work that is discussed in section 3.2. Related work on user-interactive editing operations of volumetric meshes appears in section 3.3. Interactive deformation of geometry has received a lot of attention in the literature, which is briefly discussed in section 3.4. Section 3.5 provides an overview on the available methods to create a cage for deformation control. The cage coordinate types for cage-based deformation are reviewed in section 3.6. A frequently discussed method to accelerate VP processes that performs the FEM on B-Reps is reviewed in section 3.7. Massively parallel post-processing applications for unstructured volumetric meshes require a spatial data structure for acceleration, wherefore the related work reviewed in section 3.8 addresses this topic. The DVR of unstructured volumetric meshes exhibits a long history in the literature that is presented in section 3.9.

3.1. The Two Scopes of Mesh Editing

Numerous mesh editing tools [Cig+08; Ble23; Pla23] enable user-interactive editing of surface meshes offering operations such as smoothing, repairing self-intersections, re-orientation of faces or deformation. Some tool provide more advanced modeling functionality such as the merging and subtraction of surface meshes [SS10a]. User-interactive modeling of geometry involves the orchestration of many mesh editing operations, because changing the shape of the mesh $\mathcal{M}$ typically requires the relocation of vertices V and re-meshing the triangulation T. Therefore, modeling the mesh $\mathcal{M}$ relies on the application of fundamental mesh editing operations such as gradient-based relocation of vertex positions, edge refinement, edge/face flips or edge collapses. While these fundamental mesh editing operations are typically not controlled by the user, these fundamental operations classify as mesh editing on a low-level of abstraction. Consequently, the term “mesh editing” can be associated with different operations. Which operations a user associates with the term “mesh editing” highly depends on the user’s perspective and prior work experience.

As the term “mesh editing” is quite general and therefore allows for many interpretations, this section draws the scope of mesh editing in the light of previous work. Jiang et al. [Jia+22] draw a wide scope defining editing of unstructured meshes as executing primitive operations that change the connectivity of a mesh. Following this definition, even fundamental meshing operations such as flips, element refinement or collapsing edges classify as mesh editing operations. Many important algorithms such as mesh generation, mesh adaptation or mesh improvement apply these fundamental operations, in order to produce a mesh that satisfies certain predetermined or user-specified criteria. VP frequently involves the application of these algorithms in the volumetric meshing step of the cycle. In particular, the editing of volumetric meshes frequently needs to apply re-meshing to ensure that the mesh still fulfills the requirements of numerical simulation. However, in the light of user-guided customization of geometry, the term “mesh editing” is rather associated with interactive editing of the mesh on a high level of abstraction, where the user does not care about the implementation details of meshing. Interactive mesh editing systems typically enable users to apply mesh editing operations locally in an intuitive way, e.g., using control structures (see section 3.4).

In view of the two different levels of abstraction, the following categories describe the two different scopes of mesh editing:

Low-level mesh editing: The low-level scope of mesh editing is an algorithmic point of view. An algorithm performs many primitive operations that change either the vertices V or elements T of the mesh $\mathcal{M}$ until certain criteria are fulfilled. These algorithms are the foundation of high-level mesh editing and their direct application is rather a task for domain experts.

High-level mesh editing: The user selects certain local parts of the mesh that represent a semantic meaning in the eye of the user. For this purpose, the user wishes a control structure or a convenient method to select semantic parts of the mesh. The user then specifies an editing operation on the selected parts of the mesh. The interaction with the mesh is on a high layer of abstraction and does not reveal the actual low-level meshing operations to the user.

This thesis addresses both scopes of mesh editing and presents massively parallel algorithms for low-level and high-level mesh editing. The low-level algorithms presented in this thesis focus on mesh optimization (see chapter 4) and re-meshing (see chapter 5). The related work on these topics appears in section 3.2. This thesis presents high-level volumetric mesh editing algorithms in chapter 6 and the related work in that field appears in section 3.3.

3.2. Mesh Optimization and Re-meshing of Unstructured Tetrahedral Meshes

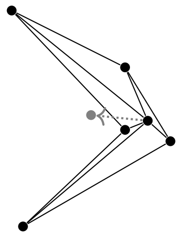
The literature comprises a long history in investigating the adaptation and optimization of unstructured tetrahedral meshes. The earliest optimization methods performed Laplacian smoothing along with re-meshing, but were superseded by optimization of pre-determined element quality metrics. Section 3.2.1 briefly describes this development. Further research revealed the potential of distortion energy metrics, which are described in section 3.2.2. The operations to improve the quality of a mesh can be categorized into two different groups [KS07]:

1. **Vertex Relocation:** Vertices are relocated to improve element quality but the connectivity of the mesh remains unaltered
2. **Re-meshing:** While the coordinates of the vertices remain unchanged, the connectivity of the mesh is altered, in order to improve the quality

As this thesis contributes massively parallel algorithms for mesh optimization, sections 3.2.5 and 3.2.6 review vertex relocation and re-meshing, respectively. This thesis also addresses boundary treatment of mesh optimization and highlights related work about boundary treatment in section 3.2.7.

3.2.1. From Laplacian Smoothing to Optimizing Element Quality Metrics

Many mesh optimization methods use computational efficient Laplacian smoothing, relocating a vertex in the direction of the arithmetic average of the adjacent vertices [FO97; SV03; Nea+06; Sha+16; Xi+21]. However, Laplacian smoothing does not strictly guarantee to produce a high(er)-quality or even an inversion-free mesh. When the local vicinity of a vertex forms a concave region, Laplacian smoothing can reposition the vertex outside of this region (see inset), which leads to inversions and a tangled mesh. In addition, the gradient of the Laplacian does not vanish in general, which complicates finding appropriate termination criteria.



Because of the shortcomings of Laplacian smoothing, previous work devised many different differentiable quality metrics for an element of a mesh [She02b]. The key benefit of these quality metrics is that their gradient allows for optimization using the method of steepest descent. Through consecutive relocation in the direction of steepest descent, i.e. the inverted gradient, a local optimum for the mesh quality can be found. Typically, element quality metrics evaluate a term that includes the determinant of an element's Jacobian in the denominator. As the Jacobian of an element is related to its signed volume/area (cf. section 2.1.1), the gradient of the element quality metric prevents the inversion of elements. Knupp [Knu00] confirms the use of the Jacobian as a building block for quality metrics of a finite element. Today, the element quality metrics based on the Jacobian of an element are typically used in the context of the FEM [Sor+23]. Therefore, many mesh optimization methods rely on distortion energies using the Jacobian. The subsequent section 3.2.2 discusses these distortion energies.

3.2.2. Distortion Energies for Mesh Optimization

Besides mesh improvement, energies minimizing global distortion are typically used for parameterization tasks such as surface fitting or re-meshing. Hormann and Greiner [HG00] introduce the most isometric parameterizations (MIPS). Originally, MIPS is intended for mapping a triangulation of data points to a triangulation in the plane. As this mapping can result in needle-like triangles of low shape quality, MIPS provides an energy of distortion, which can be minimized. In order to obtain triangles of good shape, the MIPS energy measures the distortion of a given triangle from a reference ideal triangle. Since the Jacobian can be used to express the deviation of a given element to an ideal element [Knu00], the MIPS energy $\mathcal{D}^{\mathrm{MIPS}}$ is computed using the Frobenius norm $\|\cdot\|_{\mathrm{F}}$ of the Jacobian:

$$\mathcal{D}^{\mathrm{MIPS}}(\tau) = \|\mathbf{J}_\tau\|_{\mathrm{F}}\|\mathbf{J}_\tau^{-1}\|_{\mathrm{F}} = \frac{\mathrm{tr}(\mathbf{J}_\tau^{\mathsf{T}}\mathbf{J}_\tau)}{\det(\mathbf{J}_\tau)}.$$

The MIPS energy is invariant to translation, orthogonal transformation and scaling. In addition, MIPS strictly penalizes elements of near-zero volume, i.e., infinitesimally small volume, because its denominator consists of the determinant of the Jacobian.

For support of a broader set of geometry processing tasks, Fu et al. [FLG15] extend MIPS to the advanced MIPS (AMIPS) conformal energy $\mathcal{D}^{\mathrm{AMIPS}}$ that effectively measures distortion in 2D and 3D:

$$\mathcal{D}^{\mathrm{AMIPS}}(\tau) = \frac{\mathrm{tr}(\mathbf{J}_\tau^{\mathsf{T}}\mathbf{J}_\tau)}{\det(\mathbf{J}_\tau)^{2/d}}, \qquad \text{where } d \in \{2,3\}.$$

For vertex relocation, they perform non-linear Gauss-Seidel iterations simultaneously on sets of non-adjacent vertices. However, non-linear optimization methods typically impose slow run times and do not scale well to meshes with many elements. For this reason, Rabinovich et al. [Rab+17] present a local/global algorithm that scales to large data sets through replacing the non-linear energy with a simple proxy energy. The local step calculates weights mapping gradients to the distortion of elements using the proxy energy. With the weighted gradients, a global system can be efficiently assembled and solved. For solving the global system, an initial inversion-free step size is found using the method of Smith and Schaefer [SS15].

As demonstrated by Hu et al. [Hu+18; Hu+20], the conformal AMIPS energy is effective in improving ill-shaped elements On the contrary, harmonic triangulations provide a local order of bistellar flips with a relationship to Delaunay flips (cf. section 2.1.3). As flips are locally ordered by energy reduction, it is possible to formulate a massively parallel algorithm performing locally most beneficial flips that quickly improves element quality (see section 4.4). Additionally, this thesis focuses on Delaunay-based methods, as harmonic flips are related to Delaunay flips. The goal of this thesis is to achieve scalability of Delaunay-based methods by proper parallelization.

It is worth to discuss the differences and commonalities of $\mathcal{D}^{\text{AMIPS}}$ and η in terms of evaluating element shape quality. Both $\mathcal{D}^{\text{AMIPS}}$ and η penalize degenerate elements, because for both energies $\det(\mathbf{J})$ appears in the denominator [Knu00]. While for η the denominator scales linearly with $\det \mathbf{J}$, the denominator of $\mathcal{D}^{\text{AMIPS}}$ is a cubic root function of $\det(\mathbf{J})$, which means that large volumes minimize the energy $\mathcal{D}^{\text{AMIPS}}$ less but small volumes lead to more penelization. Thus, both energies avoid element inversion, i.e. sign flip in the volume of an element, while $\mathcal{D}^{\text{AMIPS}}$ approaches infinity faster due to stricter penelization of low volumes. Both energies are undefined for $\det(\mathbf{J}) = 0$ and both energies are minimal for the equilateral tetrahedron. As an important difference, the $\mathcal{D}^{\text{AMIPS}}$ energy is scale invariant but the harmonic index η scales with $\det(\mathbf{J})$.

3.2.3. Edge Collapse in Tetrahedral Meshes

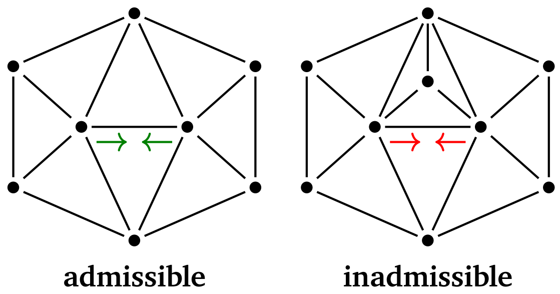


Figure 3.1.: Collapsing is inadmissible if the set intersection of the two one rings of edge vertices includes simplices that do not contain the to-be-collapsed edge.

Pioneering works about coarsening triangle meshes such as the progressive meshes by Hoppe [Hop96] inspired many publications about the coarsening of tetrahedral meshes. Staadt and Gross [SG98] explore the use of edge collapsing for progressive tetrahedralizations and present various geometric checks necessary for preserving consistency. They check the oriented volumes of simplices to prevent degeneracies and inverted elements. In addition, they check for intersections when collapsing boundary edges to prevent self intersections. Dey et al. [Dey+99] detail the preconditions for collapsing edges under which they preserve the topological type of simplicial complexes up to the third dimension. They show that only collapse operations that satisfy the link condition (see fig. 3.1) are admissible. As an alternative to collapsing edges, Chopra and Meyer [CM02] replace one tetrahedron with one vertex to rapidly reduce the count of tetrahedra. This approach is only efficient for interior tetrahedra, because avoiding intersections at the boundary is computationally expensive. Kraus and Ertl [KE03] present a solution for collapsing boundary edges in a non-convex tetrahedral mesh. To prevent self-intersections they first convexify the mesh computing the convex hull and subsequently check for inversions, whenever a boundary edge of the non-convex mesh is collapsed within the convex hull. As tetrahedral re-meshing involves a variety of operators, Loseille and Menier [LM14] propose a cavity-based re-meshing operator that embeds collapsing, refinement and face/edge swaps. While the work of Loseille and Menier [LM14] addresses coarse-grained parallelization, the conflict detection in section 5.1.3 is for fine-grained parallelization.

3.2.4. Applications of Tetrahedral Mesh Coarsening

In order to reduce workloads for scientific visualization, Cignoni et al. [Cig+00] simplify volume data collapsing edges. They involve an error evaluation of scalar data attached to the tetrahedra to obtain accurate visualizations. Similarly, Natarajan and Edelsbrunner [NE04] reduce workloads for visualizing large datasets that represent density functions. They use a modification of the quadric error measure of Garland and Heckbert [GH97] to prioritize collapse operations so that the density function is preserved while improving element quality. Many optimization methods for tetrahedral meshes collapse edges, because it removes low-quality elements [Mis+09]. Tetrahedral mesh generators such as Tetgen [Si20] supply mesh coarsening to adhere to a specified sizing function. A sizing function enables users to govern the size of elements. As each tetrahedral element imposes computational cost for simulations, Cutler et al. [CDM04] collapse edges in a mesh to reduce the number of elements, while removing the most low-quality elements. In order to control the accuracy of numerical methods, the adaptation of unstructured grids frequently performs coarsening besides refinement [Ala+06]. Over the years, many tools for mesh adaptation emerged [Com+09; Par22; Iba22]. The unstructured grid adaptation working group presents a concise overview and qualitative benchmark of many of these tools [Iba+17].

3.2.5. Parallel Vertex Relocation

Many publications address parallel vertex relocation for tetrahedral mesh optimization. For fast run time performance, Freitag et al. [FJP99] relocate batches of non-adjacent vertices in parallel, while preventing element inversions. Parallel strategies such as the graph coloring by Deveci et al. [Dev+16] quickly obtain a high-quality partition of the vertices into independent sets.

Benítez et al. [Ben+18] present an algorithm for smoothing and untangling meshes in a distributed environment using domain decomposition. While domain decomposition results in coarse parallelism, this thesis focuses on fine-grained parallelism that can be used in single machines and does not require a distributed system for fast run times. Zint and Grosso [ZG19] describe a GPU-parallel algorithm that searches for an optimal vertex position on a coarse grid of candidate positions within the vertex' one-ring neighborhood. While this allows for optimization of non-differentiable functions, the proposed optimization method relies on differentiable functions, as they allow the use of first-order optimization methods that converge more quickly than exhaustive search. Shontz et al. [SVH20] relocate vertices by solving a set of ordinary differential equations on a distributed system using domain partitioning. We focus on massively parallel algorithms that do not require a distributed system. Xi et al. [Xi+21] perform adaptive Laplacian smoothing massively parallel on the GPU using the method of Xiao et al. [Xia+19]. In order to prevent write conflicts, they perform a composition of Jacobi iterations and Gauss-Seidel iterations, where active threads access updated values for vertices processed by previous parallel passes and old values for still to be optimized vertices. While they focus on Laplacian smoothing, the proposed element quality optimization performs optimization-based vertex relocation.

3.2.6. Parallel Re-meshing

In contrast to parallel vertex relocation, parallel local reconnection of vertices imposes the additional challenge of preventing concurrent processing of overlapping regions. Nonetheless, vertex relocation and reconnection should be used in concert [KS07] to achieve an effective optimization.

De Cougny and Shephard [DS99] present parallel refinement and coarsening for distributed environments. The mesh is partitioned in domains and each processor performs adaptation on its assigned domain. Refinement of edges is based on subdivision patterns. Faces of adjacent tetrahedra are triangulated equally to ensure a valid mesh. Collapsing of edges is parallelized among mesh domains. Synchronization is only necessary at the domain boundaries.

As distributed systems orchestrate many machines, sophisticated methods for parallel re-meshing on a single machine emerged. DeCoro and Tatarchuk [DT07] decimate polygonal meshes on the GPU using the geometry shader stage of the graphics pipeline. They perform prior vertex-clustering and use the rendering pipeline to cull the triangles that become degenerate due to repositioning vertices. Instead of decimating polygonal meshes using the rendering pipeline, the re-meshing method presented in chapter 5 focuses on the coarsening of tetrahedral meshes. D'Amato and Vénere [DV13] present a CPU-GPU framework for parallel element shape optimization determining independent clusters around the worst quality elements. The re-meshing method presented in chapter 5 repeatedly determines dense sub-meshes forming the cavity of an edge. As these sub-meshes can be adjacent to each other while they are not overlapping, they admit efficient parallelization favoring edges with lowest cost according to a specified cost function. Papageorgiou and Platis

[PP14] present a parallel algorithm for collapsing edges of triangular meshes using the GPU. While their work is optimized for manifold surface meshes, this thesis focuses on unstructured tetrahedral meshes. Additionally, their conflict detection requires determination and sorting of super-independent vertices, whereas this thesis presents conflict detection that prioritizes by a cost function such as element quality. Loseille et al. [LMA15] propose a parallel re-meshing algorithm using the cavity-based operator by Loseille and Menier [LM14]. They achieve coarse-grained parallelism using domain decomposition to obtain a set of conflict-free cavities, whereas this thesis focuses on fine-grained parallelism for GPUs.

In the further course of history, academics devised methods to produce more dense partitions of the mesh to improve parallelism. Shang et al. [Sha+16] present a multi-threaded algorithm for parallel local reconnection, which maps re-meshing operations to feature points sorted along a space filling curve. They assume geometrical separation of re-meshing operations so that regions rarely overlap, whereas the algorithms contributed by this thesis do not rely on the assumption of geometrical separation but handle conflicts on the fly. Ibanez and Shephard [IS16] schedule the application of cavity-based re-meshing on shared memory systems. Their method finds independent cavities for parallel processing. Their determination of independent cavities constructs a graph of adjacent mesh elements and calculates the worst qualities in the cavity. They perform a conflict resolution by iteratively adding local maximas to the independent sets until no more local maxima can be found or added. The conflict detection presented in section 5.1 does not require the construction of conflict graphs. In addition, the conflict detection presented in section 5.1 finds independent sub-meshes in two parallel passes, while the required iterations for the method of Ibanez and Shephard [IS16] is only bounded by the longest path in the conflict graph. Drakopoulos et al. [DTC19] describe a parallel speculative local re-meshing approach for HPC. They use atomic operations for synchronization in case of overlapping regions. In contrast to established parallel local reconnection methods, the parallel re-meshing algorithms of this thesis do not require a precomputed decomposition of the mesh or atomic operations but rely on the local order of a priority function such as element quality.

The industry and academics invested a lot of effort into re-meshing using even more fine-grained sub-meshes, which is more apt to re-meshing on modern accelerators such as GPUs. Chen et al. [CTO20] detail the design of GPU-parallel Delaunay refinement. They show how to concurrently insert Steiner points with high occupancy among GPU threads. Jiang et al. [Jia+22] present a high-level abstraction approach to specify mesh editing algorithms using a framework that provides CPU-parallel implementations of low-level mesh editing operations. Their approach schedules operations with a shared memory locking mechanism. Conflicts are avoided by domain decomposition and locking mutexes on the two-ring neighborhood of edges. The re-meshing algorithms of this thesis do not require locking mechanisms or domain decomposition, resulting in more compact sub-meshes. Moreover, Jiang et al. [Jia+22] present a parallel variant of constructing harmonic triangulations (cf. section 2.1.3) that requires a large number of cores in a CPU, in order to outperform the sequential variant. As such CPUs are typically used in HPC-clusters, the usage of the GPU seems attractive for parallelization on a single machine. In addition, their parallel harmonic triangulation algorithm does not provide boundary preservation for optimizing vertex positions, while this thesis addresses boundary preservation for gradient descent.

Recently, Gautron et al. [GKN23] present an algorithm for GPU-parallel edge collapsing for triangular surface meshes. Their algorithm packs the cost and the index of an edge in an edge descriptor. They propagate the minimal edge descriptor over the one ring of simplices for each of the two edge vertices. With the use of atomic operations, they prevent conflicts that arise due to massively parallel execution. Their algorithm is part of NVIDIA's Micro-Mesh toolkit [NVI23b]. While their algorithm is designed for triangular meshes, it extends to tetrahedral meshes.

3.2.7. Boundary Treatment in Mesh Optimization

Boundary treatment in tetrahedral mesh optimization is a sparsely discussed field. While some methods rely on curved boundaries [Das+18], the optimization algorithm presented in chapter 4 only relies on the boundary of the discrete mesh. Many methods either subdivide ill-shaped boundary elements [KS07] or reproject boundary vertices back on the original surface [Ale19]. Subdivision of boundary elements increases the element count, which is a drawback, as each element costs computationally. The drawbacks of boundary reprojection is that it requires to find the closest point on the boundary and the reprojection step does not respect energy minimization leading to reduced convergence.

Yin and Teodosiu [YT08] replace reprojection of boundary vertices with shape functions approximating the surface based on the discrete mesh. They incorporate the shape functions as a penalty term into the to-be-optimized function to enforce boundary conformance. Contrary to the optimization algorithms in chapter 4, the penalization approach requires the choice of a suitable penalty number. Wicke et al. [Wic+10] address optimization of the mesh boundary for dynamic domain re-meshing. They penalize relocation of boundary vertices by augmenting the optimization function with a quadric error term. Although this allows for efficient relocation of boundary vertices, element quality to surface distance is an apples-to-oranges comparison. Xu et al. [Xu+09] propose harmonic guided optimization to further improve the quality of boundary elements despite the usage of a quadric error term. They pre-compute a harmonic scalar field on a voxelized grid. As the field is maximal at the boundary and minimal for the medial axis of the mesh, it enables the computation of weights tweaking the importance of boundary preservation and element quality. The optimization algorithm in chapter 4 keeps boundary vertices on the surface without using a penalization term, and thereby without the need of pre-computing additional weights.

As mesh optimization at the boundary oftentimes alters geometric features as well as the volume of the geometry, Jiao [Jia06] presents feature detection by eigenvalue decomposition of the quadric metric tensor and volume conservation by application of null space smoothing. In null space smoothing vertices move along a certain subspace. Jiao [Jia06] employs null space smoothing and iteratively solves a nonlinear equation system that pose the problem of minimal volume loss. While chapter 4 presents an approach for gradient-descent along the surface instead of error-metric guided preservation, the feature detection of Jiao [Jia06] is a part of the face group extraction presented in section 6.1.2.

3.3. High-level Volumetric Mesh Editing

Many authors addressed the high-level editing of volumetric meshes. Oftentimes, these authors have the shortening of design iterations in mind. This section covers two different methods of interactive mesh editing. The editing on the basis of user-selected semantic features is discussed in section 3.3.1. Deformation-based approaches appear in section 3.3.2.

3.3.1. Volumetric Mesh Editing on the Basis of Semantic Features

Some authors have addressed the editing of meshes with the use of semantic features. This section presents related work that resembles the editing approach described in section 6.2.

A related editing framework in the context of engineering was presented by Serna et al. [SSF10], who propose an embodiment of the FEM mesh for modification. They propose a data structure that stores and dynamically updates neighborhood information of elements so that modeling operations can be easily implemented. While they present a data structure for mesh editing on the CPU, this thesis focuses on massively parallel algorithms for mesh editing. Another tetrahedral mesh editing framework for FEA was presented by Xian et al. [XGZ11]. Their framework decomposes the model into local surface features representing the semantic meaning of particular components. Like the editing operations in section 6.2, their framework preserves element quality throughout model modification. In contrast to their framework, our system performs GPU-accelerated editing operations. Graphite [Inr24] allows users to modify tetrahedral meshes, e.g., filling holes or deleting individual facets. Graphite provides an automatic hole detection that is controlled by a user-specified number of maximum boundary loop vertices. While users work on individual mesh facets in Graphite, this thesis focuses on high-level editing with face groups (see section 6.1) to accelerate VP.

Attene et al. [Att+08] present a hierarchical clustering algorithm for tetrahedral meshes to enable intuitive selection in editing environments. The algorithm clusters single tetrahedra into approximately convex regions using a concavity-based cost function. Based on these regions, Attene et al. [Att+08] detail user-guided editing operations that allow for mesh deformations, copying and pasting meshes, and removing mesh regions. While Attene et al. [Att+08] focus on mesh segmentation and interaction, they do not couple mesh editing with simulation applications. Additionally, they do not address acceleration through massively-parallel computing.

3.3.2. Volumetric Mesh Editing by Shape Deformation

Various approaches enable the deformation of unstructured tetrahedral meshes by user-specified shape deformation.

Stoll et al. [SATS07] present a method for interactive deformation of a tetrahedral mesh. For an input high-resolution surface mesh, they first simplify the surface mesh and subsequently apply Delaunay meshing (cf. section 2.1.2) to generate a coarse tetrahedral mesh. Similarly to cage-based deformation, they calculate coefficients that allow to express the vertices of the coarse tetrahedral mesh as linear combinations of the input surface mesh. As a result, the user can specify handles for interactive mesh deformation. After deformation, they reconstruct the deformed version of the input mesh from the deformed tetrahedral mesh. While Stoll et al. [SATS07] focus on the deformation of surface meshes and obtain only a coarse tetrahedralization of the surface, this thesis focuses on editing unstructured tetrahedral meshes without coarsening their surfaces altering the input shape.

In order to deform unstructured tetrahedral meshes for accelerating VP processes, many different morphing methods [Sta+11; SMB13; Por+21] were presented. Typically, these morphing methods rely on a parametrization of the geometric model of the prototype. By changing one parameter of the model, the morphing method interpolates the mesh vertex positions between the model shapes corresponding to the original and newly specified parameters. The mesh connectivity is typically only changed, when the element quality falls below a specific threshold or an element inversion occurs. Therefore, morphing can benefit from the massively parallel optimization algorithms presented in this thesis. In section 3.6 this thesis explores the use of interactive and control handle-driven cage-based deformation for deforming unstructured tetrahedral meshes, which in contrast to morphing does not depend on a parametrization of the model.

Frank [Fra06] addresses the visualization and modeling of unstructured tetrahedral meshes for geological applications such as reservoir simulations. The modeling functionality covers boolean operations on the basis of implicit iso-values, local mesh refinement, surface reconstruction and iso-value surface reconstruction. While Frank's [Fra06] tetrahedral modeling framework relies on iso-value based implicit modeling, the editing functionality presented in this thesis does not require iso-value data or surface interpolation. The visualization functionality covers iso-value surface interpolation and multitexturing for the boolean operations. On the contrary, this thesis covers volume rendering of unstructured meshes in chapter 7, which is not limited to the surface but enables exploration of inner structure. Moreover, this thesis comprises modeling for unstructured meshes for VP applications and does not focus on geological modeling.

In order to shorten VP cycles, Xian et al. [XZG13] present cage-based deformation (cf. section 2.3) of meshes for FEM using a mapping of semantic features to cage vertices. If the user intends to alter a semantic feature, the user can select the corresponding cage vertices for relocation. In contrast to this thesis, Xian et al. [XZG13] do not address massively parallel processing for cage-based deformation. In addition, they do not focus on volumetric meshes specifically and do not address issues such as mesh quality.

3.4. Interactive Mesh Deformation for Model Manipulation

For interactive shape deformation, this thesis investigates the capabilities of cage-based deformation. Besides cage-based deformation, other deformation methods using control points to manipulate geometry have been presented. This section highlights these methods and their advantages as well as disadvantages compared to cage-based deformation (see section 2.3).

3.4.1. Free-form Deformation using Lattices

One such method is free-form deformation [SP86]. The deformation domain Ω is defined by a volumetric control mesh denoted as control volume, which enables geometry manipulation through relocating its control points. Like in cage-based deformation, the vertices of $\mathcal{M}$ are expressed as an affine sum of the control points. The weights of the sum are defined by the location of the vertex in the parameter space of the lattice geometry. Due to the expression of the vertices in the parameter space, the computation of the updated vertices is equal to the evaluation of the lattice geometry at predefined parameter values. However, mapping a point in Euclidean space to the parameter space requires a known mapping between the spaces [SP86; MJ96].

Various geometry representations have been shown to be suitable as control volumes, such as tensor product trivariate Bernstein polynomials by Sederberg and Parry [SP86], trivariate B-Splines by Griessmair and Purgathofer [GP89], and Catmull-Clark (CC) volumes by MacCracken and Joy [MJ96]. Depending on the geometry representation, algebraic, numerical, or approximating methods may be best suited [SP86; MJ96] to find the parameter values of each vertex. As the deformation is based on the re-evaluation of the control volume, the mathematical properties of the deformation, such as continuity and locality, are defined by the basis functions of the chosen geometry representation. Furthermore, the deformation domain may only include parts of the model. This allows for the definition of multiple control volumes on a model to achieve local deformations using varying degrees of control [SP86].

Similar to cage-based methods, free-form deformation techniques use a mesh for deformation control, which contains the to-be-deformed geometry. However, the control structure is inherently volumetric and typically imposes additional restrictions on mesh connectivity, which complicates the construction of a viable control volume. Cages are more simple to handle, as they are less restrictive.

3.4.2. Skeletal-based Deformation

While cages are good control structures for detailed shape manipulation, skeletal-based deformation introduced by Magnenat-Thalmann et al. [MLT88] excel at expressing character motion. The user either generates the skeleton manually or automatically [LW07; VF09]. In an intuitive way, the user configures the joints of the skeleton to specify the character's pose [CHP89]. Whenever the joint configuration changes, the surface mesh, i.e., the character's skin, is deformed accordingly. The coupled use of skeletons and cages allows for motion expression with the skeleton, while details can be modeled with the cage [Cor+20].

Deformation of the skin can be efficiently implemented on a GPU for real-time interaction with the application of LBS (see section 2.3.3), where transformations are associated with each joint. Although LBS is frequently used due to its simplicity and efficiency, it can produce artifacts, e.g., candy wrapper artifacts, which can be prevented by using an advanced skinning scheme such as dual quaternions [Kav+08]. Another method to avoid these artifacts is the animation space by Merry et al. [MMG06], which is a larger family of linear deformation methods with many benefits such as improved fitting to example poses and advanced distance computations. As the computation of high-quality skinning weights can be expensive, Wang and Solomon [WS21] recently presented quasi-harmonic weights for near real-time computation.

3.4.3. Linear Subspaces

As the generation and re-meshing of cages for deformation control is tedious, Wang et al. [Wan+15] efficiently calculate linear subspaces in Ω allowing users to interactively define point and region handles without using cages. A point handle is represented by a point in space, whereas a region handle is a manipulator object with control vertices to deform a selected subdomain of Ω, where the undeformed parts of Ω are blended smoothly [BK04]. Deformation by handle control uses a set of weights that allow for deformation using LBS (see section 2.3.3). The weights are computed numerically by discretizing Ω with an embedding volumetric mesh $\mathcal{E}$.

For smooth deformation, computation of the weights minimizes a squared Laplacian energy. As the fairness component of this energy contains $\mathbf{1}$ in its null space, minimization can be implemented efficiently by solving a sparse linear system with a right-hand side for each handle. Due to the efficiency of weight calculation, users are able to add or remove handles at interactive rates and deform the model applying handle translations and rotations, albeit at the cost of negative weights and the need to generate $\mathcal{E}$.

3.4.4. Radial-Basis-Function-based Deformation

In contrast to the mesh-based definition of the deformation space, the deformation space may be manipulated by arbitrary, unconnected control points. Each control point is assigned a radial basis function, which defines the influence of the control point by means of its distance to the vertices. The influence reduces with increasing distance from the control point. This approach allows for accurate control over the deformation locality and poses no restriction on the placement of the control points. The choice of the specific radial basis function (RBF) determines the deformation behavior when manipulating a control point.

The specific coefficients of each RBF may be found automatically by solving a linear system [BSB07; Por+21]. The deformation of the model is computed by the interpolation defined by the previously specified RBFs. The interpolation computes multiple scalar fields, containing the coordinates of the deformed mesh. Each scalar field interpolates a component of the deformed mesh [BSB07].

Due to the universal nature of the RBF interpolation, the same deformation may be applied to multiple meshes and may extrapolate or interpolate [Por+21]. However, RBF-based techniques only offer point handles for deformation control. Contrary to offering only point handles, cage-based deformation enables users to define the local influence of handles based on the topology of the cage.

3.5. Generation of Cages for Deformation Control

While the shape deformation methods presented in the previous section 3.4 rely on interaction with point handles, skeleton joints or lattices, cage-based deformation relies on an enclosing polygonal mesh, i.e., a cage, for deformation control (cf. section 2.3). Enclosing meshes are needed for many different applications [MCA15]. This thesis focuses on cages for deformation control. As the polygonal mesh can adapt to surface details, the advantage of cage-based deformation is its ability to intuitively express high-resolution deformation relocating a part of the cage at once. For this reason, cage-based deformation is a versatile approach. Today, cage-based deformation is used in various fields including character animation [Ju+08; Cor+12; Kim+14], image deformation [Men+09], mesh modeling [TMB18; TB22], VP [AAN12; XZG13;

DJS16], 3D motion capture [SF11; TTB12], and recently even virtual environments [Sca+20; LLH22] and neural networks [Yif+20; Pen+22; XH22].

As the focus of this thesis lies on VP, a systematic comparison in chapter 6 evaluates the capabilities of cage-based deformation for VP. Before this detailed evaluation, this chapter presents the available methods to generate a cage and the available coordinates to achieve deformation control. Although cage-based deformation is primarily used for artistic applications, it can provide useful method to modify prototypes in VP processes. The easy-to-use deformation control can facilitate modification of meshes without using CAD, in order to shorten VP cycles. Moreover, smooth deformation properties and anisotropic stretch provide means for using cage-based deformation to modify organic or organic-like geometries, which are frequently used in VP applications such as 3D printing or shape optimization. However, as isotropic deformations can be difficult to guarantee with cage-based deformation, the use of cage-based deformation for modeling mechanical parts is probably suboptimal.

Before users can benefit from easy-to-use cage-based deformation, they need to obtain a cage that is suitable for the intended deformation. Therefore, the literature exhibits many methods to quickly generate a cage for deformation control. Laube and Umlauf [LU16] present a survey on the early automatic cage generation methods. The following provides a comprehensive discussion of generation methods to obtain cages for deformation control. First, section 3.5.1 gives an overview on the desirable properties of a cage for deformation control. Like in the previously published survey [Str+24], a systematic review organizes the available cage generation methods into four categories. Each category is discussed in a section. Section 3.5.2 discusses offset surface simplification methods. Section 3.5.3 discusses voxelization-based methods. Section 3.5.4 discusses template-based methods. Section 3.5.5 discusses interactive methods. As some cage coordinate types require an embedding tetrahedral mesh of the geometry and the cage (see section 3.6.2), section 3.5.6 comprises the process of creating a suitable embedding.

3.5.1. Cage Quality

Cages for interactive modeling should exhibit certain properties for convenient deformation control [Jac14]. Users wish to deform the model with only few control vertex relocations. Thus, the cage should include reasonably few control vertices for the user to manage. At the same time, the cage should wrap the model tightly with control vertices near the semantic parts of the model. A self-intersecting cage can lead to erroneous deformation results and should be avoided [SVJ15]. In addition, many coordinate types do not work if the cage intersects the model. In summary, the properties of a high-quality cage are:

- Reasonably low number of control vertices
- No self-intersections
- No intersections with the model, i.e., the cage is conservative
- Wrap the model tightly
- Control vertices should be close to the to-be-deformed parts of the model
- The cage should provide symmetric structures, where the model exhibits symmetric features

As these properties only loosely define the requirements for a high-quality cage, the quality of the cage needs to be evaluated in light of each use case. In order to allow for the comparison of cages, Xian et al. [XLX15] present a quality metric for cages:

$$E_{\mathcal{C}} = \left(1 - \frac{N_{\mathrm{C}}}{N_{\mathrm{V}}}\right) S(\mathcal{C}, \mathrm{T})\, e^{1-(\operatorname{vol}\mathcal{C}/\operatorname{vol}\mathrm{T})},$$

where vol represents the volume of a mesh and $S(\cdot, \cdot) \in [0, 1]$ represents a shape similarity measure, such as the similarity measure by Elad et al. [ETA02], which enables user-guided evaluation of shape similarity.

3.5.2. Offset Surface Simplification Methods

One of the early efforts to automatically generate an enclosing cage is the progressive hull generation by Sander et al. [San+00]. Building upon the progressive meshes by Hoppe [Hop96], the hull is generated by collapsing edges of the input mesh such that the coarser mesh includes every vertex of the input mesh. After a sequence of edge collapses, a coarse cage is obtained from the input mesh.

Shen et al. [SOS04] create an implicit surface enclosing an input polygon soup using a constrained least-squares formulation. They detail an iso-surface extraction method to ensure that all input points are enclosed by the extracted iso-surface.

For deformation transfer, Ben-Chen et al. [BWG09] construct an offset surface for an input model with repeated simplification until the number of cage faces is lower than a user-defined threshold. As the offset surface is formed by relocating vertices along the normals of the previous offset surface, some models require heterogeneous step sizes for computing offset positions to avoid artifacts.

To avoid self-intersections, Deng et al. [DLM11] decimate the input mesh, prioritizing edge collapses with an error metric, and perform a clean-up step. They incorporate a fidelity function for shape preservation and a function penalizing opposing face normals for triangle quality. Throughout decimation, they position vertices along surface normals to ensure an enveloping cage. Self-intersections are detected and removed by re-meshing the intersecting parts.

For editing meshes for VP purposes, Xian et al. [XZG13] present a cage generation method that maps semantic feature lines and faces to cage vertices. Their initial step is to detect feature lines and semantic relations using a detection approach such as iwires [Gal+09]. Subsequently, their cage generation method performs point sampling on the resulting feature lines and faces. As the sampling points form the cage vertices, a mapping of cage vertex to semantic line or face can be established during the sampling stage. A reconstruction algorithm [She+08] creates a surface mesh from the sampling points. The relocation of the sampling points towards a determined outward offset direction creates an enclosing cage.

Sacht et al. [SVJ15] generate nested cages in layers so that high-resolution cages tightly bound the model and coarser cages wrap the finer cages. Each cage layer is the result of the previous finer layer after passes of decimation, flow pushing the cage inside the previous layer and re-inflation until obtaining the bounding condition. The cage layers offer a nested collection of viable cages for modeling.

3.5.3. Voxelization-based Methods

Xian et al. [XLG09] voxelize the oriented bounding box (OBB) of the model to obtain a cage. The use of principal component analysis (PCA) quickly provides a tight OBB for the model [Dim+06]. After voxelization of the OBB, they calculate the max-norm distances [Var+03] of mesh triangles to voxel centers to determine the feature voxels intersecting the mesh surface. Extracting and triangulating the outer feature voxel faces yields a cage bounding the model. A final smoothing step relocates the cage vertices towards the model and achieves a smoothed tightly bounding cage.

To construct a coarse enclosing grid of highly detailed linear elastic deformable objects, Nesme et al. [Nes+09] organize an initial fine hexahedral embedding into an octree structure, where each hexahedron is associated with the local material properties of the model. The material properties of a parent voxel can be deduced from its children [NF+06]. As a result, a coarse bounding cage enables the quick retrieval of deformation properties at any desired level of the octree hierarchy.

As coarse cages are convenient for deformation control, Xian et al. [XLG11] automatically generate cages by optimizing an OBB tree with a slicing rule and subsequent mesh improvement. First, a voxelization of the model's OBB is decomposed into the voxel types feature, inside and outside. The mesh vertices and barycenters of inner voxels form a point set, whose PCA provides the root OBB of the tree. Recursively, each OBB is split according to a termination criterion considering local shape variation and OBB edge lengths. Each OBB split bisects the point set at its barycenter perpendicular to the longest edge and applies PCA to the two new point sets. Finally, boolean union operations merge the OBBs and mesh improvement ensures a high quality triangular cage.

For fast computation of coarse cages, Xian et al. [XLX15] present a voxelization-based region decomposition with subsequent simplification. The method relies on a voxel grid of the model's OBB. A scanline method categorizes the voxels into inside, surface, and outside. The use of flood filling divides the inside voxels into disconnected groups. Dilating the voxels of inner groups yields surface voxels that envelope the input model. The surface voxels dilating the inner voxels constitute the broad model parts, while the other surface voxels constitute the narrow parts. Determining the outside voxel faces of broad regions and the OBBs of narrow regions yields a set of partial cages, which forms a single cage through successive application of boolean operations. Finally, a simplification step merges co-planar voxel faces and collapses edges so that the cage still envelops the model.

3.5.4. Template-based Methods

To enable the reuse of skinning behaviors across similar models, Ju et al. [Ju+08] construct cages from a library of skinning templates. A skinning template is a specific skeleton configuration that can be applied to characters with a similar joint structure. The system first selects a template matching the character and subsequently constructs a cage fitting the character's shape. For each bone or joint, a polygon represents the cross section along the skeleton. Cage generation scales the cross section vertices radially from its bone or joint so that the cage does not intersect the model. Users may interactively adjust cage vertices to achieve a better cage embedding. To enable cage-based deformation, the system binds the cage to the skeleton to enable cage deformation, whenever the user loads a new skinning template.

As retargeting muscles of a character requires surface deformation, Yang et al. [Yan+12] use cages for muscle construction from skeletons. Joint by joint, their method generates a cage applying different rules for the number of bones at a joint. For joints with two bones, cage generation constructs a cross section polygon like Ju et al. [Ju+08]. For joints with one, three or four bones, ray casting finds positions for polygon vertices at intersection points with the skin. To avoid self-intersections, their method shoots several rays to detect overshooting. In addition, a spherical mapping of the cage to the unit sphere enables optimization of vertex positions according to an energy that penalizes self-intersections.

3.5.5. Interactive Cage Generation Methods

As the automatic construction of cages oftentimes requires subsequent adjustments of the cage, many research papers present methods, where prior user interaction affects the manner of cage construction.

Chen and Feng [CF14] construct cages adapted from skeletons for animated meshes. Users first sketch a skeleton on a silhouette of the model. From the sketched skeleton, Chen and Feng [CF14] extract prominent cross-sections [Sel+10] along the joints. The cage construction interpolates offsets of simplified cross-sections and connects the vertices to adjacent offset cross-sections to construct partial cages. Finally, the partial cages are stitched together into a single cage. Due to the use of the skeleton for cage construction, a sequence of cages can be generated for an animated model.

As automatic cage computation can produce unintuitive results for models with high topological complexity, Le and Deng [LD17] propose an interactive cage generation method for designing cages. Their method enables users to specify the semantic model parts with cut slide planes. These slides are transformed into cross sections spanned by quadrilaterals, whose vertices are part of the cage. As the quadrilaterals should be well aligned to the model and consistently oriented, an optimization step rectifies orientations in a least squares manner. Subsequently, Delaunay meshing followed by edge flips for surface smoothness improvement produce a cage. The final step optimizes cage vertex positions using least squares fitting respecting a user-specified offset. The method of Le and Deng [LD17] allows for quick interactive cage design, while it may produce self-intersecting cages.

To enable quick, interactive generation of coarse cages, Calderon and Boubekeur [CB17] combine an initial parallel voxelization step with subsequent mesh coarsening. After the user chooses a global voxel scale for cage design, the user can interactively brush the regions of the cage that require a finer or coarser bounding approximation. For cage construction, a voxelization forms a 3D rasterization of the input object using a GPU-parallel conservative voxelization method [SS10b]. With the use of a 3D structuring element, a morphological closing of the voxels intersecting the object is obtained, performing morphological dilation followed by erosion. From the resulting voxels, a quad-dominant mesh can be extracted that is further simplified using constrained edge collapse operations, guaranteeing that the cage bounds the model with respect to the voxel scale.

For semi-automatic generation using a skeleton, Casti et al. [Cas+19] guide cage construction by placing bending points along the skeleton curve. A heuristic pre-calculates bending points based on abrupt changes in the thickness of the surface of $\mathcal{M}$. If $\mathcal{M}$ is a surface mesh, then volumetric meshing of $\mathcal{M}$ allows for computing a harmonic field that emanates radially from the skeleton. If $\mathcal{M}$ is a volumetric mesh itself, then it can be used straightforwardly for computing the harmonic field. After circular sampling of the field around the bending points, their method extracts cross sections of $\mathcal{M}$ orthogonal to the skeleton. Connecting polygons representing the cross sections obtains an initial tight cage that is inflated to avoid intersections with the model. While the method of Casti et al. [Cas+19] enables fast generation of high-quality symmetric cages, it requires an input skeleton, a prior volumetric meshing step, and a sufficiently dense set of bending points.

3.5.6. Embedding the Cage

Many cage-based deformation methods (see section 3.6.2) not only depend on a polygonal cage $\mathcal{C}$ but also a volumetric grid $\mathcal{E}_\mathcal{C}$ embedding the cage, because these methods solve partial differential equations to construct a set of GBC. Due to its robustness, unstructured tetrahedral meshing is frequently used to generate $\mathcal{E}_\mathcal{C}$. Although fast and robust tetrahedral meshing algorithms are available [Che+13d; Hu+18; Hu+20], the dependency on an embedding complicates the workflow, because the quality of the deformation is affected by the properties of $\mathcal{E}_\mathcal{C}$.

Generally, the more points $\mathcal{E}_\mathcal{C}$ includes, the better the accuracy of the resulting GBC becomes. Thus, a finer resolution of $\mathcal{E}_\mathcal{C}$ improves deformation quality, whereas the run time performance of calculating the GBC benefits from a coarse $\mathcal{E}_\mathcal{C}$, because each element costs computationally. It can be difficult to estimate an appropriate resolution for $\mathcal{E}_\mathcal{C}$, if the intended deformations are unclear at bind time. An alternative to increasing the resolution of $\mathcal{E}_\mathcal{C}$ is the use of higher-order elements, while current cage-based deformation methods only use linear elements. In addition, the deformation quality also depends on the quality of the tetrahedral elements [She02b] in $\mathcal{E}_\mathcal{C}$. Constructing a constrained tetrahedral mesh without elements of low shape quality is an ongoing research topic [Lo15]. Although scalable mesh improvement methods, e.g. the one of Rabinovich et al. [Rab+17] or the GPU-parallel methods presented in chapter 4, are available, they do not provide guarantees on the resulting element quality. To decouple deformation quality from element quality, one could use the technique of Schneider et al. [Sch+18] at the cost of decreased run time performance and implementing a more complex construction of the finite element system.

In order to extract the deformed model from $\mathcal{E}_\mathcal{C}$, two different methods can be used:

1. Insert the vertices $\mathbf{V}$ into $\mathcal{E}_\mathcal{C}$ as constrained points and extract them after deformation
2. Interpolate the weights of $\mathbf{V}$ from $\mathcal{E}_\mathcal{C}$ and pose the model with interpolated weights

As meshing tools such as TetGen [Si20] typically provide meshes with constrained points in consecutive order, the first method is simple to implement. However, it complicates the meshing of $\mathcal{E}_\mathcal{C}$, because insertion of constrained points leads to additional mesh refinement to avoid elements of low shape quality. The second method provides more convenient meshing and reusability of $\mathcal{E}_\mathcal{C}$, as it can be used for any model enclosed by the cage, given that the resolution of $\mathcal{E}_\mathcal{C}$ is fine enough, albeit at the cost of performing a point lookup step that should be accelerated with a spatial data structure such as the OLBVH (see section 7.1) or using the hardware accelerated data structure for RTX [Mor+22] (see section 3.8.3).

3.6. Construction of Cage Coordinates for Deformation Control

Since the previous section 3.5 presented the creation of cages for deformation control, this section discusses the methods to compute cage coordinates for interactive deformation by cage control. While a detailed review of the individual coordinate types can be found in the survey [Str+24], this thesis discusses the construction of the coordinate types on a higher level of abstraction to highlight the advantages and disadvantages of each construction. Again, a systematic categorization classifies the available methods into four categories. Section 3.6.1 discusses barycentric coordinates with explicit formula. Section 3.6.2 discusses coordinates constructed by energy minimization. Section 3.6.3 discusses probability-based coordinates. Section 3.6.4 discusses coordinates with additional deformation control by normals of cage faces.

3.6.1. Barycentric Coordinates with Explicit Formula

The simplest category of GBC are barycentric coordinates with explicit formula. The construction of these coordinates requires the evaluation of closed form expressions for every to-be-deformed point $\mathbf{x} \in \Omega$ to obtain a corresponding set of GBC $\lambda_i, i = 1, \ldots, N_\mathrm{C}$ (cf. section 2.3). For cage-based deformation, several of the available constructions are limited to convex cages [War96; Ju+05; War+07; JLW07]. Only mean value coordinates support non-convex cages.

Mean Value Coordinates

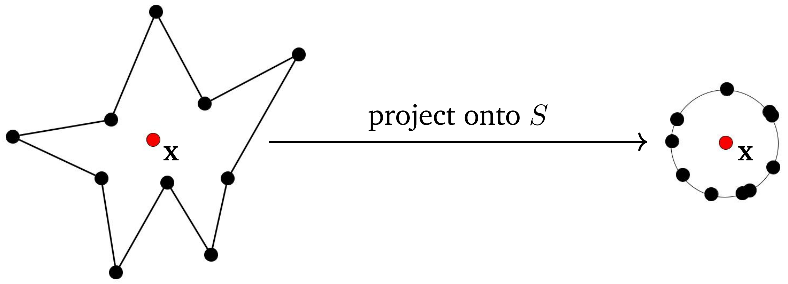
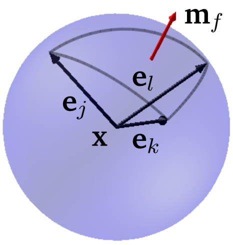


Figure 3.2.: Projection of $\mathcal{C}$ onto the unit sphere S around $\mathbf{x} \in \Omega$ provides support for non-convex cages.

Floater [Flo03] exploited the fact that the integral of outward unit normals of a sphere evaluates to zero to construct 2D mean value coordinates (MVC). As every input cage can be projected onto the unit sphere, this construction achieves support for non-convex cages. A 2D visualization of the projection of the cage onto the unit sphere centered around $\mathbf{x} \in \Omega$ appears in fig. 3.2.



The unit sphere centered around the point $\mathbf{x} \in \Omega$ provides a simple approach to compute a set of weights λ_i so that the weighted sum of cage vertices reproduces $\mathbf{x}$. This generalizes to 3D, which admits the construction of MVC in 3D [FKR05; JSW05]. After the projection of $\mathcal{C}$ onto the unit sphere S, the spherical triangular faces $\hat{f}$ are associated with outward pointing mean vectors $\mathbf{m}_f$, as can be seen in the inset (taken from [Str+24]). These mean vectors represent the integration over outward pointing face normals $\mathbf{m}_f = \int_{\hat{f}} (\mathbf{y} - \mathbf{x}) d\mathbf{y}$, where $\mathbf{y} \in \hat{f}$. For a projected triangular face $\hat{f} = (\hat{\mathbf{c}}_i, \hat{\mathbf{c}}_j, \hat{\mathbf{c}}_k)$, the vectors $\mathbf{e}_i = \mathbf{c}_i - \mathbf{x} / \|\mathbf{c}_i - \mathbf{x}\|$ span a cone that is a part of the unit sphere S. As the mean vector $\mathbf{m}_f$ lies in this cone, there must be weights $\mu_i, \mu_j, \mu_k \in \mathbb{R}$ that express $\mathbf{m}_f$ as a linear combination:

$$\mathbf{m}_f = \mu_i \mathbf{e}_i + \mu_j \mathbf{e}_j + \mu_k \mathbf{e}_k. \tag{3.1}$$

Using the mean vectors of every cage face f, one can express the integration over the unit sphere as a sum of mean vectors:

$$\int_S (\mathbf{y} - \mathbf{x}) d\mathbf{y} = \sum_{f \in \partial \mathcal{C}} o_f \mathbf{m}_f = 0, \tag{3.2}$$

where $o_f \in \{1, \text{-}1\}$ is the orientation of f toward $\mathbf{x}$. Since eq. (3.1) expresses a given mean vector with cage vertices, it is possible to rewrite the sum of mean vectors integrating over the unit sphere. Thus, one can plug eq. (3.1) into eq. (3.2) to obtain:

$$\sum_{f \in \partial \mathcal{C}} o_f \mathbf{m}_f = \sum_{f \in \partial \mathcal{C}} o_f (\mu_i \mathbf{e}_i + \mu_j \mathbf{e}_j + \mu_k \mathbf{e}_k) = \sum_{i=1}^{N_{\mathrm{C}}} \sum_{f \ni i} \frac{o_f \mu_i}{\|\mathbf{c}_i - \mathbf{x}\|} (\mathbf{c}_i - \mathbf{x}) = 0. \tag{3.3}$$

For convenience, the weights w_i^f express the factors in the sum $w_i^f = o_f \mu_i / \|\mathbf{c}_i - \mathbf{x}\|$. The total sum of weights for a given point $\mathbf{x} \in \Omega$ and a given cage $\mathcal{C}$ is represented by $w_{\mathbf{x}}^{\mathcal{C}} = \sum_{i=1}^{N_{\mathrm{C}}} \sum_{f \ni i} w_i^f$. This admits to rewrite eq. (3.3) so that a weighted sum of cage vertices reproduces the given point $\mathbf{x} \in \Omega$:

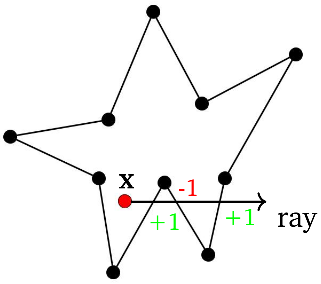
$$\sum_{i=1}^{N_{\mathrm{C}}} \sum_{f \ni i} w_i^f (\mathbf{c}_i - \mathbf{x}) = 0 \iff \sum_{i=1}^{N_{\mathrm{C}}} \sum_{f \ni i} \frac{w_i^f}{w_{\mathbf{x}}^{\mathcal{C}}} \mathbf{c}_i = \mathbf{x}. \tag{3.4}$$

As a result, the GBC $\lambda_i(\mathbf{x}) = \sum_{f \ni i} w_i^f / w_{\mathbf{x}}^{\mathcal{C}}, i = 1, \ldots, N_{\mathrm{C}}$ define MVC for a given cage $\mathcal{C}$ even for non-convex cages. While this is a simple construction for cages of arbitrary shape, the GBC can be negative outside of the convex region of $\mathcal{C}$, i.e., the kernel of $\mathcal{C}$. An additional downside is that the above construction of MVC only applies to triangular cage faces.

Positive Mean Value Coordinates

In order to obtain positive GBC, Lipman et al. [Lip+07] present a modification of MVC that ensures the positivity of the coordinates at an arbitrary point inside the cage. This modification is called positive mean value coordinates (PMVC). The reason for negative MVC is that the orientation of cage faces o_f toward $\mathbf{x} \in \Omega$ can be negative. Lipman et al. [Lip+07] perform a visibility calculation of cage faces from $\mathbf{x} \in \Omega$. The visibility calculations can be efficiently performed on GPUs.

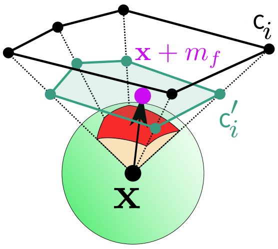
If one orders all the cage faces by intersection with a view ray emanating from $\mathbf{x} \in \Omega$, then the orientation o_f of the faces alternates (see inset). Thus, if one only considers the first visible face from $\mathbf{x} \in \Omega$ it is guaranteed that $o_f = +1$ and the resulting coordinates are positive. Therefore, Lipman et al. [Lip+07] only use the first visible cage face to calculate PMVC. This approach projects the unit sphere S onto $\mathcal{C}$ instead of projecting $\mathcal{C}$ onto S. However, PMVC are an approximation, which is accurate and fast enough for real-time shape deformation.



Spherical Mean Value Coordinates

As the original construction of MVC only supports triangular cages, Langer et al. [LBS06] present spherical barycentric coordinates (SBC) that are another modification of MVC to support arbitrary planar cage polygons. The construction of SBC exploits the fact that there is an infinite set of weights w_i, which satisfies:

$$\sum_{i=1}^{N_\mathcal{C}} \sum_{f \ni i} w_i(\mathbf{c}_i - \mathbf{x}) = \sum_{f \in \partial\mathcal{C}} o_f \mathbf{m}_f = 0. \tag{3.5}$$



With weights and corresponding mean vectors that satisfy eq. (3.5), one can apply the same rearrangements as performed to obtain eq. (3.4), which provides a set of GBC. Therefore, one needs to define mean vectors $\mathbf{m}_f$and appropriate weights for a non-triangular face f to obtain GBC for cage-based deformation. After calculating a mean vector $\mathbf{m}_f$ for the non-triangular face f, the SBC construction projects the point $\mathbf{x}$ onto the point $\mathbf{x} + \mathbf{m}_f$, as can be seen in the inset (taken from [Str+24]). This projected point defines a plane that is tangent to the unit sphere S. Subsequently, the SBC construction projects the cage vertices in f onto this tangent plane, which provides the projected cage vertices $\mathbf{c}_i'$. The projection can be expressed as scaling the vectors from $\mathbf{x}$ to $\mathbf{c}_i$ by a factor λ_i^f so that $\mathbf{c}_i' - \mathbf{x} = \lambda_i^f(\mathbf{c}_i - \mathbf{x})$. The projected cage vertices define a planar polygon f' that lies in one plane with $\mathbf{x} + \mathbf{m}_f$. The next step of the SBC construction is to calculate 2D MVC for $\mathbf{x} + \mathbf{m}_f = \sum_{i \in f} \omega_i^f \mathbf{c}_i'$. Using the projection factors λ_i^f, one can express the mean vectors of the non-triangular face:

$$\sum_{i \in f} \omega_i^f (\mathbf{c}' - \mathbf{x}) = \sum_{i \in f} \lambda_i^f \omega_i^f (\mathbf{c} - \mathbf{x}) = \mathbf{m}_f.$$

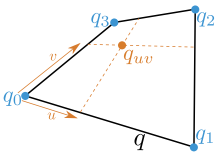
As a result, the construction satisfies eq. (3.5) so that it yields valid GBC for non-triangular cage faces. However, this only applies for planar cage faces and the coordinates can be negative.

Mean Value Coordinates for Tri-Quad Cages

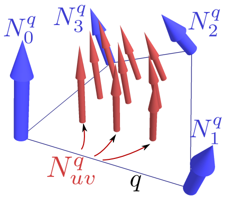
The downside of SBC is that they are limited to planar polygons. The support of non-planar polygons can significantly extend the freedom of the user at cage design as well as the space of user-defined deformation.

For computing valid GBC for non-planar cage quads, Thiery et al. [TMB18] present a method to construct mean value coordinates for tri-quad cages (QMVC). Their construction computes GBC for cages with triangles and quadrilaterals, i.e., tri-quad cages. The key strategy is the usage of bilinear quadrilaterals for interpolating outward face normals on a non-planar quad. A quad $q = (\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$ enables bilinear interpolation using the bilinear coordinates $\{b_{uv}^0, b_{uv}^1, b_{uv}^2, b_{uv}^3\} = \{(1-u)(1-v), u(1-v), uv, (1-u)v\}$ at parameters $(u, v) \in \mathbb{R}^2$.

Bilinear quads provide a smooth geometry given by $\mathbf{q}_{uv} = \sum_{k=0}^{3} b_{uv}^{k}\mathbf{q}_k$, as can be seen in the inset to the right (taken from [Str+24]). Denoting the four non-normalized quad corner normals as $\mathbf{N}_k^q, k = 0, 1, 2, 3$ defined by $\mathbf{N}_k^q = (\mathbf{q}_{k+1} - \mathbf{q}_k) \times (\mathbf{q}_{k+3} - \mathbf{q}_k)$ (all indices modulo 4), the non-normalized normal vector $\mathbf{N}_{uv}^q$ at parameter (u, v) is given by



$$\mathbf{N}_{uv}^q = \sum_{k=0}^{3} b_{uv}^{k}\mathbf{N}_k^q.$$



As a result, the bilinear interpolation provides outward pointing normal vectors for any parameter point (u, v) at the non-planar quad, as can be seen in the inset to the left (taken from [Str+24]). The *surface element* corresponding to the (u, v)-parameterization is given by $\mathrm{d}\mathbf{q}_{uv} = \|\mathbf{N}_{uv}^q\| \,\mathrm{d}u\,\mathrm{d}v$ and surface integrals over q can be written as

$$\int_q f \mathrm{d}q = \int_0^1 \int_0^1 f(\mathbf{q}_{uv})\|\mathbf{N}_{uv}^q\| \,\mathrm{d}u\,\mathrm{d}v.$$

Using the bilinear quad's geometric model to integrate outward face normals, one can define the mean vector $\mathbf{m}_q = \sum_{i=0}^{3} w_{\mathbf{q}_i}^q(\mathbf{q}_i - \mathbf{x})$ for the non-planar quad. This can be written in matrix form as $\mathbf{m}_q = \mathbf{A}_q\mathbf{w}^q$, where $\mathbf{w}^q = (w_{\mathbf{q}_0}^q, w_{\mathbf{q}_1}^q, w_{\mathbf{q}_2}^q, w_{\mathbf{q}_3}^q)^\mathsf{T} \in \mathbb{R}^4$ are the four unknown non-normalized coordinates, and the j-th column of $\mathbf{A}_q$ is provided by $(\mathbf{q}_j - \mathbf{x})$. However, as $\mathbf{A}_q$ is not full-rank, the equation cannot be straightforwardly solved to deduce QMVC. Consequently, Thiery et al. [TMB18] resorted to an approximate solution that efficiently computes smooth, bilinearly interpolating, and valid coordinates. In order to compute QMVC, an adaptive Riemann summation strategy relies on bilinear interpolation at several parameter points over the bilinear quad. This thesis does not discuss the details of this approximation strategy, which provides geometrical expressions for the construction of QMVC. For the exact details, the interested reader is referred to the original paper [TMB18] or the open source implementation of CageModeler.

3.6.2. Energy Minimization-based Barycentric Coordinates

While the coordinate types in section 3.6.1 offer simple closed form expressions for constructing a valid set of GBC, the resulting coordinates suffer from a variety of issues, e.g., negative coordinates and limited locality of deformation control. Therefore, another approach has been devised, which ensures positive coordinates and local influence. The energy minimization-based coordinates (EMC) provide a set of GBC by discretizing a suitable PDE and minimizing the resulting energy function. The discretization also requires the generation of a volumetric mesh for the interior of $\mathcal{C}$ (cf. section 3.5.6). High-quality coordinates are ensured by enforcing additional constraints to the minimization problem.

Harmonic Coordinates

Joshi et al. [Jos+07] present a construction of non-negative GBC by numerically solving the Laplace equation subject to suitable boundary conditions. The resulting harmonic coordinates (HC) pose an approximate solution that is sufficient for cage-based deformation and provide the desired properties listed in section 2.3.2. In order to numerically compute HC, let $\xi_i : \partial\mathcal{C} \to \mathbb{R}, i = 1, \dots, N_\mathbf{C}$ be the continuous functions subject to a list of constraints:

- ξ_i is linear along the edges of $\mathcal{C}$
- ξ_i is harmonic inside each cage face f
- ξ_i exhibits the Lagrange property $\xi_i(\mathbf{c}_j) = \delta_{i,j}$ for $j = 1, \dots, N_\mathbf{C}$

This definition implies that ξ_i vanishes over all faces that do not include $\mathbf{c}_i$, ξ_i varies linearly over cage triangles, and ξ_i is bilinear over cage quads. In order to provide local influence near each cage vertex, Joshi et al. [Jos+07] define HC λ_i as the solution to the Laplace equation:

$$\Delta\,\lambda_i(\mathbf{x}) = 0,\ \mathbf{x} \in \operatorname{Int}\mathcal{C} \qquad \text{subject to the Dirichlet boundary condition} \qquad \lambda_i(\mathbf{x}) = \xi_i(\mathbf{x}),\ \mathbf{x} \in \partial\mathcal{C}. \tag{3.6}$$

Typical numerical methods for approximating the exact solution of the Laplacian PDE include finite difference methods [Jos+07], the FEM [Mar+08], the boundary element method [Rus07], and the method of fundamental solutions [FK98]. However, only the use of the FEM guarantees the correct behavior over $\partial\mathcal{C}$.

Bounded Biharmonic Weights

Modeling objects using either cages or skeletons can be tedious, because each control structure has its merits and shortcomings. Thus, Jacobson et al. [Jac+11] present bounded biharmonic weights (BBW) to bind a model to several cages, skeletons, and point handles. Strictly speaking, BBW are the result of minimizing piecewise-linear functions w_j, which do not provide coordinates for the domain Ω of the deformation. In order to combine cages with other control structures, Jacobson et al. [Jac+11] exploit the fact that cage-based deformation is a special case of LBS (see section 2.3.3). For a number $N \geq N_\mathbf{C}$ of handles, the computation of BBW performs variational weight optimization minimizing Laplacian energy subject to a multitude of constraints that enforce interpolation:

$$\begin{aligned}
&\operatorname*{arg\,min}_{w_1,\dots,w_N} \sum_{j=1}^{N} \frac{1}{2} \int_{\Omega} \|\Delta w_j\|^2 \,\mathrm{d}V \\
&\text{subject to } w_j(\mathbf{h}_k) = \delta_{jk}, \\
&\qquad \forall f \in \partial\mathcal{C}, w_j \text{ is linear on } f, \\
&\qquad \forall \mathbf{x} \in \Omega, \sum_{j=1}^{N} w_j(\mathbf{x}) = 1, \\
&\qquad \forall \mathbf{x} \in \Omega, w_j(\mathbf{x}) \in [0,1], j = 1, \dots, N.
\end{aligned}$$

As shape preservation during deformation is an important property, the deformation should allow to preserve a user-specified sub-region $\Pi \subset \Omega$. For this reason, the BBW formulation admits the incorporation of a rigidity mask represented by functions $\rho\colon \Pi \to \mathbb{R}^+$ and the addition of the following least squares term:

$$\sum_{j=1}^{N} \frac{1}{2} \int_{\Pi} \rho \|\Delta w_j\|^2 \,\mathrm{d}V.$$

Local Barycentric Coordinates

Since cage-based deformation expresses the geometry as an affine sum of cage vertices (cf. eq. (2.15)), the relocation of a single control point can lead to a global change by propagation into the entire domain Ω, which is not intended by the user. In order to overcome this limitation, Zhang et al. [Zha+14] present local barycentric coordinates (LBC). For each interior point, LBC only include a small set of nearby cage vertices for the affine combination, while the coordinates of distant cage vertices vanish, falling below $\varepsilon \approx 0$.

The key strategy to compute LBC is to minimize total variation. Total variation energies offer two crucial benefits for deformation control. First, total variation provides a metric for oscillation [CV01]. Second, the total variation of a set equals the perimeter of the set [EG15]. Given a strict super level set $L_s^+(\lambda_i) = \{\lambda_i <$

$s\} = \{\mathbf{x} \in \Omega \mid \lambda_i(\mathbf{x}) > s\}$ of an arbitrary but fixed s and a GBC function λ_i, the locality of λ_i increases if the area/volume of $L_s^+(\lambda_i)$ decreases. With the use of total variation of λ_i, one can minimize the perimeter $P(L_s^+(\lambda_i);\Omega)$ to increase locality:

$$\int_{-\infty}^{+\infty} P(L_s^+(\lambda_i);\Omega)\,\mathrm{d}s = \int_\Omega |\nabla \lambda_i\,|\,\mathrm{d}V.$$

Minimizing the sum of total variations for all N_C functions λ_i yields LBC for $\mathcal{C}$. In order to control locality, the penalty coefficients $\hat{\phi}_i$ adjust the locality of λ_i. As a result, solving the following variational convex optimization problem subject to constraints enforcing the desired interpolation properties produces LBC:

$$\begin{aligned} &\underset{\lambda_1,\ldots,\lambda_{N_\mathsf{C}}}{\arg\min} \sum_{i=1}^{N_\mathsf{C}} \int_\Omega \hat{\phi}_i \|\,\lambda_i\,\| \mathrm{d}V \\ \text{subject to } &\sum_{i=1}^{N_\mathsf{C}} \lambda_i(\mathbf{x})\mathbf{c}_i = \mathbf{x}, \sum_{i=1}^{N_\mathsf{C}} \lambda_i(\mathbf{x}) = 1, \lambda_i(\mathbf{x}) \geq 0, \forall \mathbf{x} \in \Omega \\ &\lambda_i(\mathbf{c}_j) = \delta_{ij}, \forall i,j \in \{1,\ldots,N_\mathsf{C}\} \\ &\lambda_i \text{ is linear on all } f, \forall i \in \{1,\ldots,N_\mathsf{C}\} \end{aligned}$$

The coefficients $\hat{\phi}_i$ penalize the gradient norm based on the geodesic distance $g_i(\cdot)$ to the cage vertex $\mathbf{c}_i$ and a continuous function $\hat{\tau}\colon [0,1] \to [0,1]$:

$$\hat{\phi}_i = \hat{\tau}\left(\frac{g_i(\mathbf{x})}{\arg\max_{\mathbf{y}\in\Omega} g_i(\mathbf{y})}\right), \quad \text{with } \mathbf{x} \in \Omega.$$

When choosing a monotonically increasing function for $\hat{\tau}$, points more distant from $\mathbf{c}_i$ receive larger penalty coefficients, which leads to more local support. Analogously, the choice of a monotonically decreasing function for $\hat{\tau}$ reduces local support.

As the numerical computation of LBC suffers from low run time performance for high-resolution models, Tao et al. [TDZ19] simplify the discretized formulation, incurring only negligible deviation from the original discretization. This simplification primarily capitalizes on the removal of the non-negativity constraint to reformulate the problem so that costly computations for solving global linear systems can be omitted. The simplified formulation uses auxiliary variables to obtain a separable target function that can be minimized with the alternating direction method of multiplier [Boy10]. The numerical computation is a complex scheme of many steps, which is omitted in this thesis, while the details can be found in the survey [Str+24] and the original paper [TDZ19].

3.6.3. Probability-based Coordinates

One major advantage of EMC is the guarantee of positive coordinates for non-convex cages, though these constructions impose to solve a global non-linear optimization problem. Due to the lack of explicit closed-form expressions, it is impossible to compute EMC locally for an arbitrary point $\mathbf{x} \in \Omega$. An approximation must first be computed by discretizing and solving a PDE. An alternative construction of GBC to efficiently compute non-negative coordinates relies on statistical concepts. In this construction, the GBC of $\mathbf{x} \in \Omega$ represent the probabilities of discrete random events that are associated with the cage vertices $\mathbf{c}_i$, and the construction determines a probability distribution, such that the expected value of the random variable $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_{N_\mathsf{C}})$ is $\mathrm{E}[\mathbf{c}] = \sum_{i=1}^{N_\mathsf{C}} \lambda_i(\mathbf{x})\mathbf{c}_i = \mathbf{x}$. This construction typically admits the computation of the GBC by solving an optimization problem that is local in the sense that it determines $\lambda_i(\mathbf{x}), i = 1, \ldots, N_\mathsf{C}$ of $\mathbf{x} \in \Omega$ independently of the coordinates $\lambda_i(\mathbf{y}), i = 1, \ldots, N_\mathsf{C}$ of any other point $\mathbf{y} \neq \mathbf{x}$.

Maximum Entropy Coordinates

Sukumar [Suk04] presents a construction for maximum entropy coordinates (MEC) that maximizes the *Shannon entropy*

$$H_{\mathbf{x}}(\lambda) = -\sum_{i=1}^{N_{\mathrm{C}}} \lambda_i(\mathbf{x}) \log \lambda_i(\mathbf{x}) \qquad \text{subject to} \qquad \sum_{i=1}^{N_{\mathrm{C}}} \lambda_i(\mathbf{x}) = 1, \qquad \sum_{i=1}^{N_{\mathrm{C}}} \lambda_i(\mathbf{x}) \mathbf{c}_i = \mathbf{x}. \tag{3.7}$$

While the resulting GBC are well-suited for convex cages, the Lagrange property is not satisfied at concave cage vertices. For better support of non-convex cages, Hormann and Sukumar [HS08] use the *Shannon–Jaynes entropy* instead of eq. (3.7):

$$H_{\mathbf{x}}(\lambda) = -\sum_{i=1}^{N_{\mathrm{C}}} \lambda_i(\mathbf{x}) \log \frac{\lambda_i(\mathbf{x})}{m_i(\mathbf{x})}. \tag{3.8}$$

However, this construction requires the definition of *prior functions* $m_i \colon \mathcal{C} \to \mathbb{R}$ as

$$m_i(\mathbf{x}) = \frac{\pi_i(\mathbf{x})}{\sum_{j=1}^{N_{\mathrm{C}}} \pi_j(\mathbf{x})}, \qquad \pi_i(\mathbf{x}) = \frac{1}{\prod_{f \ni i} \rho_f(\mathbf{x})}, \tag{3.9}$$

where the product in the denominator of π_i ranges over all faces adjacent to $\mathbf{c}_i$. Provided that $f \in \partial\mathcal{C}$ is a polygon with k vertices $\mathbf{c}_1, \ldots, \mathbf{c}_k$, the function $\rho_f \colon \mathcal{C} \to \mathbb{R}$ is defined as

$$\rho_f(\mathbf{x}) = \sum_{i=1}^{k} \operatorname{area}(\mathbf{x}, \mathbf{c}_i, \mathbf{c}_{i+1}) - \operatorname{area}(\mathbf{c}_1, \ldots, \mathbf{c}_k).$$

$\rho_f(\mathbf{x})$ is positive and vanishes if and only if $\mathbf{x} \in f$. Thus, the prior function m_i in eq. (3.9) vanishes over $\partial\mathcal{C}$, except on the faces including $\mathbf{c}_i$, and provides the Lagrange property.

Maximizing $H_{\mathbf{x}}(\lambda)$ in eq. (3.8) under the constraints in eq. (3.7) is equivalent to first finding the unique vector $\eta \in \mathbb{R}^3$ that minimizes the strictly convex function

$$F(\eta) = \log \sum_{i=1}^{N_{\mathrm{C}}} Z_i(\eta), \qquad Z_i(\eta) = m_i(\mathbf{x}) e^{-\eta^{\mathsf{T}}(\mathbf{c}_i - \mathbf{x})},$$

which can be solved efficiently with a few iterations of Newton's method and then computing the coordinates as

$$\lambda_i(\mathbf{x}) = \frac{Z_i(\eta)}{\sum_{j=1}^{N_{\mathrm{C}}} Z_j(\eta)}, \qquad i = 1, \ldots, N_{\mathrm{C}}.$$

The resulting MEC provide all the desired properties listed in section 2.3.2, except for locality. However, since the particular choice of prior functions in eq. (3.9) is not "geometry-aware", the use of MEC can lead to badly shaped coordinate functions and deformation artifacts.

Maximum Likelihood Coordinates

Another probability-based construction of GBC are the maximum likelihood coordinates (MLC) presented by Chang et al. [CDH23]. Instead of maximizing *Shannon entropy*, they maximize the product of the GBC $\lambda_i, i = 1 \ldots, N_{\mathrm{C}}$. A key advantage of MLC over MEC is the support of non-convex cages without the need to choose prior functions.

The first step to calculate MLC is to relocate the cage vertices by $-\mathbf{x}$ so that $\mathcal{C}$ is centered around $\mathbf{x} \in \Omega$. Similar to constructing MVC the cage is projected onto the unit sphere (see fig. 3.3), which provides a simple convex domain for coordinate construction. Thus, each cage vertex $\mathbf{c}_i$ is projected to $\mathring{\mathbf{c}}_i = (\mathbf{c}_i - \mathbf{x})/\|\mathbf{c}_i - \mathbf{x}\|$. A few smoothing operations smooth the projected cage, which provides the cage $\hat{\mathcal{C}}$ with vertices $\hat{\mathbf{c}}_i, i = 1, \dots, N_\mathsf{C}$. In the first smoothing step, the construction computes the mean vectors $\mathbf{m}_f$ for each spherical face. In the subsequent smoothing step, the construction computes the sum of normalized $\mathbf{m}_f$ around each projected vertex $\mathring{\mathbf{c}}_i$ to obtain:

$$\hat{\mathbf{c}}_i = \mathbf{t}_i/\|\mathbf{t}_i\|, \qquad \mathbf{t}_i = \sum_{f \ni i} \mathbf{m}_f/\|\mathbf{m}_f\|.$$

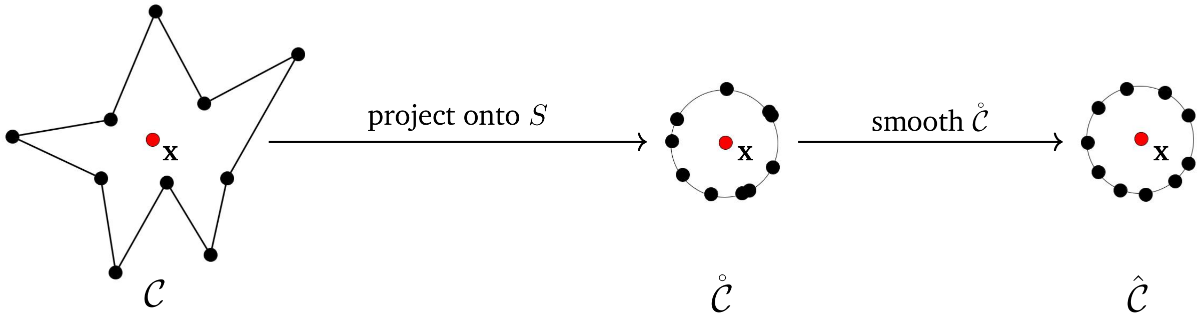


Figure 3.3.: Projection of $\mathcal{C}$ onto the unit sphere S around $\mathbf{x} \in \Omega$ and subsequent smoothing provides a simple scheme to compute coordinates even for non-convex cages.

Since the mean vectors $\mathbf{m}_f$ can be expressed as non-negative linear combinations of the cage vertices (cf. eq. (3.1)), one can assemble a matrix of all the vertices in $\hat{\mathcal{C}}$ with the input vertices $\mathbf{c}_i$:

$$\hat{\mathbf{C}} = \mathbf{M}\big(\mathbf{C} - \mathbf{e}\mathbf{x}^\mathsf{T}\big),$$

where $\mathbf{e} = (1, \dots, 1)^\mathsf{T} \in \mathbb{R}^n$ is a vector of ones and $\mathbf{M} \in \mathbb{R}^{N_\mathsf{C} \times N_\mathsf{C}}$ is a non-negative transformation matrix.

With the use of the transformation matrix $\mathbf{M}$ and the non-negative GBC $\hat{\lambda} = (\hat{\lambda}_1, \dots, \hat{\lambda}_{N_\mathsf{C}})$ of the origin respecting the cage $\hat{\mathcal{C}}$, one can define the GBC $\lambda = (\lambda_1(\mathbf{x}), \dots, \lambda_{N_\mathsf{C}}(\mathbf{x}))$ of $\mathbf{x} \in \Omega$ for the original cage as:

$$\lambda = \frac{\hat{\lambda}\mathbf{M}}{\hat{\lambda}\mathbf{M}\mathbf{e}}. \tag{3.10}$$

Therefore, the problem to finding GBC for $\mathbf{x} \in \Omega$ with respect to $\mathcal{C}$ is reduced to the problem of finding GBC $\hat{\lambda} = (\hat{\lambda}_1, \dots, \hat{\lambda}_{N_\mathsf{C}})$ of the origin with respect to the spherical cage $\hat{\mathcal{C}}$. Chang et al. [CDH23] determine $\hat{\lambda}$ by maximizing the function

$$\ell(\hat{\lambda}) = \log \prod_{i=1}^{N_\mathsf{C}} \hat{\lambda}_i = \sum_{i=1}^{N_\mathsf{C}} \log \hat{\lambda}_i \qquad \text{subject to} \qquad \sum_{i=1}^{N_\mathsf{C}} \hat{\lambda}_i = 1, \qquad \sum_{i=1}^{N_\mathsf{C}} \hat{\lambda}_i \hat{\mathbf{c}}_i = \mathbf{x}.$$

This optimization problem can be efficiently solved with Lagrangian multipliers, which reduces the problem to finding the minimum of the following function F depending on only $d = 3$ variables:

$$F(\eta) = -\sum_{i=1}^{N_\mathsf{C}} \log\big(N_\mathsf{C} + \eta^\mathsf{T}(\hat{\mathbf{c}}_i - \mathbf{x})\big).$$

A few iterations of Newton's method efficiently determine the minimum $\eta^* \in \mathbb{R}^3$. The GBC of the origin with respect to $\hat{\mathcal{C}}$ are then defined as

$$\hat{\lambda}_i = \frac{1}{N_{\mathbf{C}} + \eta^\mathsf{T}(\hat{\mathbf{c}}_i - \mathbf{x})}.$$

It remains to plug these non-negative coordinates into eq. (3.10) to obtain MLC that exhibit all the listed properties in section 2.3.2. Chang et al. [CDH23] also detail how to obtain the derivatives of MLC at $\mathbf{x} \in \Omega$. However, one robustness issue is that for $\mathbf{x}$ being close to $\partial\mathcal{C}$ the spherical cage $\hat{\mathcal{C}}$ might not include the origin, which leads the construction of MLC to produce improper GBC.

3.6.4. Coordinates with Normal Control

While many constructions for GBC with different advantages and disadvantages were proposed, cage-based deformation with GBC always interpolates the model from the cage vertices alone. This interpolatory deformation tool is inherently limited by preserving the model shape and surface features only be means of the positioning of cage vertices. However, better shape preservation can be achieved by not only considering cage vertices but also the normals of cage faces, because the face normals control the deformation for the geometry in between cage vertices. The resulting coordinates with normal control are derived from boundary integral formulations of PDEs. Thus, they inherit simple pointwise evaluation coordinates with explicit formulas (cf. section 3.6.1) but also offer the smoothness of EMC (cf. section 3.6.2) without the requirement to discretize and numerically solve a linear system.

Green Coordinates

The Green coordinates (GC), formulated by Lipman et al. [LLC08] for triangular cages, differ from the use of GBC $\lambda_i, i = 1, \ldots, N_{\mathbf{C}}$, as GC consist of coordinates $\phi_i, i = 1, \ldots, N_{\mathbf{C}}$ for cage vertices and coordinates $\psi_j, j = 1, \ldots, N_{\mathbf{F}}$ for triangle normals (see fig. 3.4). Therefore, GC infer local shape rotations from pure cage vertex translations.

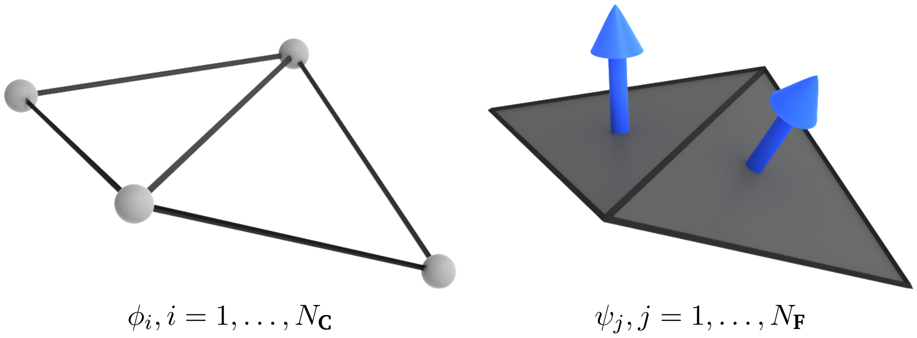

$\phi_i, i = 1, \ldots, N_{\mathbf{C}}$
$\psi_j, j = 1, \ldots, N_{\mathbf{F}}$

Figure 3.4.: Instead of using GBC for cage vertices only, GC take into account coordinates ϕ_i and ψ_j for cage vertices and face normals, respectively.

The derivation of GC is based on Green's third identity, which expresses the interior of a volumetric domain by diffusion from its boundary. This is what one wants to achieve in cage-based deformation, as the configuration of the cage $\mathcal{C}$ is supposed to define the shape wrapped by $\mathcal{C}$. For a harmonic function $u(\mathbf{y}), \mathbf{y} \in \mathbb{R}^3$, Green's third identity provides a concise identity mapping:

$$u(\mathbf{y}) = \int_{\partial\mathcal{C}} u(\mathbf{y}) \frac{\partial_1 G(\mathbf{y}, \mathbf{x})}{\partial \mathbf{n}_\mathbf{y}} \, \mathrm{d}a_\mathbf{y} - \int_{\partial\mathcal{C}} G(\mathbf{y}, \mathbf{x}) \frac{\partial u(\mathbf{y})}{\partial \mathbf{n}_\mathbf{y}} \, \mathrm{d}a_\mathbf{y}, \tag{3.11}$$

where $\mathrm{d}a_{\mathbf{y}}$ is the area element on $\partial\mathcal{C}$, $\mathbf{n}_{\mathbf{y}}$ is the normal at $\mathbf{y}$ on $\partial\mathcal{C}$, and $G(\mathbf{y}, \mathbf{x}) = -(4\pi\|\mathbf{y} - \mathbf{x}\|)^{-1}$ is the fundamental solution of the Laplacian equation in $d = 3$.

Since the normals and function G can be determined, it remains to set boundary Dirichlet and Neumann conditions for u to express the diffusion in eq. (3.11). The Dirichlet condition can be obtained by interpolating from cage triangle vertices:

$$u(\mathbf{y}) = \sum i \in t\Gamma_i^t(\mathbf{y})\mathbf{c}'_i,$$

where Γ_i^t are predetermined basis functions. For the Neumann condition, Lipman et al. [LLC08] propose to set a mapping that is a good approximation of a quasi conformal mapping. They map the input normal $\mathbf{n}_t$ to the deformed normal $\mathbf{n}'_t$ and multiply by a stretch factor σ_t, which accounts for the stretch of a triangle when mapping t to t':

$$\frac{\partial u(\mathbf{y})}{\partial \mathbf{n}_{\mathbf{y}}} = \sigma_t \mathbf{n}'_t, \qquad \forall \mathbf{y} \in t.$$

With the use of two edge vectors $e_1 \in \mathbb{R}^3$ and $e_2 \in \mathbb{R}^3$ from the triangle t, the stretch factor σ_t can be computed as:

$$\sigma_t = \sqrt{\frac{\|\mathbf{e}'_1\|^2\|\mathbf{e}_2\|^2 + \|\mathbf{e}'_2\|^2\|\mathbf{e}_1\|^2 - 2(\mathbf{e}_1 \cdot \mathbf{e}_2)(\mathbf{e}'_1 \cdot \mathbf{e}'_2)}{2\|\mathbf{e}_1 \times \mathbf{e}_2\|^2}}. \tag{3.12}$$

After setting the boundary conditions as above, the final deformation scheme can be written as:

$$\mathbf{x}' = \sum_{i=1}^{N_{\mathrm{C}}} \phi_i(\mathbf{x})\mathbf{c}'_i + \sum_{j=1}^{N_{\mathrm{F}}} \psi_j(\mathbf{x})\sigma_{t_j}\mathbf{n}'_{t_j},$$

with coordinates ϕ_i and ψ_j:

$$\begin{aligned}
\phi_i^t(\mathbf{x}) &= \int_t \Gamma_i^t(\mathbf{y}) \frac{\partial_1 G(\mathbf{y}, \mathbf{x}))}{\partial \mathbf{n}_{\mathbf{y}}} \, \mathrm{d}a_{\mathbf{y}}, \\
\phi_i(\mathbf{x}) &= \sum_{i \in t} \phi_i^t(\mathbf{x}), \\
\psi_j(\mathbf{x}) &= -\int_{t_j} G(\mathbf{y}, \mathbf{x}) \, \mathrm{d}a_{\mathbf{y}}.
\end{aligned}$$

Computing GC with the above Dirichlet and Neumann conditions empirically provides a 3D quasi conforming deformation. As a consequence to building upon boundary diffusion, the construction of GC only works for non-self-intersecting cages. While the diffusion is limited to the the interior of the Ω, Lipman et al. [LLC08] detail an algorithm to enable the extrapolation to points $\mathbf{x} \notin \Omega$, which is not presented in this thesis for brevity. Since Ben-Chen et al. [BWG09] noted that expressions for more efficient and accurate computation of GC are given by Urago [Ura00], along with their gradients and Hessians, it is recommended to use the expressions in [Ura00] instead of the original scheme formulated by Lipman et al. [LLC08].

Green Coordinates for Tri-Quad Cages

As the above construction for GC is limited to triangular cages, Thiery and Boubekeur [TB22] propose a construction of Green coordinates for tri-quad cages (QGC). Their construction relies on many key ideas for QMVC including:

- the use of validity equations to derive coordinates with a reproduction of identity,

- the usage of bilinear quads for interpolation,
- numerical approximation with a Riemann summation for efficiency,
- solving for the null-space of the validity equations.

Since the key to the derivation of GC is the determination of suitable Dirichlet and Neumann conditions, one needs to set appropriate boundary conditions for a non-planar quad q to achieve support for tri-quad cages. If one maps a bilinearly interpolant from its rest state $\mathbf{q}_{uv}$ to its deformed state $\mathbf{q}'_{uv}$, it leads to non-constant face normals. The stretch factors for quads $\sigma_q(u,v)$ are longer computed using triangle edge vectors. As the stretch factors for quads are computed with tangent vectors $(\partial_u \mathbf{q}_{uv}, \partial_v \mathbf{q}_{uv}, \partial_u \mathbf{q}'_{uv}, \partial_v \mathbf{q}'_{uv})$ of the bilinear quad, the resulting stretch factors are non-constantly varying over q. Further following the exact derivation for GC with bilinear quads provides the following integrals for the Dirichlet and Neumann conditions:

$$\text{Dirichlet:} \qquad \mathrm{u}^q_D(\mathbf{x}) = \int_0^1 \int_0^1 \mathbf{q}'_{uv} \frac{(\mathbf{q}_{uv} - \mathbf{x}) \cdot \mathbf{N}^q_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|^3} \, \mathrm{d}u \, \mathrm{d}v,$$

$$\text{Neumann:} \qquad \mathrm{u}^q_N(\mathbf{x}) = \int_0^1 \int_0^1 \frac{\sigma_q(u,v) \mathbf{n}^{\mathbf{q}'}_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|} \|\mathbf{N}^q_{uv}\| \, \mathrm{d}u \, \mathrm{d}v.$$

Unfortunately, these integrals are technically difficult to solve with closed-form expressions. For this reason, Thiery and Boubekeur [TB22] resort to a robust approximation strategy that uses an auxiliary stretch factor:

$$\sigma^{\text{aux}}_q(u,v) = \frac{\|\mathbf{N}^{q'}_{uv}\|}{\|\mathbf{N}^q_{uv}\|}.$$

Inserting the auxiliary stretch factor into the Neumann condition leads to a simpler form:

$$\begin{aligned}
\mathrm{u}^q_N(\mathbf{x}) &= \int_0^1 \int_0^1 \frac{\sigma_q(u,v) \mathbf{n}^{q'}_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|} \frac{\sigma^{\text{aux}}_q(u,v)}{\sigma^{\text{aux}}_q(u,v)} \|\mathbf{N}^q_{uv}\| \, \mathrm{d}u \, \mathrm{d}v \\
&= \int_0^1 \int_0^1 \frac{\mathbf{N}^{q'}_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|} \frac{\sigma_q(u,v)}{\sigma^{\text{aux}}_q(u,v)} \, \mathrm{d}u \, \mathrm{d}v \\
&= \sum_{k=0}^{3} \int_0^1 \int_0^1 \frac{b^k_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|} \frac{\sigma_q(u,v)}{\sigma^{\text{aux}}_q(u,v)} \, \mathrm{d}u \, \mathrm{d}v \, \mathbf{N}^{q'}_k \\
&\simeq \sum_{k=0}^{3} \psi^k_q(\mathbf{x}) \sigma^k_q \mathbf{N}^{q'}_k, \qquad (3.13)
\end{aligned}$$

with

$$\psi^k_q(\mathbf{x}) = \int_0^1 \int_0^1 \frac{b^k_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|} \, \mathrm{d}u \, \mathrm{d}v, \qquad (3.14)$$

appearing as the contribution of a quad vertex to the coordinates for quad normals and σ^k_q as a stretch factor approximating the ratio $\sigma_q(u,v)/\sigma^{\text{aux}}_q(u,v)$ over the bilinear quad, importance-sampled by b^k_{uv}.

Discretizing the Dirichlet condition for the non-planar quad q, one obtains

$$\mathrm{u}^q_D(\mathbf{x}) = \int_0^1 \int_0^1 \mathbf{q}'_{uv} \frac{(\mathbf{q}_{uv} - \mathbf{x}) \cdot \mathbf{N}^q_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|^3} \, \mathrm{d}u \, \mathrm{d}v = \sum_{k=0}^{3} \phi^q_{q_k}(\mathbf{x}) \mathbf{q}'_k$$

with

$$\phi^q_{q_k}(\mathbf{x}) = \int_0^1 \int_0^1 b^k_{uv} \frac{(\mathbf{q}_{uv} - \mathbf{x}) \cdot \mathbf{N}^q_{uv}}{4\pi \|\mathbf{q}_{uv} - \mathbf{x}\|^3} \, \mathrm{d}u \, \mathrm{d}v \tag{3.15}$$

appearing as the coordinate for a quad vertex.

Merging the original computation scheme of GC with the approximate coordinates for bilinear quads results in a deformation function for tri-quad cages:

$$\mathbf{x}' = \sum_{i=1}^{N_\mathbf{C}} \phi_i(\mathbf{x})\mathbf{c}'_i + \sum_{t \in T} \psi_t(\mathbf{x})\sigma_t \mathbf{n}'_t + \sum_{q \in Q} \sum_{k=0}^{3} \psi^k_q(\mathbf{x})\sigma^k_q \mathbf{N}^{q'}_k .$$

Since Thiery and Boubekeur [TB22] compute the coordinates ϕ_{q_k} and ψ^k_q approximately, they use an adaptive Riemann summation strategy on the triangles T of tesselated quads, which is similar to the one used to obtain QMVC. Nonetheless, QGC are guaranteed to be valid, because they reproduce identity. Like for GC, Thiery and Boubekeur [TB22] empirically observe a quasi conformal mapping for QGC. Through the usage of non-planar quads for quasi conformal cage-based deformation, QGC provide good preservation of symmetric features.

Somigliana Coordinates

In order to allow for better control of the volume compression or inflation in cage-based deformation, Chen et al. [CDD23] present Somigliana coordinates (SC). These coordinates are derived from the Somigliana identity [Som85], which is a generalization of Green's third identity. As the Somigliana identity applies to linear elasticity that is associated with the compressibility and the smoothness of deformation. Thus, the derived SC allow for more variations in deformation control than GC. Due to the use of the linear elasticity model, the cage $\mathcal{C}$ is associated with a volumetric homogeneous material that should tend to the state of equilibrium. The state of equilibrium is obtained, if the material's displacement field $\mathbf{u}$ satisfies the Navier-Cauchy equation:

$$\mu \Delta \mathbf{u} + \frac{\mu}{1 - 2\nu} \nabla (\nabla \cdot \mathbf{u}) = \mathbf{b}, \tag{3.16}$$

where $\mu > 0$ is the shear modulus, ν is the Poisson ratio, and $\mathbf{b}$ is the external body load. If no external force is applied ($\mathbf{b} = 0$), the displacement within the volumetric material is only determined by boundary conditions. To set the boundary conditions, one needs to specify boundary displacement $\mathbf{u}$ and the traction $\boldsymbol{\tau}$. The relationship of boundary displacement with the displacement of the body is expressed by the Somigliana identity as follows:

$$\mathbf{u}(\mathbf{x}) = \int_{\partial\Omega} \left[\mathcal{T}(\mathbf{y}, \mathbf{x})\mathbf{u}(\mathbf{y}) + \mathcal{K}(\mathbf{y}, \mathbf{x})\boldsymbol{\tau}(\mathbf{x}) \right] \mathrm{d}a_\mathbf{y}, \quad \forall \mathbf{x} \in \Omega, \tag{3.17}$$

where $\mathcal{K}$ represents the fundamental solutions of linear elasticity, i.e., Kelvinlet [Tho48], and $\mathcal{T}$ represents the boundary traction. Chen et al. [CDD23] combine the expressions for $\mathcal{K}$ and $\mathcal{T}$ with extra terms for volume control to increase the space of available deformations. The extend of these extra control terms is governed by the Poisson ratio ν.

Since SC like GC rely on an identity mapping, the derivation of SC also uses the identity mapping to obtain coordinates for cage vertices and face normals so that a reproduction of a given point $\mathbf{x} \in \Omega$ is achieved. As the identity mapping of displacement $\mathbf{u}(\mathbf{x}) = \mathbf{x}$ describes a state in equilibrium, which satisfies the Navier-Cauchy eq. (3.16), the derivation of SC substitutes the identity mapping into the boundary integral of the Somigliana identity (cf. eq. (3.17)). This yields a reproduction of the rest pose of $\mathbf{x} \in \Omega$:

$$\mathbf{x} = \sum_{i=1}^{N_\mathbf{C}} \mathbf{T}_i(\mathbf{x}) \, \mathbf{c}_i + \sum_{j=1}^{N_\mathbf{F}} \mathbf{K}_j(\mathbf{x})(c \, \mathbf{n}_{t_j}), \tag{3.18}$$

where the constant $c = 2\mu(1+\nu)/(1-2\nu)$ represents the magnitude of the boundary traction induced by $\mathbf{u}(\mathbf{x}) = \mathbf{x}$, and the matrices $\mathbf{T}_i$, $\mathbf{K}_{t_i}$ provide SC defined for cage vertices and faces, respectively. The two matrices $\mathbf{T}_i$ and $\mathbf{K}_{t_i}$ can be computed as boundary integrals:

$$\mathbf{T}_i(\mathbf{x}) = \int_{\partial\Omega} \mathcal{T}(\mathbf{y}, \mathbf{x}) \Gamma_i(\mathbf{y}) \, da_{\mathbf{y}},$$
$$\mathbf{K}_{t_i}(\mathbf{x}) = \int_{\partial\Omega} \mathcal{K}(\mathbf{y}, \mathbf{x}) \Pi_{t_i}(\mathbf{y}) \, da_{\mathbf{y}},$$

where Γ_i represents the piecewise linear basis function for the cage vertex and Π_{t_i} is piecewise constant across the cage triangle t_i. Unfortunately, the derivation of closed-form expressions for the above integrals is technically difficult. For this reason, Chen et al. [CDD23] resort to a quadrature rule to compute SC. The quadrature rule subdivides each triangle into three quads and subsequently applies Gauß-Legendre quadratures on each of the three quads. Thus, the computation of SC can be massively-parallelized, enabling exploitation of the computational power of GPUs.

The fact that SC are matrices instead of scalars is an interesting difference to the other coordinate types. One important property of matrix-valued coordinates is that they are dependent of the scale and orientation of $\mathcal{C}$ at rest state. Consequently, some large cage-based deformation with SC can potentially lead to unintuitive volume inflation artifacts. Therefore, Chen et al. [CDD23] present a corotational formulation of SC to achieve similarity invariance. For computing corotational SC, the rest cage is approximately aligned with the deformed cage through application of a rotation to each cage face. The rotations of cage vertices can be averaged from the face rotations. As a result of the alignment, the volume artifacts are effectively avoided. The details for deriving and computing corotated SC can be found in the paper of Chen et al. [CDD23].

3.7. Isogeometric Analysis

Since the conversion of a CAD model to a discrete volumetric mesh can be laborious, Hughes et al. [HCB05] introduced the isogeometric analysis (IGA) method to solve PDEs directly on a CAD model instead of performing FEM on a volumetric mesh. Cottrell et al. [CHB09] have consulted experts in the numerical analysis field to estimate the potential impact on the industry, which revealed that the creation of suitable meshes frequently accounts for a significant portion of development times. The use of IGA provides several advantages:

- As production teams are able to execute the analysis of the prototype in the the CAD environment, the use of IGA can save the conversion overhead of the prototype into a volumetric mesh for an FEA.
- If the CAD geometry provides an accurate representation of the prototype, the error due to numerical discretization can be reduced with the use of IGA.
- A considerable set of smooth geometries can be represented with few design parameters [Qia10; YHC13], which facilitates quick optimization of prototype designs.

While these advantages encourage to use IGA for VP, some disadvantages complicate its use in common VP processes. One of these disadvantages is that a geometry representation of connected CAD surfaces is not suitable to specify properties, e.g., material or interior loads, for the interior volume of a prototype [LAR20]. For this reason, proper IGA demands a volumetric representation of the prototype such as trivariate NURBS, in order to support numerical simulation of volumetric geometries. The conversion of a CAD geometry to trivariate NURBS requires specialized algorithms [Al +16] and (similar to the generation of an unstructured tetrahedral mesh) cannot guarantee sufficient quality of elements for numerical simulation of complex geometries.

Over the years, the IGA has evolved to support representations other than NURBS. For changing the shape of a prototype immediately after IGA, Burkhart et al. [BHU10] present an approach for IGA on CC subdivision solids coupled with user-guided modeling of the geometry. As CC subdivision solids offer a comprehensive set of modeling operations, users are able to alter the geometry for design adjustments. The IGA approach of Burkhart et al. [BHU10] only supports hexahedral elements, which can be restrictive for modeling sharp surface features. For this reason, Altenhofen [Alt21] extend the approach of Burkhart et al. [BHU10] for better numerical integration of irregular CC solid cells, which allows for more freedom when modeling the geometry with CC subdivision solids. Nonetheless, the use of IGA for volumetric geometries oftentimes demands the conversion of the CAD model into a special volumetric representation, which frequently is restrictive, prone to issues for complex geometries, and involves the use of uncommon data structures.

Although the IGA has received a lot of attention over the years, many VP processes still rely on unstructured volumetric meshes. One major reason for this is the continuous progress of facilitating meshing tasks with robust tools and frameworks (see, e.g., [Hu+18; Hu+20; Dia+23]), which motivates to stick to well-established workflows that use a volumetric mesh. With the continuous improvement of the algorithms exploiting the available computational power (cf. section 3.2), complex meshing tasks can be significantly accelerated. In addition, a transition to IGA oftentimes requires restructuring of well-established workflows. Many production teams avoid this overhead, as long as the well-established workflows do not impose unmanageable development times. This thesis focuses on unstructured tetrahedral meshes to support well-established workflows.

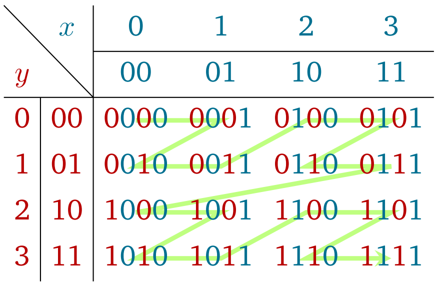
3.8. Spatial Datastructures for GPU-parallel Construction and Rendering

Post-processing with the use of the GPU requires a suitable spatial data structure for interactive rendering. Thus, this section reviews the history on spatial data structures for GPUs. Section 3.8.1 reviews publications that address the massively parallel construction of linear data structures. As the efficient traversal of data structures is also an important topic, section 3.8.2 highlights related work on spatial data structure traversal. The recently introduced RTX technology for hardware accelerated ray tracing is discussed in section 3.8.3.

3.8.1. Linear Spatial Datastructures

An important issue of post-processing unstructured meshes is the fast construction of a spatial data structure to accelerate post-processing. For massively parallel construction, linear spatial data structures arrange primitives linearly in memory using space filling curves. The linear bounding volume hierarchy (LBVH) introduced by Lauterbach et al. [Lau+09] forms the basis of many of the newer approaches.

The LBVH relies on Morton codes [Mor66] (see fig. 3.5) to approximately spatially sort discretized element centroids in parallel. Split positions for the hierarchy are determined in parallel according to differing bits of neighboring codes. While construction is very fast, it results in many singleton nodes, i.e., nodes with only one child, and nodes overlap significantly due to the approximate nature of Morton order sorting. They also introduce a surface area heuristic (SAH) hierarchy that ameliorates these issues, but significantly increases construction cost. In addition, SAH do not map well to volumetric meshes.



y \ x		0	1	2	3
		00	01	10	11
0	00	0000	0001	0100	0101
1	01	0010	0011	0110	0111
2	10	1000	1001	1100	1101
3	11	1010	1011	1110	1111

Figure 3.5.: Morton codes for a 4×4 grid. The bits of the individual integer coordinates are interleaved, leading to a space-filling curve with a fractal Z-like shape.

A number of authors have since improved the LBVH construction performance. Pantaleoni and Luebke [PL10] introduced a two-level hierarchical linear bounding volume hierarchy (HLBVH) Morton sorting method with better performance for dynamic meshes. This method was further improved by Garanzha et al. [GPM11] who introduced a task-based approach to HLBVH construction that enables construction in a single kernel call. Karras [Kar12] developed an approach that computes all hierarchy levels in parallel leading to better scaling, while also generalizing to k-D trees and octrees. This approach was further improved by Apetrei [Ape14] using atomic operations to avoid binary searches and improve construction speed.

A potential alternative to the SAH are the extended Morton codes introduced by Vinkler et al. [VBH17]. Extended Morton codes encode the discretized bounding box size into an arbitrary number of bits of the code while the number of bits per coordinate can be varied as well. However, the position of these bits must be chosen carefully for good performance. Furthermore, high-quality volumetric meshes for simulation tend to vary smoothly in element size (see, e.g., Alliez et al. [All+05]), reducing the potential benefit of extended Morton codes.

For time-varying data, refitting, as used in an early CPU-based work by Wald et al. [Wal+07], is an interesting alternative. However, the resulting BVHs are of lower quality and would loose the beneficial properties of tightly fitting nodes. While Zellmann et al. [ZHL19] use the LBVH data structure for volumetric data, they do so for volumetric data on a sparse regular grid, avoiding overlapping bounding volumes a priori. General unstructured volumetric meshes require a different approach. In particular, octrees are beneficial as volumetric data fills space more densely and octrees help avoid excessive hierarchy depth. For the use case of surface reconstruction from point clouds, Zhou et al. [Zho+11] construct an octree using a bottom up approach based on Morton order sorting. While large point clouds can be stored efficiently, the memory overhead of the hierarchy itself is large and limits tree depth. Furthermore, hierarchy levels are allocated separately, leading to increased allocation and synchronization overhead. Gu et al. [GJG18] introduce the loose octree bounding volume hierarchy (LOBVH) and a binary tree variant loose octree linear bounding volume hierarchy based on the work of Karras [Kar12]. However, their LOBVH approach requires full allocation of the finest octree level followed by a compaction step. This severely limits the maximum octree depth and results in a large memory overhead.

3.8.2. Massively Parallel Traversal

Besides efficient construction, traversal requires special attention on the GPU as it is typically performed recursively and the stack resides in local memory (cf. section 2.2.1), where the compute-to-bandwidth ratio is large. Murguia et al. [Mur+13], García et al. [Gar+14], as well as Binder and Keller [BK16] use bit trails for stackless traversal. Bit trails store the path through a tree as individual bits in an integer. However, these approaches are designed for binary trees and incur additional memory overheads such as allocation of empty nodes or hash maps. Vaidyanathan et al. [VWB19] recently introduced a short stack approach which avoids these overheads and supports higher tree arities, albeit at the cost of requiring some stack space. The traversal algorithm presented in section 7.1.4 relies on the short stack approach by Vaidyanathan et al. [VWB19].

3.8.3. Hardware Accelerated Spatial Data Structure

Since the advent of specialized RT cores in consumer NVIDIA GPUs for hardware-accelerated ray tracing operations, the real-time capabilities for GPU-accelerated ray tracing significantly improved. RT cores can be controlled with NVIDIA's RTX platform (see, e.g., [Sti18; SFG20; NVI24b]).

For finding intersections of rays with the scene, the RTX platform provides a built-in hardware-accelerated spatial data structure. As the RTX platform is designed for ray tracing of complex scenes, the spatial data structure management is designed for maintaining many geometries in a scene. Typically, some geometries of a scene exhibit fine-grained geometric details, while other parts of the scene only feature low-resolution details or are even empty. Therefore, the RTX platform provides the bottom level acceleration structure (BLAS) for the high-resolution geometries of the scene and the top level acceleration structure (TLAS) to subdivide the scene into BLASes. While a BLAS constructs a high-quality structure for many primitives, building a TLAS is substantially more economical in terms of run time and memory consumption [Sti18].

On the contrary to the RTX platform, this thesis focuses on post-processing of unstructured tetrahedral meshes for analysis purposes, where typically only one geometry is inspected instead of a complete scene. Due to its origin in ray tracing dynamic scenes, the RTX platform is rather focused on surface meshes instead of volumetric meshes. In addition, the availability of RT cores is currently limited to only the newest GPUs from a single vendor. Thus, chapter 7 presents a spatial data structure the OLBVH for memory-efficient DVR of a single volumetric mesh. The OLBVH is not only focused on rendering and supports other use cases such as re-meshing. In order to support arbitrary GPUs, the construction of the OLBVH does not depend on RT cores. Furthermore, efficient hardware construction of spatial data structures is itself an open research topic (see, e.g., Doyle et al. [DTM17] or Viitanen et al. [Vii+17; Vii+18]).

3.9. Direct Volume Rendering of Unstructured Meshes

Pioneering works in the GPU-based DVR of structured grids [KW03] such as voxel grids inspired many publications about the GPU-based DVR of unstructured grids [Sil+05] such as unstructured tetrahedral meshes. Typically, the rendering of unstructured volumetric meshes imposes slower run time performance than the rendering of structures volumetric grids because of the additional overhead of performing a spatial search for finding the primitives of the mesh that intersect a specific ray. The majority of current DVR methods relies on a spatial data structure to quickly reduce the set of potentially intersecting mesh elements [Mei+21].

As the aggregated processing power of GPUs is attractive for interactive DVR performance, many early approaches [Wyl+02; KSE04; Mui+11] use programmable GPU shaders for ray casting. The performance of these DVR approaches degrades with increasing the number of samples for each view ray, while more samples typically improve the quality of the resulting images (cf. section 2.4). Therefore, academics introduced adaptive sampling methods [Kra+07] that adapt the sampling rate to the local variance of the sampled scalar field, in order to reduce the number of samples for improved DVR performance. Due to the limited available memory of each GPU shader and the restrictions of shader programming, later approaches [Max+08; OG14; Lar+15] relied on APIs such as CUDA (cf. section 2.2.1) to exploit convenient and versatile programming environments, which provides extendable DVR applications and good image quality.

With further increases in processing power of GPUs, people started to prefer the convenient and extendable API-based DVR applications over the restrictive GPU shader variants. Consequently, interactive DVR approaches using CUDA were devised by people in the industry and academics. Wald et al. [Wal+19] trace very short segments of rays to perform point location in tetrahedral meshes using the RTX acceleration structure (cf. section 3.8.3) for triangle meshes, in order to benefit from the capabilities of built-in hardware acceleration capabilities of NVIDIA cards. This requires the conversion of tetrahedra into triangles. Due to hardware acceleration, they achieve significant speedups compared to a pure general purpose computing on graphics processing units approach, although their method performs an intersection test for a segment of the ray instead of direct point location for each sampling point. Their approach also extends to bilinear elements [Mor+22]. As substantial performance improvements can be achieved with adaptive sampling, Morrical et al. [Mor+19] dynamically adapt the sampling rate to the local variance of the transfer function, which significantly accelerates the approach of Wald et al. [Wal+19]. For determining the local variance, they partition the input mesh into sub-regions and calculate the variance for each region. The non-overlapping leaves of an additional KD-tree spatial data structure form the partitions of the mesh, which requires additional construction and memory overheads.

While interactive rendering performance coupled with convenient programmability is provided by present-day GPU technologies, the memory capacity on a GPU is limited, which poses problems for rendering large meshes or entire data sets. For this reason, recent work addresses the memory-efficient organization of meshes and annotated data for interactive DVR. Wald et al. [WMZ21] present an encoding of the mesh and the BVH for reducing the memory demands of DVR of unstructured meshes. This encoding represents the input mesh as a set of sub-meshes. Partitioning a mesh into sub-meshes admits to represent each sub-mesh with fewer indices and use fewer bytes per index. For further reduction of memory consumption, Şahistan et al. [Şah+21] present a memory layout for vertex indices that includes an exclusive-or-sum field to calculate vertex indices from other indices. With the use of this field, less indices need to be stored.

Besides compression of the mesh, the compression of the spatial data structure is an effective measurement to reduce memory consumption. For tree compression, Morrical et al. [Mor+23] sort primitives along the space-filling Hilbert curve to arrange clusters of spatially nearby primitives. The spatial data structure construction can then generate a hierarchy of clusters instead of a hierarchy of individual primitives, which saves memory. Since meshes for numerical simulation may exhibit a more fine-grained resolution at the features of interest, there is a need for memory-efficient DVR of meshes generated by adaptive mesh refinement. As these meshes may exhibit deep and highly-irregular hierarchies of grid cells at the fine-grained parts, Wald et al. [Wal+21] combines several same-level grid cells into non-overlapping bricks. For each of these bricks, the regions of mutual spatial influence for rendering are determined. An RTX acceleration structure is then built for DVR. Similarly, Zellmann et al. [Zel+23] support cell-centric scalar value annotation for DVR of meshes generated by adaptive mesh refinement. Their approach relies on a dual mesh for proper interpolation of scalars. Inspired by brick-based approach of Wald et al. [Wal+21], they cluster the dual mesh into small gridlets that combine several hexahedron grid cells. Organizing the hierarchy of refined cells into clusters of gridlets instead of individual grid cells saves memory consumption.

In order to enable memory-efficient DVR of an unstructured tetrahedral mesh representing a virtual prototype, section 7.1 presents a spatial data structure that provide sparse memory usage and control over the hierarchy depth. For many meshes, the proposed spatial data structure consumes less memory than the RTX acceleration structure (see section 7.4.2). The DVR with the proposed spatial data structure is sufficient for interactivity for reasonably large meshes and keeps up well with state of the art methods. This proposed spatial data structure does not depend on RTX hardware acceleration. Thus, the proposed spatial data structure is more general and applicable to more use cases of volumetric rendering, because it does not require to convert a tetrahedron into triangles. While the usage of RTX hardware acceleration provides superior rendering performance, the proposed spatial data structure admits fast run time performance of ray marching on GPUs without RTX (see section 7.4.3), which is majority of commodity GPUs.

4. Fast Harmonic Mesh Optimization

The following papers contain the core content of this chapter:

[Str+22] **D. Ströter**, J. Mueller-Roemer, D. Weber, D. W. Fellner, "Fast harmonic tetrahedral mesh optimization". In: *The Visual Computer* (June 2022). **Visual Computer Best Paper Award at Computer Graphics International 2022**. DOI: 10.1007/s00371-022-02547-6

[Str+23] **D. Ströter**, A. Halm, U. Krispel, J. S. Mueller-Roemer, D. W. Fellner, "Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation". In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Gerd Wagner, Frank Werner, Tuncer Oren, and Floriano De Rango. Springer International Publishing, 2023, pp. 24–42. DOI: 10.1007/978-3-031-23149-0_2

As this thesis is concerned with editing unstructured tetrahedral meshes for VP processes, the element quality of the meshes should be sufficient for numerical simulation. Therefore, the first topic this thesis addresses is element quality optimization. This chapter presents a massively parallel algorithm to optimize the element quality of an unstructured tetrahedral mesh performing vertex relocation and re-meshing with edge/face flips (see section 2.1.2). For this purpose, this chapter investigates RQ1 to find parallelization strategies of foundational operations for mesh optimization, addressing the field of low-level mesh editing (cf. section 3.1). Since the robustness of mesh optimization is important for VP processes, this chapter investigates RQ2 as well. Thus, the target of the proposed optimization should not only be good run time performance but also to design the algorithms such that they produce valid meshes (cf. section 2.1.1) with good element shape quality. Consequently, this chapter investigates not only parallelization strategies but also attempts to contribute algorithms that provide robust and fast convergence to a mesh with good element qualities. The intention behind this approach is to obtain robust and massively parallel algorithms for meshing operations.

In orchestration, these operations can be used for optimization or other re-meshing applications in the VP cycle. Therefore, this chapter is concerned with the investigation of RQ3. The foundation for the meshing algorithms proposed in this chapter is the harmonic triangulation (cf. section 2.1.3), in order to provide a contribution to Delaunay-based methods. The choice of harmonic triangulations for mesh optimization is relevant because of the following properties:

- It provides an efficiently calculable energy for minimization
- This energy is a measure for element quality and differentiable
- It tends to provide low element counts saving computation time of numerical methods
- It is Delaunay-related opening the gate to Delaunay-based methods
- Its gradient can be computed using only face areas, normals and the Jacobian (see section 4.1)

To achieve efficient use of first-order optimization, this chapter first introduces an efficient scheme for gradient computation in section 4.1. As mesh optimization requires boundary treatment (cf. section 3.2.7), section 4.2 provides a method for massively parallel boundary extraction. Section 4.3 presents a method for robust, massively parallel optimization of vertex positions. Section 4.4 presents a strategy to massively parallelize harmonic flips. How to combine the relocation of vertices with harmonic flipping for good convergence and run time performance is discussed in section 4.5. A critical evaluation of the proposed concepts appears in section 4.6. Finally, section 4.7 briefly summarizes the key conclusions of this chapter.

4.1. Harmonic Gradient

As the goal in harmonic triangulations is energy minimization, the gradient of the energy is useful for first order methods such as gradient descent. Alexa [Ale19] details the calculation of the gradient for one tetrahedron (cf. eq. (2.14)). Harmonizing a mesh, i.e., minimizing Dirichlet energy, substantially benefits from the use of gradient descent. The gradient of the trace used for mesh harmonization is denoted as the harmonic gradient, henceforth. The sum of all the harmonic gradients of tetrahedra assembles the harmonic gradient of the entire mesh:

$$\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}}) = \sum_{\tau \in \mathrm{T}} \frac{\partial \operatorname{tr}(\mathbf{L}_{\tau})}{\partial \mathbf{X}_{\mathrm{T}}} \in \mathbb{R}^{d N_{\mathrm{V}}}, \tag{4.1}$$

where $\partial \operatorname{tr}(\mathbf{L}_{\tau}) / \partial \mathbf{X}_{\mathrm{T}}$ only contains a nonzero entry for vertices included by τ. Thus, $\partial \operatorname{tr}(\mathbf{L}_{\tau}) / \partial \mathbf{X}_{\mathrm{T}}$ is a global version of the local harmonic gradient $\partial \operatorname{tr}(\mathbf{L}_{\tau}) / \partial \mathbf{X}_{\tau}$ for a simplex τ. As the gradient of each vertex $\mathbf{v} \in \mathrm{V}$ depends on multiple tetrahedra, assembly of the global gradient performs several additions for every vertex, which is part of more than one tetrahedron. Therefore, parallel assembly of the gradient over simplices $\tau \in \mathrm{T}$ requires to handle potential write conflicts to ensure deterministic and correct results. Another drawback of using a global harmonic gradient for the entire mesh is that the global gradient becomes invalid, when only one single vertex of the mesh changes or any re-meshing operation alters the mesh. Furthermore, gradient descent methods typically exhibit better convergence, if performed locally. Relocating two adjacent vertices at a time can mitigate convergence. For this reason, it is preferable to perform local gradient descent steps for individual vertices. In order to avoid global assembly and enable calculation of the local gradient for a given vertex, a formula for the gradient of the trace for a specific vertex provides means for local gradient descent steps. A formula of the local harmonic gradient for a specific vertex $\partial \operatorname{tr}(\mathbf{L}_{\tau}) / \partial \mathbf{v}$ can be extracted from the gradient formula for a simplex in eq. (2.14). This results in the following gradient formula:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_{\tau})}{\partial \mathbf{v}_{t_i}} = \frac{a_i}{d! d^3 v_{\tau}} \left(\left(\frac{2a_i^2}{v_{\tau}} - \eta(\tau) \right) \mathbf{n}_i + \frac{2}{v_{\tau}} \sum_{\substack{j=0 \\ j \neq i}}^{d} a_j^2 (\mathbf{n}_i^{\mathsf{T}} \mathbf{n}_j) \mathbf{n}_j \right), \quad \text{where} \quad t_i \in \tau. \tag{4.2}$$

Proof. Suppose the vertex $\mathbf{v} \in \mathrm{V}$ has the index $t_i \in \tau$, where $i \in \{0, 1, \dots, d\}$. The harmonic gradient of $\mathbf{v}$ can then be obtained by evaluating eq. (2.14) and taking the i-th entry of the resulting vector of gradients. The idea is now to formulate an equation for $\partial \operatorname{tr}(\mathbf{L}_{\tau}) / \partial \mathbf{v}$ by rewriting eq. (2.14) to only consider the vertex $\mathbf{v}$ of index $t_i \in \tau$. For convenience, recalling eq. (2.14) provides the following:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_{\tau})}{\partial \mathbf{X}_{\tau}} = \frac{1}{d!} (\mathbf{X}_{\tau} \mathbf{M}_d)^{-\mathsf{T}} \mathbf{M}_d^{\mathsf{T}} (\operatorname{tr}(\mathbf{L}_{\tau}) \mathbf{I} - 2\mathbf{L}_{\tau}).$$

From [Ale19] it is known that:

$$(\mathbf{X}_{\tau} \mathbf{M}_d)^{-\mathsf{T}} \mathbf{M}_d^{\mathsf{T}} = \frac{-1}{d v_{\tau}} (a_0 \mathbf{n}_0, \dots, a_d \mathbf{n}_d), \tag{4.3}$$

$$(\mathbf{L}_{\tau})_{ij} = \frac{a_i a_j}{d^2 v_{\tau}} \mathbf{n}_i^{\mathsf{T}} \mathbf{n}_j. \tag{4.4}$$

Inserting eq. (4.3) and eq. (2.11) in eq. (2.14) results in:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{X}_\tau} = \frac{-1}{d! d v_\tau}(a_0 \mathbf{n}_0, \ldots, a_d \mathbf{n}_d)\left(\frac{\sum_{j=0}^{d} a_j^2}{d^2 |v_\tau|} \cdot \mathbf{I} - 2\mathbf{L}_\tau\right). \tag{4.5}$$

The next step is to develop the matrix on the right side of the product. While the non-diagonal entries of the matrix are given by eq. (4.4), the diagonal entries require the evaluation of the subtraction with the trace:

$$\left(\frac{\sum_{j=0}^{d} a_j^2}{d^2 |v_\tau|} \cdot \mathbf{I} - 2\mathbf{L}_\tau\right)_{ii} = \frac{1}{d^2}\left(\frac{\sum_{j=0}^{d} a_j^2}{|v_\tau|} - \frac{2}{v_\tau} a_i^2\right), \tag{4.6}$$

$$\left(\frac{\sum_{j=0}^{d} a_j^2}{d^2 |v_\tau|} \cdot \mathbf{I} - 2\mathbf{L}_\tau\right)_{ij} = -2\frac{a_i a_j}{d^2 v_\tau} \mathbf{n}_i^\mathsf{T} \mathbf{n}_j, \quad \text{where} \quad i \neq j. \tag{4.7}$$

As the goal is to obtain the gradient regarding **v** of index $t_i \in \tau$, the dot product of the left row vector and the i-th column vector of the right matrix in eq. (4.5) is of interest now. Since the entries of the matrix are given by eqs. (4.6) and (4.7), the dot product evaluates to:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{v}_{t_i}} = \frac{-1}{d! d v_\tau}\left(a_i \mathbf{n}_i \cdot \frac{1}{d^2}\left(\frac{\sum_{j=0}^{d} a_j^2}{|v_\tau|} - \frac{2}{v_\tau} a_i^2\right) + \sum_{\substack{j=0 \\ j \neq i}}^{d} a_j \mathbf{n}_j \cdot (-2)\frac{a_i a_j}{d^2 v_\tau} \mathbf{n}_i^\mathsf{T} \mathbf{n}_j\right). \tag{4.8}$$

As a result, eq. (4.8) provides a formula for the calculation of $\partial \operatorname{tr}(\mathbf{L}_\tau)/\partial \mathbf{v}$. Now, it still remains to simplify eq. (4.8). Interestingly, the harmonic index η (see eq. (2.13)) is part of the equation:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{v}_{t_i}} = \frac{-1}{d! d v_\tau}\left(a_i \mathbf{n}_i \cdot \frac{1}{d^2}\left(\eta(\tau) - \frac{2}{v_\tau} a_i^2\right) + \sum_{\substack{j=0 \\ j \neq i}}^{d} a_j \mathbf{n}_j \cdot (-2)\frac{a_i a_j}{d^2 v_\tau} \mathbf{n}_i^\mathsf{T} \mathbf{n}_j\right). \tag{4.9}$$

The resulting equation can be further simplified by rearrangements:

$$\frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{v}_{t_i}} = \frac{a_i}{d! d^3 v_\tau}\left(\left(\frac{2a_i^2}{v_\tau} - \eta(\tau)\right)\mathbf{n}_i + \frac{2}{v_\tau}\sum_{\substack{j=0 \\ j \neq i}}^{d} a_j^2 (\mathbf{n}_i^\mathsf{T} \mathbf{n}_j)\mathbf{n}_j\right).$$

After the rearrangements, the equation is the same as eq. (4.2). □

Using the gradient from eq. (4.2), a thread can efficiently calculate the direction of steepest descent for any vertex given its set of adjacent tetrahedra. This allows for optimization of vertex positions, calculating gradients on demand without the need of global gradient assembly. Furthermore, it is an interesting observation that the gradient can be calculated as a linear combination of area weighted face normals. For efficiency, it is beneficial that the gradient calculation requires only the determinant of the simplex as well as face normals and determinants, while costly functions are not part of the formula. Another interesting aspect of eq. (4.2) is that its linear form allows for geometric interpretation of the harmonic gradient, which is an interesting avenue for future work, because it might reveal insights about the nature of harmonic triangulations.

4.2. Boundary Features Extraction for Gradient Descent

As re-meshing unstructured tetrahedral meshes and gradient descent of boundary vertices, requires access to the boundary elements of the mesh, fast extraction of the triangular surface mesh is a necessity for fast run time performance. Therefore, this thesis presents a massively parallel algorithm for quick extraction of the triangular surface coupled with computation of surface features and mapping data from a tetahedral mesh to its surface mesh and vice versa. The algorithm computes the data structure presented in listing 4.1:

```
struct {
        int num_boundary_triangles;
        int num_boundary_vertices;
        float3 * boundary_vertices;
        int3 * boundary_triangles;
        uchar * vertex_markers;
        int * tet_mesh_idx_to_boundary_idx;
        int * boundary_idx_to_tet_mesh_idx;
} TetMeshBoundary;
```

Listing 4.1.: The data structure representing the boundary of an unstructured tetrahedral mesh. The listing is provided in a C++ style syntax.

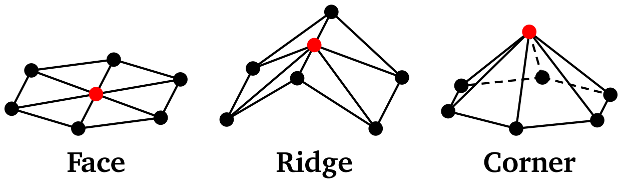


Figure 4.1.: Classification of vertices (red) into types to optimize vertices on the boundary.

The data structure saves the numbers of boundary vertices and triangles as well as the surface mesh itself as arrays of three contiguous floats and integer indices per vertex and triangle, respectively. Each vertex of the tetrahedral mesh is annotated with a flag to indicate, if it is a boundary vertex or not. If the vertex is a boundary vertex, the boundary data structure additionally encodes whether the vertex lies on a geometrical face, ridge, or corner. Figure 4.1 visualizes these three cases. This enables downstream element quality optimization algorithms to optimize boundary vertices without changing the shape. For mapping purposes, the `tet_mesh_idx_to_boundary_idx` array includes the boundary vertex index for each vertex index of the tetrahedral mesh. By construction, this array only includes valid values for vertices that are boundary vertices, i.e., whose markers indicate a boundary vertex. For each boundary vertex, the `boundary_idx_to_tet_mesh_idx` includes the corresponding index in the tetrahedral mesh.

The proposed algorithm extracts the triangular boundary of the tetrahedral mesh using GPU-accelerated sub-mesh extraction similar to the method presented by Wald [Wal21]. A key difference to Wald's method is that the proposed algorithm is able to map from the vertex of the extracted mesh to the corresponding vertex in the original mesh. The first step is to detect the boundary triangles as well as boundary vertices. With the use of the TCSR data structure for simplex meshes (cf. section 2.2.3), boundary extraction finds the triangles of the tetrahedral mesh as well as the following two mappings in parallel on the GPU:

1. The mapping of any given triangle to the tetrahedra containing this triangle

2. The mapping of any given vertex to the triangles containing this vertex

In terms of mesh connectivity, a boundary triangle is connected to only one tetrahedron. Thus, the algorithm initializes a pair of marker arrays to 0 for each triangle and vertex. The boundary extraction algorithm checks the number of neighboring tetrahedra of each triangle in parallel. The check marks boundary triangles and their vertices with the marker value 1. Subsequently, the algorithm performs a parallel prefix scan on the marker array in order to obtain offset positions as well as `num_boundary_triangles` and `num_boundary_vertices`. As the prefix sum over the boundary vertex markers can be used to map a boundary vertex index to the index in the tetrahedral mesh, the algorithm keeps this prefix sum in the `tet_mesh_idx_to_boundary_idx` array. Using the prefix offset positions, a stream compaction writes boundary vertices to the `boundary_vertices` array and boundary triangles to the `boundary_triangles`. Unlike Wald's method [Wal21], the proposed algorithm does not require a permutation array to update the indices, because the prefix positions are sufficient for a simple copy. After determining the boundary vertices, the boundary extraction algorithm classifies them into one of the following three categories (see fig. 4.1):

- **FACE**: The boundary triangles surrounding the vertex are coplanar, forming a geometric face.
- **RIDGE**: The boundary triangles surrounding the vertex form a geometric edge.
- **CORNER**: The vertex is neither classified as FACE nor EDGE.

Classification of boundary vertices proceeds in parallel on the GPU. Optimization tools such as Stellar [KS08] evaluate the scalar products of the normals and check if they differ by a given threshold. The eigenvalue decomposition based method by Jiao [Jia06] achieves even more robust classification. The following describes a classification based on the method in Stellar, while the classification based on the method of Jiao [Jia06] is discussed in section 6.1.2.

The classification first retrieves the tetrahedral mesh index of the boundary vertex with the subsequently calculated mapping. The algorithm classifies the boundary vertices based on the scalar products of the normalized normals of surrounding boundary triangles. If two normals are co-linear, their scalar product evaluates to 1. Feature classification relies on the threshold ε_F. For efficiency, the classification associates a group of triangles with one normal. An initial group is associated with the normal of the first triangle in the list. Subsequently, the classification iterates through the surface triangles of a boundary vertex. Whenever the scalar product of a triangle's normal and the normal of any group deviates from 1 by more than ε_F, a new group is added. This new group is associated with the normal of the current triangle. For a face vertex, the classification finds only one group of triangles. For a ridge vertex, the classification finds two groups. If the classification finds more than two groups, the vertex is a corner vertex. This thesis consistently chooses $\varepsilon_F = 0.01$.

In a final parallel pass over the vertices, the algorithm sets the `boundary_idx_to_tet_mesh_idx` mapping. The algorithm only needs to check the boundary marking of the vertex and lookup the boundary vertex index in the `tet_mesh_idx_to_boundary_idx` array in order to write the tetrahedral mesh vertex index to the entry at the boundary vertex index position.

4.3. Vertex Relocation

Optimizing vertex positions is a crucial operation to improve element quality [KS07; Lo14a]. Therefore, this section presents algorithms for robust and massively parallel vertex relocation using the harmonic gradient from section 4.1.

4.3.1. Relocating Interior Vertices

An important action for improving element shape is relocating vertices minimizing $\mathrm{tr}(\mathbf{L}_\mathrm{T})$. This section describes vertex relocation using the harmonic energy in general, while the boundary treatment is discussed in sections 4.3.2 to 4.3.4.

As mesh optimization can be significantly accelerated through parallel processing, a parellelization strategy shall allow for fast optimization using GPUs. Since individual optimization of vertex positions in Gauß-Seidel iterations is more stable than simultaneous vertex relocation in Jacobi iteration order, the parallelization strategy shall consider the connectivity of vertices so that simultaneous updates of adjacent vertices is avoided. Consequently, a graph coloring method assigns a color to every mesh vertex respecting the condition that adjacent vertices receive different colors. Figure 4.2 visualizes the simultaneous optimization of vertices using graph coloring. A multitude of efficient massively-parallel graph coloring methods is available. Frequently applied methods are for example Deveci's graph coloring [Dev+16] or the graph coloring algorithm of NVIDIA's Cusparse library [NVI22c] implementing the Cohen-Castonguay algorithm [CC12; NCC15]. The graph coloring of Cusparse is deterministic, while Deveci's graph coloring [Dev+16] is usually faster but not deterministic. In order to obtain independent sets, vertices of the input mesh are colored as a pre-process and whenever mesh connectivity is altered. The set $\mathcal{S}_C$ denotes the sets of vertices per color.

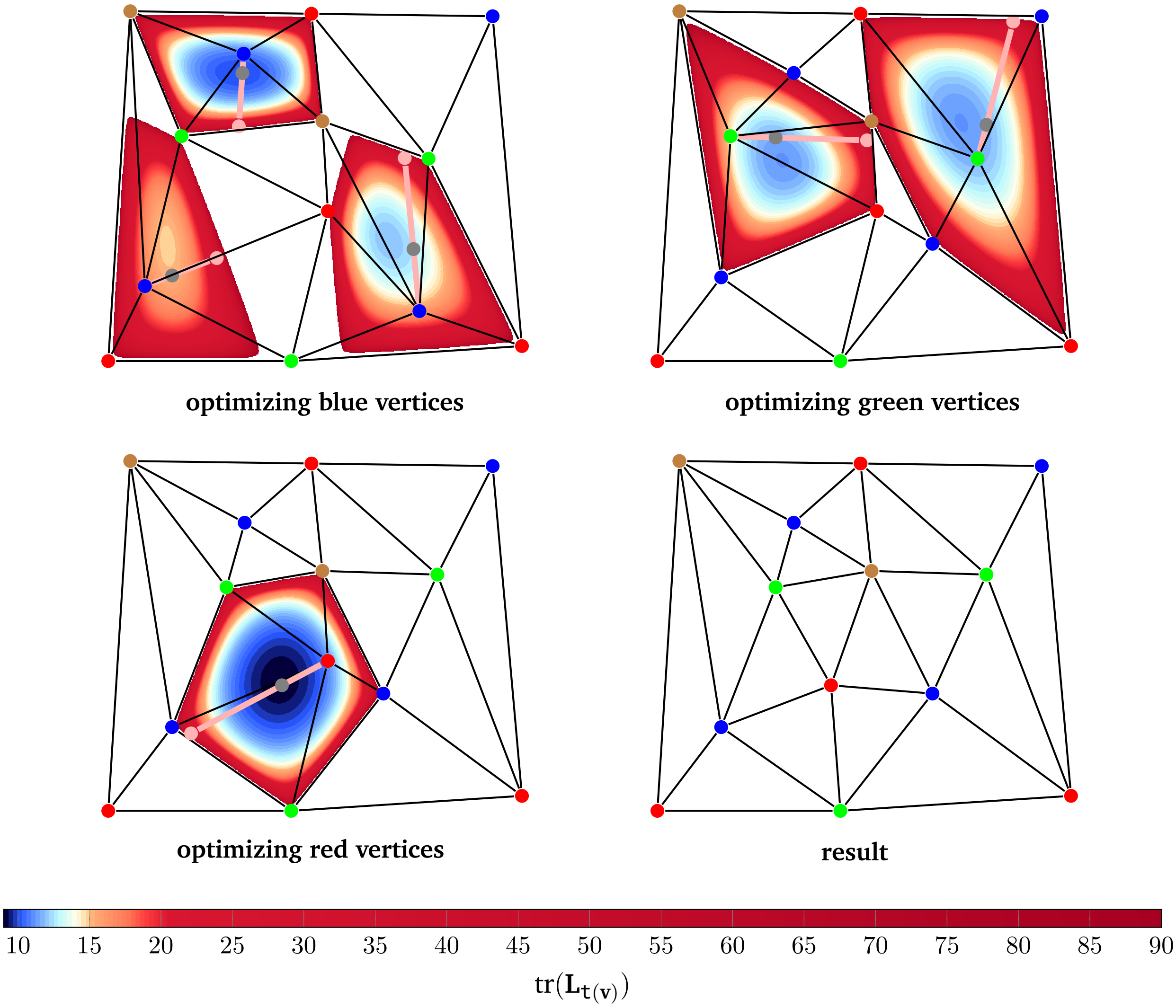


Figure 4.2.: Optimizing interior vertices with coloring. The figure shows the energy fields of $\mathrm{tr}(\mathbf{L}_{\mathbf{t}(\mathbf{v})})$ for the to-be-optimized vertices, the directions of steepest descent bounded by inversion-free intervals, and the minima (gray) along the search lines.

Algorithm 2 presents an outline of the massively-parallel vertex relocation. For now, the boundary treatment conditions $\mathrm{T} = \mathcal{B}$ and $\mathrm{T} \cong \mathcal{B}$ are ignored, because this section only details the general procedure for vertex relocation. The massively-parallel vertex relocation on the basis of graph coloring optimizes vertex positions in batches of independent sets (lines 2 and 3). In order to achieve a robust vertex relocation, element inversions, i.e. $v_\tau < 0$, shall be avoided. For this reason, a factor λ bounds the vertex relocation along the direction of steepest descent. The upper bound λ is computed after determining the relocation direction and is, thus, initialized to ∞ (line 4). As it is important to consider every simplex that includes the relocated vertex, the gradient calculation considers every simplex from the one-ring of simplices. For any vertex $\mathbf{v}_i \in \mathrm{V}$ of index i, $\mathtt{t}(i)$ denotes the simplices in the one-ring of $\mathbf{v}_i$:

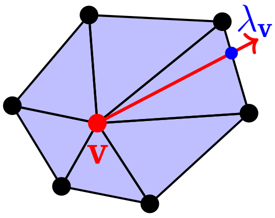
$$\mathtt{t}(i) = \{\tau \in \mathrm{T} \mid i \in \tau\}, \quad \text{where} \quad 0 \le i < N_{\mathrm{V}}. \tag{4.10}$$

The notation $\mathtt{t}(\cdot)$ shall indicate that the one-ring is also a sub-triangulation of T. Applying the sum rule, the vertex position optimization calculates the harmonic gradient for the sub-triangulation $\mathtt{t}(\cdot)$. For convenience, the following definition denotes the harmonic gradient of the one-ring:

$$\nabla_{\mathbf{v}} \operatorname{tr}(\mathbf{L}_{\mathtt{t}(i)}) = \sum_{\tau \in \mathtt{t}(i)} \frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{v}}, \quad \text{where} \quad \mathbf{v} = \mathrm{V}_i. \tag{4.11}$$

The TCSR mesh data structure allows for massively-parallel precomputation of the connectivity relationships of vertices to their one-rings by calculation of the corresponding coboundary operator, e.g., d_3^2 (see section 2.2.3). Thus, it can be used to obtain $\mathtt{t}(\cdot)$ for all the vertices of the mesh. Iterating over the simplices in $\mathtt{t}(\cdot)$ and adding their gradients (together) determines the harmonic gradient of the one ring (line 8).

As gradient descent methods converge if the gradient approaches zero, the optimization checks if the gradient norm is lower than a predetermined threshold ε_g to avoid unnecessary steps (line 14). After considering the gradient norm for the termination check, the procedure inverts and normalizes the gradient to prepare for a line search along the direction of steepest descent (line 15). The line search shall be robust preventing element inversions. For this end, the intention is to determine a value for the upper bound λ such that the relocation along the search direction is bounded to an interval without element inversions. Hence, the interval $[0, \lambda]$ is called inversion-free interval. The determination of the inversion-free interval performs root finding like the method of Smith and Schaefer [SS15] for 2D triangles. The key idea is to start from an initial value for λ and perform a binary search until no inversion within the one ring occurs.



A visualization of the determination of the initial value for λ appears in the inset to the right. As an inversion occurs, when a vertex $\mathbf{v}$ of a simplex τ passes the half space defined by the opposing face $\tau \setminus \mathbf{v}$, an intersection test of the half space and the search direction yields the maximal step size $\lambda_{\mathbf{v}}$ without an inversion. Finding the minimal step size for all simplices in the one-ring provides λ_{init}, which is an upper bound for an inversion (line 16). This bound lies within the convex interior of the one-ring, since the intersection test considers half spaces and is thereby not limited to the boundary faces of $\mathtt{t}(i)$. If the one-ring forms a concave sub-triangulation, this property is important, because inversions may occur for vertex positions inside $\mathtt{t}(i)$. The next step is to ensure the absence of inversions for the interval $[0, \lambda]$. For this purpose, a binary search is performed until every determinant of a simplex is larger than a predetermined threshold ε_v (lines 20 to 27). In order to avoid unnecessary iterations of binary search, the initial value for λ is scaled by a constant $\mu \in [0, 1)$ beforehand (line 18). The implementation used for the experiments in this thesis was sufficiently robust for choosing $\mu = .95$ (see section 4.6.3).

Finally, a bracketing method chooses an optimal step size α within the inversion-free interval $[0, \lambda]$ (line 29). Like Alexa [Ale19], this thesis relies on the robust and quickly converging Brent's method [Pre+02]. However, this optimization algorithm optimizes vertex positions locally, while Alexa [Ale19] performs global gradient descent steps.

4.3.2. Directional Derivatives for Boundary Treatment

As optimization of only interior vertices is not sufficient to achieve high-quality tetrahedral meshes, a mesh optimization tool requires an approach to treat the boundary of a mesh. The relocation of boundary vertices can deform the shape of the model, which potentially leads to a significant loss of boundary details. For this reason, many mesh optimization methods either subdivide boundary elements with subsequent costly backtracking or perform an apples-to-oranges comparison of surface distance to element quality improvement (see section 3.2.7). In this thesis, the intention is to relocate vertices on the boundary in an efficient massively-parallel manner and avoid reprojection of boundary vertices, because a reprojection step does not respect convergence and potentially degrades element quality. In order to relocate vertices along the surface, while at the same time respecting convergence, this thesis explores the use of directional derivatives for mesh

Algorithm 2 Parallel vertex relocation algorithm.

```
procedure RELOCATEVERTICES($\mathcal{M} = (\mathrm{T}, \mathrm{V})$, $\mathcal{S}_C$)
    for all $c \in \mathcal{S}_C$ do                                   ▷ Go through independent sets
        for all $i \in c$ do                                               ▷ In parallel
            $\mathbf{v} \leftarrow \mathrm{V}_i$; $\lambda \leftarrow \infty$
            if subject to $\partial\mathrm{T} = \mathcal{B} \wedge \mathbf{v} \in \mathrm{V}_C$ then
                continue
            end if
            $\mathbf{g} \leftarrow \nabla_{\mathbf{v}} \operatorname{tr}(\mathbf{L}_{\mathtt{t}(i)})$                                   ▷ See eq. (4.11)
            if subject to $\partial\mathrm{T} = \mathcal{B} \wedge (\mathbf{v} \in \mathrm{V}_{\mathcal{F}} \vee \mathbf{v} \in \mathrm{V}_{\mathcal{R}})$ then
                $\mathbf{g} \leftarrow$ TANGENTSUBSPACEDERIVATIVE($\mathbf{v}$, $\mathbf{g}$)
            else if subject to $\partial\mathrm{T} \cong \mathcal{B} \wedge \mathbf{v} \in \partial\mathrm{V}$ then
                $\mathbf{g} \leftarrow$ FINDTANGENTDERIVATIVE($\mathbf{v}$, $\lambda$, $\mathbf{g}$)
            end if
            if $\|\mathbf{g}\| < \varepsilon_g$ then continue end if
            $\mathbf{g} \leftarrow -\frac{\mathbf{g}}{\|\mathbf{g}\|_2}$
            $\lambda_{\text{init}} \leftarrow \underset{\tau \in \mathtt{t}(i)}{min} \{t \mid t = \text{intersect-ray-plane}(\mathbf{g}, \mathbf{v}, \tau \setminus \mathbf{v})\}$
            $\lambda \leftarrow min(\lambda_{\text{init}}, \lambda)$
            $\lambda \leftarrow \lambda \cdot \mu$                             ▷ Prevent division by zero at upper bound
            search $\leftarrow$ true
            do
                $\mathrm{V}_i \leftarrow \mathbf{v} + \lambda\mathbf{g}$
                search $\leftarrow$ is $v_\tau < \varepsilon_v$ for any $\tau \in \mathtt{t}(\mathbf{v})$
                if subject to $\partial\mathrm{T} \cong \mathcal{B}$ then
                    search $\leftarrow$ is $\mathbf{v}$ still on boundary primitive?
                end if
                if search then $\lambda \leftarrow \lambda/2$ endif
            while search
            $\mathrm{V}_i \leftarrow \mathbf{v}$
            $\alpha \leftarrow bracketing(\mathbf{v}, \mathbf{g}, [0, \lambda], \operatorname{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{v})}))$        ▷ Implementation uses [Pre+02]
            $\mathrm{V}_i \leftarrow \mathbf{v} + \alpha\mathbf{g}$
            if subject to $\partial\mathrm{T} \cong \mathcal{B}$ then
                IDENTIFYBOUNDARYSTATE($\mathbf{v}$)
            end if
        end for
    end for
end procedure
```

optimization. With the directional derivative it is possible to optimize on a subspace of the domain Ω. As a result, the directional derivative of a functional, e.g. harmonic energy (cf. section 2.1.3), can be computed for tangent planes or lines of the mesh. Relocating the directional derivative along tangent planes or lines allows for both full boundary preservation detailed in section 4.3.3 and approximate boundary preservation detailed in section 4.3.4. The remainder of this section shows how directional derivatives can be calculated for an optimization functional.

The boundary vertices of a mesh form the set $\partial \mathrm{V} \subseteq \mathrm{V}$. Let $\mathbf{p}$ be a tangent plane on the boundary of the mesh with linearly independent unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2$. It is sufficient to detail the calculation for tangent planes, because for a tangent line one only needs to drop $\mathbf{u}_2$ and perform the analog steps. As the goal is to relocate boundary vertices along a tangent subspace, it is further assumed that $\mathbf{p}$ includes the boundary vertex:

$$\mathbf{p}(t,s) = \mathbf{v} + t\,\mathbf{u}_1 + s\,\mathbf{u}_2, \quad \text{where} \quad \mathbf{v} \in \partial \mathrm{V},\ \mathbf{n}_\mathbf{v}^\mathsf{T}\mathbf{u}_1 = 0,\ \mathbf{n}_\mathbf{v}^\mathsf{T}\mathbf{u}_2 = 0,\ t,s \in \mathbb{R}. \tag{4.12}$$

Relocating the vertex $\mathbf{v}$ along $\mathbf{p}$ replaces the current vertex with a new vertex. Let the function $g_\mathbf{v}$ replace a boundary vertex $\mathbf{v} \in \partial \mathrm{V}$ with a given vertex $\mathbf{v}'$ and calculate a functional, e.g. the trace from eq. (2.11), for all incident tetrahedra:

$$g_\mathbf{v}(\mathbf{v}') = \sum_{\tau \in \mathsf{t}(\mathbf{v})} \mathrm{tr}(\mathbf{L}_{(\tau \setminus \mathbf{v}) \cup \mathbf{v}'}). \tag{4.13}$$

With the use of $g_\mathbf{v}$, the energy field on the tangent plane can be expressed as:

$$g_\mathbf{v}(\mathbf{p}(t,s)), \quad \text{where} \quad t,s \in \mathbb{R}. \tag{4.14}$$

The energy field on the tangent plane typically exhibits local minima that can be found with optimization methods such as gradient descent. Thus, adapting the gradient descent scheme to the directional derivative on $\mathbf{p}$, allows for minimizing on the tangent subspace. As the goal is to obtain a gradient for $\mathbf{v} \in \partial \mathrm{V}$ on the tangent plane $\mathbf{p}$, it suffices to develop the gradient at $t = 0$ and $s = 0$. The gradient follows by the chain rule:

$$\nabla g_\mathbf{v}(\mathbf{p}(0,0)) = \nabla_{\mathbf{p}(0,0)} g_\mathbf{v}(\mathbf{p}(0,0)) \nabla \mathbf{p}(0,0) = \nabla_\mathbf{v} g_\mathbf{v}(\mathbf{v}) \nabla \mathbf{p}(0,0). \tag{4.15}$$

The gradient on the tangent plane $\nabla \mathbf{p}(0,0)$ evaluates to $(\mathbf{u}_1, \mathbf{u}_2)^\mathsf{T}$. Due to $g_\mathbf{v}(\mathbf{v})$ simply replacing $\mathbf{v}$ with itself, further simplification reveals a simple calculation scheme for the gradient:

$$\nabla g_\mathbf{v}(p(0,0)) = \nabla_\mathbf{v}\, \mathrm{tr}(\mathbf{L}_{\mathsf{t}(\mathbf{v})}) \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} t_0 \\ s_0 \end{pmatrix} \in \mathbb{R}^2. \tag{4.16}$$

The gradient on the plane can be transformed to $\mathbb{R}^3$, resulting in the directional derivative:

$$\mathbb{R}^3 \ni \nabla_\mathbf{v}|_\mathbf{p}\ \mathrm{tr}(\mathbf{L}_{\mathsf{t}(\mathbf{v})}) = t_0\,\mathbf{u}_1 + s_0\,\mathbf{u}_2. \tag{4.17}$$

In case of a tangent line, one can just drop $\mathbf{u}_2$. The use of directional derivatives provides several benefits for mesh optimization:

1. Reprojection of vertices after gradient descent is not necessary. Thus, it becomes obsolete to find the closest surface triangle, which can be computationally expensive.

2. Line search on a tangent subspace converges against a local minimum. No special convergence criteria are necessary for the boundary.

3. Projection of relocated vertices to the closest surface triangle can produce inversions or projection of vertices onto opposing faces. Line search avoids these issues by inversion-free intervals for gradient descent along the boundary.

4.3.3. Preserving the Boundary

As many applications do not tolerate the loss of geometric detail, this section describes the use of directional derivatives for full boundary preservation. Let $\mathcal{B}$ represent the input boundary and $\partial\mathrm{T}$ represent the current boundary. Full boundary preservation leads to the following optimization problem:

$$\begin{aligned} &\underset{\mathcal{M}=(\mathrm{T},\mathrm{V})}{\text{minimize}} \ \operatorname{tr}(\mathbf{L}_{\mathrm{T}}) \\ &\text{subject to } \partial\mathrm{T} = \mathcal{B} \wedge \forall\tau \in \mathrm{T}, v_\tau > 0. \end{aligned} \tag{4.18}$$

Relocating a boundary vertex deforms the boundary triangles, which surround the vertex. Thus, it is useful to classify boundary vertices by the surface feature formed by the surrounding boundary triangles. This leads to the following classification into three subsets $\mathrm{V}_{\mathcal{F}} \subseteq \partial\mathrm{V}$, $\mathrm{V}_{\mathcal{R}} \subseteq \partial\mathrm{V}$, and $\mathrm{V}_{\mathcal{C}} \subseteq \partial\mathrm{V}$:

1. $\mathbf{v} \in \mathrm{V}_{\mathcal{F}}$: The vertex $\mathbf{v}$ is a face vertex. All incident boundary triangles are co-planar. Therefore, there is a unique tangent plane.

2. $\mathbf{v} \in \mathrm{V}_{\mathcal{R}}$: The vertex $\mathbf{v}$ is a ridge vertex. Exactly two sets of incident boundary triangles are co-planar. Therefore, there is a unique tangent line.

3. $\mathbf{v} \in \mathrm{V}_{\mathcal{C}}$: The vertex $\mathbf{v}$ is a corner vertex. The incident boundary triangles form more than two sets of co-planar triangles. Therefore, there is no tangent plane or line. A corner vertex cannot be moved without altering the boundary.

The above classification can be performed with the algorithm provided in section 4.2. For full boundary preservation, one can apply homogeneous Neumann boundary conditions [Ale+20], while alternative boundary conditions are an ongoing research topic [Ste+18]:

$$\mathbf{n}_{\mathbf{v}}^{\mathrm{T}} \nabla_{\mathbf{v}} \operatorname{tr}(\mathbf{L}_{\mathrm{t}(\mathbf{v})}) = 0, \quad \text{where} \quad \mathbf{v} \in \partial\mathrm{V}.$$

If a boundary vertex $\mathbf{v} \in \partial\mathrm{V}$ lies on a face of the model, i.e., $\mathbf{v} \in \mathrm{V}_{\mathcal{F}}$, relocating the vertex along the geometrical face does not alter the shape of the model. It is safe to calculate the directional derivative on the tangent plane and perform the vertex relocation algorithm from section 4.3.1 along the face. The inversion-free intervals safeguard the vertex relocation from moving the vertex beyond the face, because it prevents the vertex from crossing a geometric edge, which would impose an inversion. If a boundary vertex lies on a geometric edge of the model, i.e., $\mathbf{v} \in \mathrm{V}_{\mathcal{R}}$, relocating the vertex along the tangent line also does not alter the boundary. Again, inversion-free intervals safeguard the vertex relocation from moving the vertex beyond the geometric edge, because crossing a geometric corner would impose an inversion. If a boundary vertex represents a geometric corner, i.e., $\mathbf{v} \in \mathrm{V}_{\mathcal{C}}$, vertex relocation does not move this vertex to keep the boundary. As a result, vertex relocation can move vertices in $\mathrm{V}_{\mathcal{F}}$ and $\mathrm{V}_{\mathcal{R}}$ along directional derivatives on tangent subspaces without altering the boundary. Algorithm 3 details the calculation of directional derivatives on the tangent subspaces. Algorithm 2 invokes this efficient derivative calculation if full boundary preservation, i.e., $\partial\mathrm{T} = \mathcal{B}$, is the aim. Thereafter, vertex relocation performs the same steps as for interior vertices to reduce branching.

Algorithm 3 Derivative on a tangent sub-space.

```
procedure TANGENTSUBSPACEDERIVATIVE($\mathbf{v}$, $\mathbf{g}$)
    if $\mathbf{v} \in V_{\mathcal{F}}$ then
        $\tau_b \leftarrow$ get_boundary_triangle($\mathbf{v}$)
        $(\mathbf{u}_1, \mathbf{u}_2) \leftarrow$ tangent_plane($\tau_b$)
        return $(\mathbf{u}_1^\intercal \mathbf{g})\mathbf{u}_1 + (\mathbf{u}_2^\intercal \mathbf{g})\mathbf{u}_2$
    else                                                        ▷ Otherwise $\mathbf{v} \in V_{\mathcal{R}}$
        $e_b \leftarrow$ get_boundary_edge($\mathbf{v}$)
        $\mathbf{u}_1 \leftarrow$ tangent_line($e_b$)
        return $(\mathbf{u}_1^\intercal \mathbf{g})\mathbf{u}_1$
    end if
end procedure
```

4.3.4. Approximate Boundary Preservation to Relocate Every Vertex

Since corner vertices cannot be moved with full boundary preservation but can be part of ill-shaped elements, it is also interesting to explore the relocation of every boundary vertex. Thus, this section describes a method to relocate every boundary vertex. In order to approximately preserve the surface, the method keeps boundary vertices on the boundary. As a result, the approximation error is controlled by the input mesh surface, which is assumed to be of high resolution such that the approximation error for curved surfaces is low enough. At the same time, the method benefits from the convergence properties of directional derivatives. Prior to the specification of the method for boundary treatment, the optimization problem is adjusted to approximate boundary preservation:

$$\begin{aligned} &\underset{\mathcal{M}=(\mathrm{T},\mathrm{V})}{\text{minimize}}\ \operatorname{tr}(\mathbf{L}_\mathrm{T}) \\ &\text{subject to } \partial\mathrm{T} \cong \mathcal{B} \wedge \forall\tau \in \mathrm{T}, v_\tau > 0, \end{aligned} \tag{4.19}$$

where $\cong$ represents an approximate congruence between the input boundary $\mathcal{B}$ and the current mesh boundary $\partial\mathrm{T}$.

The main idea of the approximate boundary preservation method is to relax the homogeneous Neumann boundary conditions such that the gradient has to lie only on a single tangent plane (or line) $\mathbf{p}$:

$$\begin{aligned} &\mathbf{n}_p^\intercal \nabla_\mathbf{v}|_p\ \operatorname{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{v})}) = 0,\ \text{where } \mathbf{v} \in \partial\mathrm{V} \\ &\mathbf{v} - \alpha\nabla_\mathbf{v}|_p\ \operatorname{tr}(\mathbf{L}_{\mathtt{t}(\mathbf{v})}) \text{ is on } \mathcal{B},\ \text{where } \alpha \in [0, \lambda_\mathbf{v}]. \end{aligned}$$

This relaxation enables relocation of a vertex along a single boundary primitive granting deviations from the input surface for better mesh quality. Since a vertex has to lie only on a single boundary primitive, there are multiple possible tangent sub-spaces and thereby multiple relocation directions. Therefore, a shape quality optimization method needs to choose one relocation direction in a reasonable way such that convergence is achieved. Algorithm 4 summarizes the procedure to choose a directional derivative along the boundary. The possible relocation directions of a boundary vertex depend on the location of the vertex on the boundary. Figure 4.3 visualizes how a boundary vertex is relocated along the input boundary depending on the boundary primitive it lies on. Initially, every boundary vertex can be relocated along adjacent boundary triangles or edges. After relocation this boundary vertex either lies on a boundary edge or triangle. If the vertex lies on a boundary edge, it is either relocated along the one of the two adjacent boundary triangles or the current boundary edge. If the vertex lies on a boundary triangle it is relocated along this triangle. Since the available choices for next relocation directions depends on the current location of a vertex on the boundary, the optimization procedure keeps track of the current location checking for three boundary states:

1. **On vertex:** The vertex is coincident with a vertex of the input boundary $\mathcal{B}$.

2. **On edge:** The vertex lies on a boundary edge but is not coincident with one of its end points.
3. **On triangle:** The vertex lies on a boundary triangle, while it is neither on an edge of the triangle nor coincident with a vertex of the triangle.

Algorithm 4 Algorithm to find directional derivative for boundary vertex

```
 1: procedure FINDTANGENTDERIVATIVE($\mathbf{v}$, $\lambda$, $\mathbf{g}$)
 2:     $\mathbf{g}_p \leftarrow 0$                                        ▷ $\mathbf{g}_p \in \mathbb{R}^d$
 3:     if $\mathbf{v}$ is on boundary vertex $\mathbf{v}_b$ then
 4:         for boundary triangle $\tau_b$ containing $\mathbf{v}_b$ do
 5:             $(\mathbf{u}_1, \mathbf{u}_2) \leftarrow tangent\_plane(\tau_b)$
 6:             $\mathbf{d}_p \leftarrow (\mathbf{u}_1^\intercal \mathbf{g})\mathbf{u}_1 + (\mathbf{u}_2^\intercal \mathbf{g})\mathbf{u}_2$
 7:             if ray $\mathbf{v} - \beta \mathbf{d}_p$ intersects $\tau_b \setminus \mathbf{v}_b \wedge \|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ then
 8:                 $\mathbf{g}_p \leftarrow \mathbf{d}_p$
 9:                 $\lambda \leftarrow \beta_0$               ▷ For $\beta_0$ point along ray is on $\tau_b \setminus \mathbf{v}_b$
10:                 Set $\mathbf{v}$ on $\tau_b$
11:             end if
12:         end for
13:         for boundary edge $e_b$ containing $\mathbf{v}_b$ do
14:             $\mathbf{u}_1 \leftarrow tangent\_line(e_b)$
15:             $\mathbf{d}_p \leftarrow (\mathbf{u}_1^\intercal \mathbf{g})\mathbf{u}_1$
16:             if ray $\mathbf{v} - \beta \mathbf{d}_p$ intersects $e_b \setminus \mathbf{v}_b \wedge \|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ then
17:                 Analogous to lines 8 and 9
18:                 Set $\mathbf{v}$ on $e_b$
19:             end if
20:         end for
21:     else if $\mathbf{v}$ is on boundary edge $e_b$ then
22:         for boundary triangle $\tau_b$ adjacent to $e_b$ do
23:             Analogous to lines 5 and 6
24:             $(\mathbf{v}_0, \mathbf{v}_1) \leftarrow e_b$
25:             if ray $\mathbf{v} - \beta d_p$ intersects $\tau_b \setminus \mathbf{v}_0$ or $\tau_b \setminus \mathbf{v}_1$ then
26:                 if $\|\mathbf{g}_p\|_2 < \|\mathbf{d}_p\|_2$ then
27:                     Analogous to lines 8 - 10
28:                 end if
29:             end if
30:         end for
31:         Check $e_b$ analogous to lines 14 - 19
32:     else $\mathbf{v}$ is on boundary triangle $\tau_b$
33:         for boundary edge $e_b$ in $\tau_b$ do
34:             Check $e_b$ analogous to lines 14 - 19
35:         end for
36:     end if
37:     return $\mathbf{g}_p$
38: end procedure
```

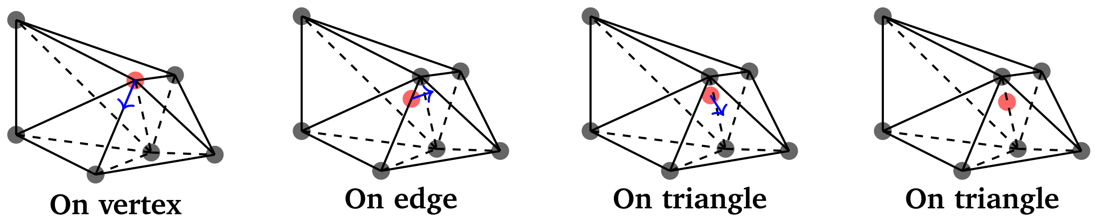


Figure 4.3.: Initially, each vertex coincides with a vertex of the input boundary $\mathcal{B}$. In this example, the directional derivative on the boundary with the largest magnitude is along an edge. After relocating the vertex, the directional derivative of a boundary triangle is chosen. Finally, optimization converges after relocating the vertex on the triangle.

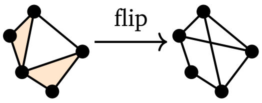
Depending on the state of the to-be-optimized vertex, algorithm 4 accesses adjacent boundary primitives to calculate directional derivatives. As the gradient magnitude is associated with the greatest impact on the to-be-optimized functional, the method follows the rule of steepest descent and chooses the directional derivative with the largest magnitude. After a descent direction is chosen, the state of the vertex is updated. If the chosen descent direction is along a boundary triangle, algorithm 4 updates the state of the vertex to be on this triangle. Likewise, if the chosen descent direction is along a boundary edge, algorithm 4 updates the state of the vertex to be on this edge. The chosen descent direction is returned to the basic gradient descent routine (see line 10 in algorithm 2). The optimization procedure then performs the same steps as for interior vertices with only few adjustments. Thus, optimization of boundary vertices is executed simultaneous to interior vertices and branching is reduced. As it is not allowed to relocate a boundary vertex away from the input boundary $\mathcal{B}$, the inversion-free interval is needs to be pruned so that the boundary vertex still lies on $\mathcal{B}$ after a gradient descent step (see lines 23 - 25 in algorithm 2). Due to the adjustment of the inversion-free interval, boundary vertices can be optimized by bracketing simultaneous to the interior vertices. After a gradient descent step, a boundary vertex still lies on the same boundary primitive but its state may change. Therefore, the vertex relocation procedure needs to identify the new boundary state of a boundary vertex, whenever a gradient descent step is performed (see lines 31 - 33 in algorithm 2). Algorithm 5 summarizes the identification of the boundary state. If a boundary vertex is relocated along a boundary triangle, the relocated vertex may lie on a boundary edge or is coincident with a boundary vertex. This can be checked by calculating the barycentric coordinates of the relocated vertex on its current boundary triangle. Likewise, if a boundary vertex is relocated along a boundary edge, the relocated vertex may coincide with on of the endpoints, which can be determined by calculating barycentric coordinates. After updating the state of a boundary vertex, another gradient descent step can be performed.

Algorithm 5 Algorithm to identify the boundary state

```
procedure IDENTIFYBOUNDARYSTATE($\mathbf{v}$)
    if $\mathbf{v}$ is on a boundary triangle $\tau_b$ then
        $\lambda_0, \lambda_1, \lambda_2 \leftarrow barycentrics(\mathbf{v}, \tau_b)$
        if $\lambda_i \approx 0 \land \lambda_j \approx 0 \land j \neq i$ then
            Set $\mathbf{v}$ on corresponding edge
        else if $\lambda_i \approx 0$ then
            Set $\mathbf{v}$ on corresponding vertex
        end if
    end if
    if $\mathbf{v}$ is on a boundary edge $e_b$ then
        $\lambda_0, \lambda_1 \leftarrow barycentrics(\mathbf{v}, e_b)$
        if $\lambda_i \approx 0$ then
            Set $\mathbf{v}$ on corresponding vertex
        end if
    end if
end procedure
```

4.4. Parallel Harmonic Flipping

After gradient descent steps, the triangulation may not be harmonic anymore, because new admissible harmonic flips (cf. section 2.1.3) can be performed due to relocation of vertex positions. In order to harmonize a triangulation, i.e., ensure that a triangulation is harmonic, the optimization procedure performs harmonic flips. While the harmonic flipping algorithm of Alexa [Ale19] performs flips sequentially in a queue, the algorithm described in this section allows for massively parallel flipping without significantly different triangulations compared to sequential flipping.



Performing flips in parallel requires conflict detection, because otherwise flipping does not guarantee a valid mesh. The inset to the left shows an example on how simultaneous flipping can produce an invalid mesh for triangles in the plane. Suppose one thread is assigned to each of the two orange triangles adjacent to the white triangle. Concurrently, both threads perform a flip, which leads to an invalid triangulation.

Harmonic flips are ordered by their reduction of the trace $\mathrm{tr}(\mathbf{L}_\mathrm{T})$. If two flips are in conflict with each other, the flip with a larger reduction of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$ is preferred over the flip with a smaller reduction of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$. This property is useful for parallel flipping, because it provides a rule to decide for a flip, when several flips are in conflict. Thus, the massively parallel flipping algorithm performs harmonic flips, whose reduction of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$ is greater than every harmonic flip in the local vicinity. Henceforth, the flip with the largest reduction of $\mathrm{tr}(\mathbf{L}_\mathrm{T})$ in the local vicinity is referred to as locally most beneficial. Figure 4.4 provides a visualization on the preference of locally most beneficial flips to prevent conflicts.

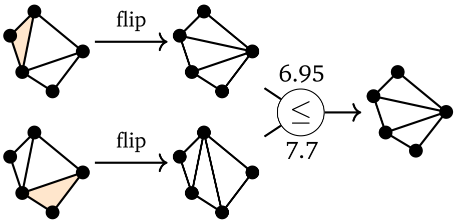


Figure 4.4.: The proposed algorithm chooses the locally most beneficial flip to obtain a valid triangulation.

Algorithm 6 outlines a massively-parallel method to identify and perform locally most beneficial harmonic flips. The algorithm encodes flips by the index `i` of the flipped mesh facet and an identifier `id` for the type of the flip:

$$\mathit{flip} = (\mathtt{i}, \mathtt{id}) \in \mathbb{Z} \times \{\text{2-3-flip}, \text{3-2-flip}, \emptyset\} \tag{4.20}$$

Initially, the algorithm performs a parallel pass over all $\tau \in \mathrm{T}$ to identify the most beneficial flip for each τ. Each τ can be flipped at either one of its six edges or one of its four faces. Hence, the method evaluates feasibility checks and quality improvement regarding the harmonic index η of the potential flips in a predetermined order. Face or edge flips are only feasible on interior faces or edges, respectively. Each flip requires its incident tetrahedra to form a convex sub-triangulation. Whenever a flip is feasible, algorithm 6 compares its quality improvement to the currently most beneficial flip (see lines 5 - 14). As a result, the algorithm determines the most beneficial harmonic flip for τ and record it in a previously allocated buffer. If there is no harmonic flip for every $\tau \in \mathrm{T}$, the mesh is already harmonic and the flipping method terminates. Otherwise, algorithm 6 proceeds with buffer allocations and another parallel pass over all $\tau \in \mathrm{T}$.

In order to prepare for building a new triangulation T', the flipping method allocates an array of markers indicating whether $\tau \in \mathrm{T}$ is part of T' or not and an array of integers for the number of newly added tetrahedra. After buffer allocation, the method finds locally most beneficial flips in parallel to avoid conflicts. In a parallel pass over tetrahedra, algorithm 6 accesses the recorded harmonic flip – if any – for every $\tau \in \mathrm{T}$. Using flip type and facet index, it is possible to obtain the the tetrahedra incident to the flip by precomputing connectivity relationships with the TCSR mesh data structure (cf. section 2.2.3). Before the flip can be performed, it is mandatory to check for conflicting flips to ensure a valid mesh. No conflict occurs, if the flip is the most beneficial flip for each incident tetrahedron. In this case, the flip is locally the most beneficial and is selected to be performed. Consequently, the tetrahedron associated with the thread can be marked for removal. As many threads are mapped to a locally most beneficial flip, the flipping method elects a coordinator thread to perform the flip. If the index of τ is the lowest of the incident tetrahedra, the associated thread is declared as the coordinator (see line 27 in algorithm 6). The coordinator thread records the number of tetrahedra added by performing the flip in the previously allocated buffer (see line 31 in algorithm 6). Since only coordinator threads write to the buffer for counting newly added tetrahedra, thread i is a coordinator thread if this buffer holds a non-zero entry at position i.

An exclusive prefix sum over the integers counting newly added tetrahedra provides offset positions and the total number of tetrahedra to be added. The marker values of the tetrahedra $\tau \in \mathrm{T}$ in sum amount to the

Algorithm 6 Parallel flipping algorithm to optimize $\mathcal{M}$

```
procedure FLIPMESH($\mathcal{M} = (\mathrm{T}, \mathrm{V})$)
    $\texttt{flips} \leftarrow ((\text{-}1, \emptyset), \ldots, (\text{-}1, \emptyset)) \in (\mathbb{Z} \times \{\text{2-3-flip}, \text{3-2-flip}, \emptyset\})^{|\mathrm{T}|}$
    for $i = 0, \ldots, |\mathrm{T}| - 1$ do                                    ▷ Find flips in parallel
        $\eta_{\text{impr}} \leftarrow 0$; $\text{flip} \leftarrow (\text{-}1, \text{-}1)$; $\tau \leftarrow \mathrm{T}_i$
        for all faces $f$ in $\tau$ do                                       ▷ In predetermined order
            if $\textit{2-3-flip}(f, \tau)$ is feasible $\wedge\ \eta_{\text{impr}} < \eta_{\text{flip}}$ then
                $\texttt{flips}_i \leftarrow (\textit{index\_of}(f),\ \text{2-3-flip})$
            end if
        end for
        for all edges $e$ in $\tau$ do                                       ▷ In predetermined order
            if $\textit{3-2-flip}(e, \tau)$ is feasible $\wedge\ \eta_{\text{impr}} < \eta_{\text{flip}}$ then
                $\texttt{flips}_i \leftarrow (\textit{index\_of}(e),\ \text{3-2-flip})$
            end if
        end for
    end for
    if $(\text{-}1, \emptyset) = \texttt{flips}_i$ for $i = 0, \ldots, |\mathrm{T}| - 1$ then
        return $\textit{false}$
    end if
    $\texttt{new\_tets} \leftarrow (0, \ldots, 0) \in \mathbb{Z}^{|\mathrm{T}|}$
    $\texttt{tets\_marked} \leftarrow (1, \ldots, 1) \in \{0, 1\}^{|\mathrm{T}|}$
    for $i = 0, \ldots, |\mathrm{T}| - 1$ do                                    ▷ Detect conflicts in parallel
        $(j, \textit{type}) \leftarrow \texttt{flips}_i$; is_coordinator $\leftarrow$ $\textit{true}$; agree $\leftarrow$ $\textit{true}$
        if $j = \text{-}1$ then continue end if
        for all tetrahedron $\tau$ involved in $\textit{flip}(j, \textit{type})$ do
            $k \leftarrow \textit{index\_of}(\tau)$; $(j_{\text{adj}}, \textit{type}_{\text{adj}}) \leftarrow \texttt{flips}_k$
            agree $\leftarrow$ agree $\wedge\ \textit{type} = \textit{type}_{\text{adj}} \wedge j = j_{\text{adj}}$
            is_coordinator $\leftarrow$ is_coordinator $\wedge\ i \leq k$
        end for
        if agree then $\texttt{tets\_marked}_i \leftarrow 0$ end if
        if agree $\wedge$ is_coordinator then
            $\texttt{new\_tets}_i \leftarrow$ if $\textit{type} = \text{2-3-flip}$ then $3$ else $2$ end if
        end if
    end for
    $\texttt{offsets} \leftarrow (0, \ldots, 0) \in \mathbb{Z}^{|\mathrm{T}|+1}$
    for $i = 1, \ldots, |\mathrm{T}|$: $\texttt{offsets}_i \leftarrow \texttt{offsets}_{i-1} + \texttt{new\_tets}_{i-1}$
    $N_{\text{remaining}} \leftarrow \sum_{i=0}^{|\mathrm{T}|-1} \texttt{tets\_marked}_i$
    $\mathrm{T}' \leftarrow \textit{allocate}(N_{\text{remaining}} + \texttt{offsets}_{|\mathrm{T}|})$
    $\textit{copy\_if\_marked}(\text{src} = \mathrm{T}, \text{dst} = \mathrm{T}', \texttt{tets\_marked})$
    for $i = 0, \ldots, |\mathrm{T}| - 1$ do                                    ▷ Perform flips in parallel
        if $\texttt{new\_tets}_i \neq 0$ then
            $(j, \textit{type}) \leftarrow \texttt{flips}_i$; offset $\leftarrow N_{\text{remaining}} + \texttt{offsets}_i$
            $\mathrm{T}'_{\text{offset}} \leftarrow \textit{flip}(j, \textit{type})$
        end if
    end for
    $\mathcal{M} \leftarrow (\mathrm{T}', \mathrm{V})$
    return $\textit{true}$
end procedure
```

number of remaining tetrahedra. Algorithm 6 allocates a new buffer for the resulting tetrahedra T′ and copies the remaining tetrahedra through a stream compaction to T′. In a final parallel pass over the tetrahedra $\tau \in$ T, the coordinator threads perform the flips and append the resulting tetrahedra to the remaining tetrahedra using the offset positions.

4.5. Combined Vertex Relocation and Flipping

For fast convergence of energy reduction, the harmonic mesh optimization algorithm should combine harmonic flips with vertex relocation. Thus, the algorithm performs several alternating passes of vertex relocation and harmonic flipping. It terminates if its effect on the mesh becomes insignificant. Gradient descent converges if the gradient approaches zero. Therefore, the optimization terminates if $\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}})$ is sufficiently small. Thus, when $\|\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}})\|$ is smaller than some ϵ_c, gradient descent is not expected to cause significant improvements. In addition, update rates can become negligible. To avoid this situation, the algorithm terminates if the difference of the current to the prior gradient is smaller than ϵ_c. As $\operatorname{tr}(\mathbf{L}_{\mathrm{T}})$ is scale dependent, it is reasonable to choose a relative ϵ_c. The implementation used in this thesis chooses ϵ_c based on a constant ϵ governing the accuracy in finding a minimum:

$$\epsilon_c \leftarrow \max(\epsilon,\ \epsilon\|\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}})\|) \tag{4.21}$$

As some vertices converge more quickly than others, the algorithm does not further optimize a vertex with a gradient norm smaller than ε_g. This thesis consistently uses $\epsilon = 10^{-5} = \varepsilon_g$ and $\|\cdot\|_2^2$ as the norm.

Connectivity relationships and coloring need to be updated after flipping. Checking for flips is an unnecessary overhead if flips are unlikely to be found. Therefore, a heuristic reduces the number of checks. A counter k_f holds the number of iterations without flip checking and is initialized as $k_f = 1$. Whenever flip checking fails to find flips, the heuristic doubles k_f. Analogously, if flip checking finds flips, k_f is halved rounding up. If the counter has reached a predetermined number 2^N, the algorithm terminates, as additional harmonic flips are unlikely to be found. This thesis uses $N = 3$. In summary, the optimization algorithm terminates at iteration i, if one of the following conditions is met:

(C1) $\|(\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}}))_i\| < \epsilon_c$,

(C2) $\|(\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}}))_i - (\nabla \operatorname{tr}(\mathbf{L}_{\mathrm{T}}))_{i-1}\| < \epsilon_c$, or

(C3) $k_f = 2^N$.

4.6. Evaluation of Harmonic Mesh Optimization

This thesis presents several experiments to evaluate the benefits and limitations of the proposed harmonic mesh optimization methods. All of the experiments were run on an evaluation machine equipped with an Intel i9-11900K CPU and an NVIDIA RTX 3090 GPU. The implementations were compiled using Visual Studio 2022 and CUDA. Before this section presents the evaluation experiments and results, it describes the implementation of the original harmonic mesh optimization algorithm by Alexa [Ale19] in section 4.6.1. The first evaluation experiment in section 4.6.2 investigates the run time improvement due to massively parallel flipping. Subsequently, section 4.6.3 critically evaluates the robustness of the proposed optimization method. Section 4.6.4 compares the convergence and resulting element qualities of the two algorithms. Finally, section 4.6.5 investigates the run time benefits due to massively parallel processing.

4.6.1. Implementation of the Original Harmonic Mesh Optimization Algorithm

The implementation of the original algorithm to harmonize a triangulation depends on the computational geometry algorithms library (CGAL) [FT15] for robustness. Harmonic flipping first iterates over the tetrahedra of the mesh checking for admissible flips. Whenever an admissible flip reduces $\mathrm{tr}(\mathbf{L}_\mathrm{T})$, the flipping algorithm pushes the flip to a priority queue that orders flips by their reduction of the trace. Subsequently, the flipping algorithm performs the flips in queued order. To obtain a valid triangulation a safety check ensures that a flip has not been invalidated by a prior flip in local neighborhood. Invalidated flips are skipped by the flipping algorithm. For vertex position optimization, the harmonization determines a global gradient for the entire triangulation.

4.6.2. Run Time Performance of Parallel Flipping

The evaluation compares the proposed GPU parallel harmonic flipping algorithm performing locally most beneficial flips to the sequential CPU algorithm performing flips in an ordered queue. Both algorithms perform flips on the input mesh, until no further harmonic flips can be found. The results appear in table 4.1. While Alexa [Ale19] performed harmonic flips on Delaunay triangulations (cf. section 2.1.2) of point sets, the evaluation performs flips on meshes generated with TetGen [Si20] a constrained Delaunay mesher, leading to a lesser reduction of the number of tetrahedra. The evaluation details the exact numbers of tetrahedra in the resulting triangulations, in order to show that postponing locally not most beneficial flips to later flipping passes does not lead to significant differences in the resulting triangulation.

Table 4.1.: Comparison of the parallel harmonic flipping algorithm to sequential harmonic flipping. Faster run times are provided in boldface.

Mesh		**Sequential flipping**		**Proposed (GPU)**		
Name	N_T	Time (sec)	N_T	Time (sec)	N_T	Speedup
Ghost	160799	0.398	154263	**0.003**	154263	133×
Die	232767	0.665	225689	**0.003**	225689	222×
Snowman	227567	0.638	217837	**0.003**	217837	213×
Barrel	463797	2.544	443838	**0.010**	443838	254×
Falcon	1072784	5.074	1034101	**0.023**	1034103	221×
Part	1099271	5.349	1057930	**0.023**	1057930	233×
Cube	1538635	9.859	1471163	**0.039**	1471162	253×
World	1786620	9.173	1725729	**0.053**	1725730	173×
Pot	4034608	25.130	3886310	**0.121**	3886309	208×

The experiments reveal substantial speedups of 133×–254×. As harmonic bistellar flips either coincide with the Delaunay triangulation or reduce a triangulation of three tetrahedra to two tetrahedra, the parallel flipping algorithm is a useful tool for mesh optimization and generation, quickly reducing the tetrahedron count while in some sense preserving the Delaunay criterion. The experiments confirm that harmonic flipping well preserves the percentage of locally Delaunay tetrahedra.

4.6.3. Robustness

In order to validate the practicability of the parallel optimization algorithms, the evaluation applied the proposed algorithm in section 4.5 to the 10 k tetrahedral meshes generated by Hu et al. [Hu+20]. An automated evaluation procedure investigates the optimized meshes. The algorithm did not produce any inversion due to the choice of the inversion-free interval for each vertex **v**. After termination, each triangular face was connected to one or two tetrahedra. In addition, the resulting meshes consistently exhibited alternating face orientations for triangular faces adjacent to two tetrahedra. The Manhattan distance of distinct vertices was larger than 10^{-10} for all except for two meshes meaning that the proposed optimization method does not produce geometrically duplicated vertices. For the two meshes with geometrically close vertices, closer investigation revealed smaller vertex distances already before optimization.

The evaluation includes the one-sided Hausdorff distance of the boundary vertices of the optimized mesh to the input mesh surface, in order to validate that the vertex relocation algorithm on the boundary (cf. section 4.3.4) keeps vertices on the boundary. Furthermore, the automated evaluation procedure divides the Hausdorff distances by the average boundary edge length to put them in relation to the dimensions of the model. For 99.95 % of the meshes, the one-sided Hausdorff distance was below 10^{-3}, which shows that boundary vertices remain on the input surface considering round-off errors. In four out of the five remaining cases, round-off errors on directional derivative calculation accumulate to a degree that the resulting deviation is roughly 10^{-2}. In only one case, a significant deviation of 0.19 can be observed. As the meshes generated by Hu et al. [Hu+20] generally are of high quality and already optimized, the experiments regarding run time, convergence and mesh quality use the unoptimized meshes in fig. 4.5.

4.6.4. Element Quality and Convergence

The evaluation includes investigation of element quality and convergence of both Alexa's method [Ale19] and the algorithm in section 4.5. In order to obtain deterministic results, this investigation uses the graph coloring of the Cusparse library [NVI22c]. This thesis covers two methods of using directional derivatives at the boundary. Using directional derivatives only for vertices in $V_{\mathcal{F}}$ and $V_{\mathcal{R}}$ provides surface preservation ($\partial T = \mathcal{B}$). In addition, directional derivatives can be used to move vertices along the input surface ($\partial T \cong \mathcal{B}$) to optimize all vertices at the cost of altering the model shape. The evaluation compares Alexa's reprojection-based method [Ale19] to both variants of boundary treatment.

Table 4.2.: Dihedral angles φ and energy states using Alexa's [Ale19] and the proposed method ($\partial T = \mathcal{B}$) on meshes with few corner vertices. Better results are provided in boldface.

	Input					**Alexa's [Ale19]**					**Proposed method $\partial T = \mathcal{B}$**				
Name	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$
Snowman	0.1	27.46	1849.08	11.4	1.33e+06	3.7	33.4	44.68	7.69	1.14e+06	**8.67**	**36.32**	**18.55**	**6.72**	**1.05e+06**
Barrel	0.12	27.71	530.69	2.92	828788	9.26	36.62	4.94	2.03	671173	**10.84**	**36.66**	**4.46**	**2**	**659666**
Part	0.26	30.38	322.04	6.76	4.53e+06	**7.3**	**38.57**	11.8	4.95	3.88e+06	7.07	38.54	**11**	**4.9**	**3.83e+06**
Cube	0.08	29.12	134.4	0.52	495820	6.8	37.48	1.29	0.37	410171	**10.72**	**38.48**	**0.8**	**0.35**	**400212**

The boundary preserving method is most useful for input meshes with few corner vertices. Thus, the boundary preserving method is a good choice to optimize the top four meshes shown in fig. 4.5. Table 4.2 provides the resulting element qualities. Although all of the input meshes include critical minimal dihedral angles, both methods achieve to improve the minimal angle, while the proposed method achieves significantly larger minimal angles with the exception of similar minimal angles for the Part. Likewise, the lower 5 % of dihedral angles is significantly larger with the exception of the Part. For relocating every boundary vertex of the Part model, the proposed algorithm achieves a minimal dihedral angle of $8.12°$ and a lower 5-percentile $\varphi_{5\%}$ of $38.63°$, which is a better result than the reprojection-based method. Relocating every boundary vertex does not result in significant differences for the other three meshes. The method in section 4.5 consistently results in lower energy states for η regarding the maximum, 95-percentile and the sum over T.

For meshes with many corner vertices approximating curved surfaces, optimizing all boundary vertices

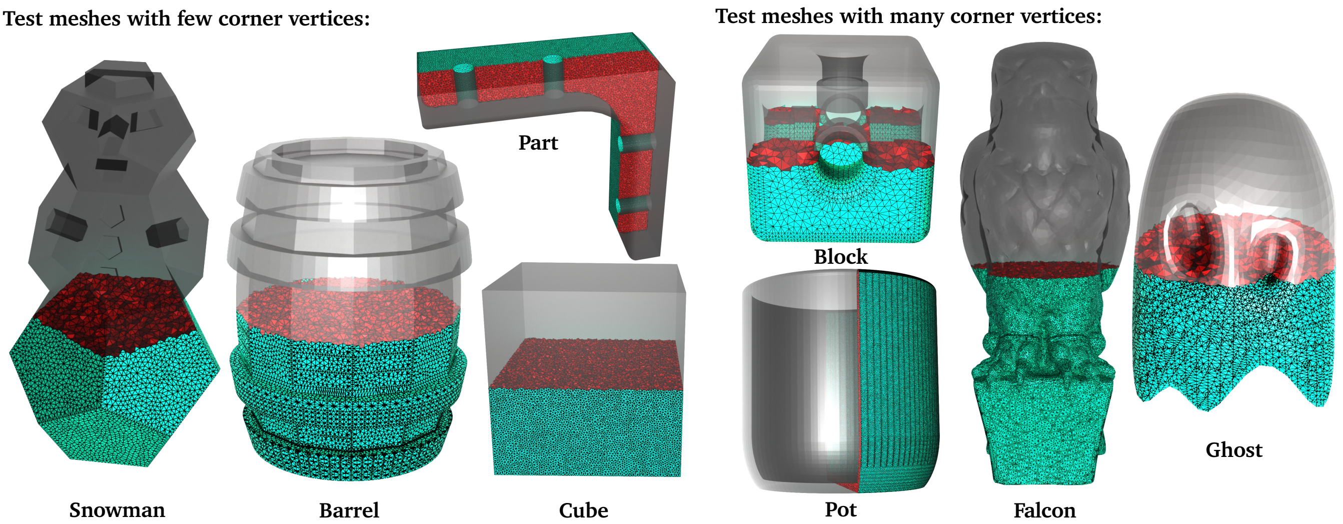

Test meshes with few corner vertices:
Test meshes with many corner vertices:
Part
Block
Snowman
Barrel
Cube
Pot
Falcon
Ghost

Figure 4.5.: Test meshes used in the evaluation.

Table 4.3.: Dihedral angles φ and energy states using Alexa [Ale19] and the proposed method ($\partial T \cong \mathcal{B}$) on meshes with many corner vertices. Better results are provided in boldface.

	Input					**Alexa's [Ale19]**					**Proposed method $\partial T \cong \mathcal{B}$**				
Name	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$	$\varphi_{\min}$	$\varphi_{5\%}$	$\eta_{\max}$	$\eta_{95\%}$	$\eta(T)$
Block	4.65	27.22	82.37	20.31	960808	3.98	**34.32**	47.5	**15.21**	796930	**9.84**	34.2	**32.77**	15.42	**770264**
Ghost	0.03	25.36	9622.21	10.42	982795	1.72	29.93	83.42	8.36	840025	**2.79**	**33.5**	**22.34**	**6.93**	**721325**
Falcon	0.65	26.82	217.02	18.62	1.15e+07	2.53	32.87	76.35	14.45	1.01e+07	**3.84**	**34.00**	**32.77**	**13.08**	**9.27e+06**
Pot	4.14	28.79	40.06	10.12	2.51e+07	0.04	33.67	2252.44	8.36	2.29e+07	**9.21**	**37.01**	**17.05**	**7.31**	**2.09e+07**

is important, as boundary preserving optimization typically results in smaller minimal angles oftentimes not even half as large. For this reason, the evaluation investigates resulting element qualities optimizing corner vertices on the four meshes on the right side of fig. 4.5 and provides the results in table 4.3. While the method from section 4.3.4 improves the minimal angles of all inputs, reprojection-based optimization impairs the initial minimal angles on the Block and Pot meshes. As the reprojection step does not respect energy minimization, a degradation of mesh quality may occur. Using directional derivatives along the boundary respects energy minimization leading to lower energy states for η with the exception of the 95-percentile of the Block mesh.

As it is important to preserve the shape of the model, the evaluation includes investigation of the impact on the mesh surfaces and the convergence of the optimization methods on many different meshes. Figure 4.6 presents typical results. While reprojection of boundary vertices distorts sharp detail, directional derivatives along boundary faces and edges can be used to preserve the mesh surface. Moreover, the reprojection step mitigates convergence, because it does not respect energy minimization. On the contrary, gradient descent of directional derivatives converges to a local minimum on the boundary, as can be seen in the monotonously decreasing curve of the gradient norm for the Barrel. The mesh harmonic optimization algorithm described in section 4.5 exhibits convergence for relocating every vertex along the boundary. The reprojection-based optimization oftentimes terminates in a premature state. Since Alexa's [Ale19] algorithm potentially chooses small step sizes, reprojection to the closest point on the mesh surface oftentimes does not significantly change vertex positions from the initial state leaving a lot of optimization potential. Directional derivatives along the boundary respect energy minimization even when migrating to different boundary primitives. However, the gradient norm does not reduce as monotonous as for exact boundary preservation, as gradient norms change, when a vertex is associated with another boundary primitive. Convergence is achieved though, while the input shape is approximately preserved. This is notable, as the use of directional derivatives enables robust improvement of high-resolution meshes and keeps boundary vertices on the boundary while converging.

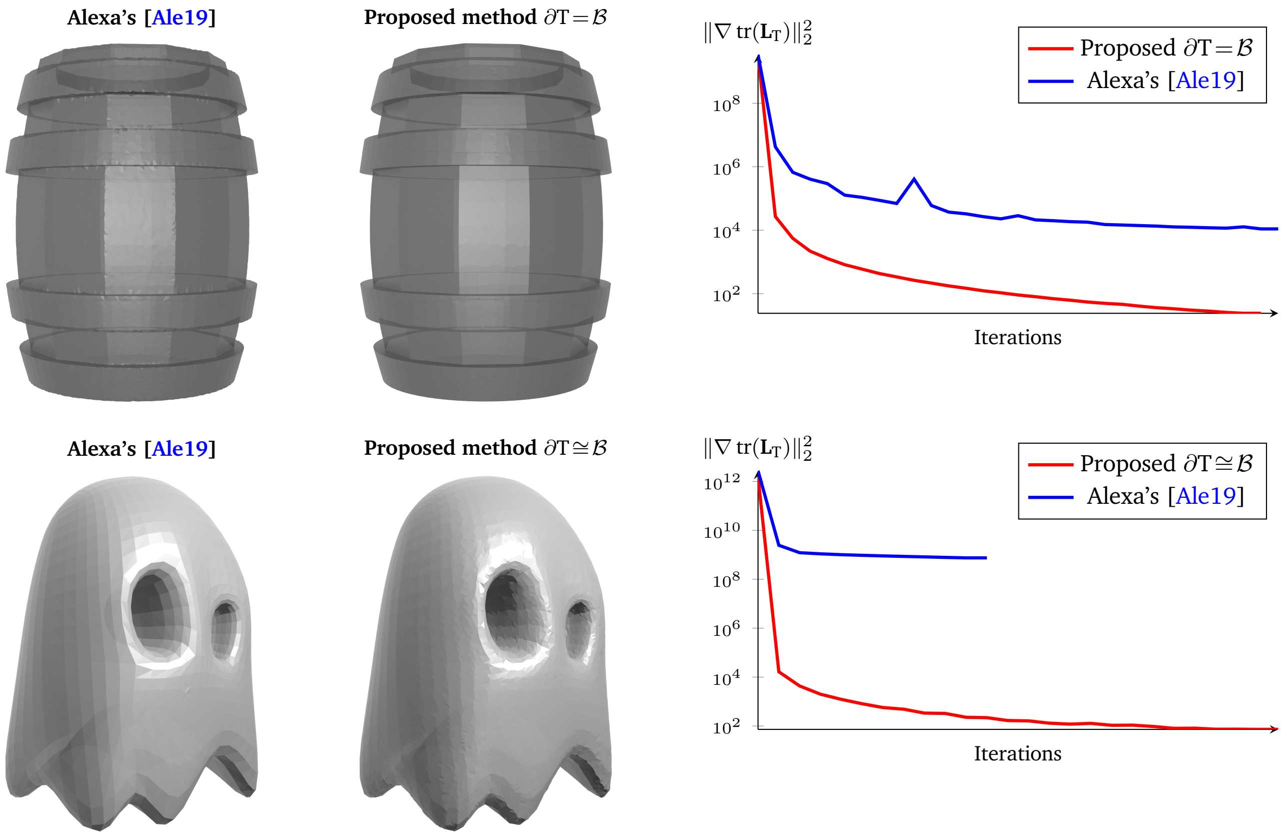


Figure 4.6.: Comparison of the proposed method (red) to Alexa's [Ale19] reprojection based method (blue). To show the effects of the gradient descent at the boundary, the evaluation visualizes resulting boundaries and plots the gradient norm throughout optimization.

4.6.5. Run Time Performance of Harmonic Mesh Optimization

As one of the goals of this thesis is to provide quick processing of unstructured tetrahedral meshes (see RQ1), the evaluation compares run times of Alexa's [Ale19] and the proposed massively parallel algorithm for full and approximate boundary preservation. Since the run time performance of vertex relocation (cf. section 4.3.1) depends on the coloring strategy of use, the evaluation covers experiments with the deterministic graph coloring method in Cusparse and the indeterministic graph coloring of [Dev+16]. In order to prevent outliers [HB15] and to obtain representative measurements, the run time evaluation determines the median run time of 20 executions. Figure 4.7 shows the run time comparisons for full and approximate boundary preservation.

For full boundary preservation, the proposed harmonic mesh optimization algorithm achieves notable speedups of $10.55\times$ - $84\times$. For the smaller meshes, the graph coloring of Deveci et al. [Dev+16] leads to up to $2\times$ faster median run time performance compared to using the deterministic graph coloring of the Cusparse library [NVI22c]. An important factor for the run time performance is the number of different colors that are used for the graph coloring. The fewer colors are used the less parallel passes are performed to relocate all the vertices of the mesh. As shown by Giebel [Gie22] in a student lab, the graph coloring of Deveci et al. [Dev+16] typically requires less colors than the graph coloring of the Cusparse library [NVI22c]. Thus, the run time performance of vertex relocation benefits from more concurrency. However, the use of fewer colors can lead to larger iteration counts, because the optimization of vertex positions does not benefit as much from prior relocation of adjacent vertices performed for previous colors in the execution order. Consequently, the run time performance can slightly degrade for larger meshes when using the graph coloring of Deveci et al. [Dev+16]. Although the boundary reprojection prevents convergence, the original algorithm still performs a considerable number of iterations until no harmonic flips can be found.

This evaluation does not show larger iteration counts for optimizing most of the more complex meshes with vertex relocation along the boundary (cf. section 4.3.4). The reduced convergence of reprojecting vertices on the boundary leads to lower iteration numbers of the original optimization algorithm. Additionally, the method for optimizing vertices along the boundary imposes more branching than the full boundary preservation reducing the impact of massively parallel processing (cf. section 2.2.1) and leading to up to 40 % slower run times. Thus, the boundary approximating harmonic mesh optimization algorithm obtains lower but still significant speedups of 3.77× - 60×. The evaluation of the approximate boundary preservation exhibits run time behaviors for the two coloring variants from Cusparse [NVI22c] and Deveci et al. [Dev+16] that are similar to the observations for exact boundary preservation.

Besides the use of the GPU, there are additional reasons for the substantial speedups. One important reason is the use of proper gradient descent at the boundary. This frequently leads to lower iteration counts to achieve convergence, which typically leads to better run time performance. The convergence of the proposed optimization method is further improved by the use of a Gauß-Seidel iteration order, because gradient descent

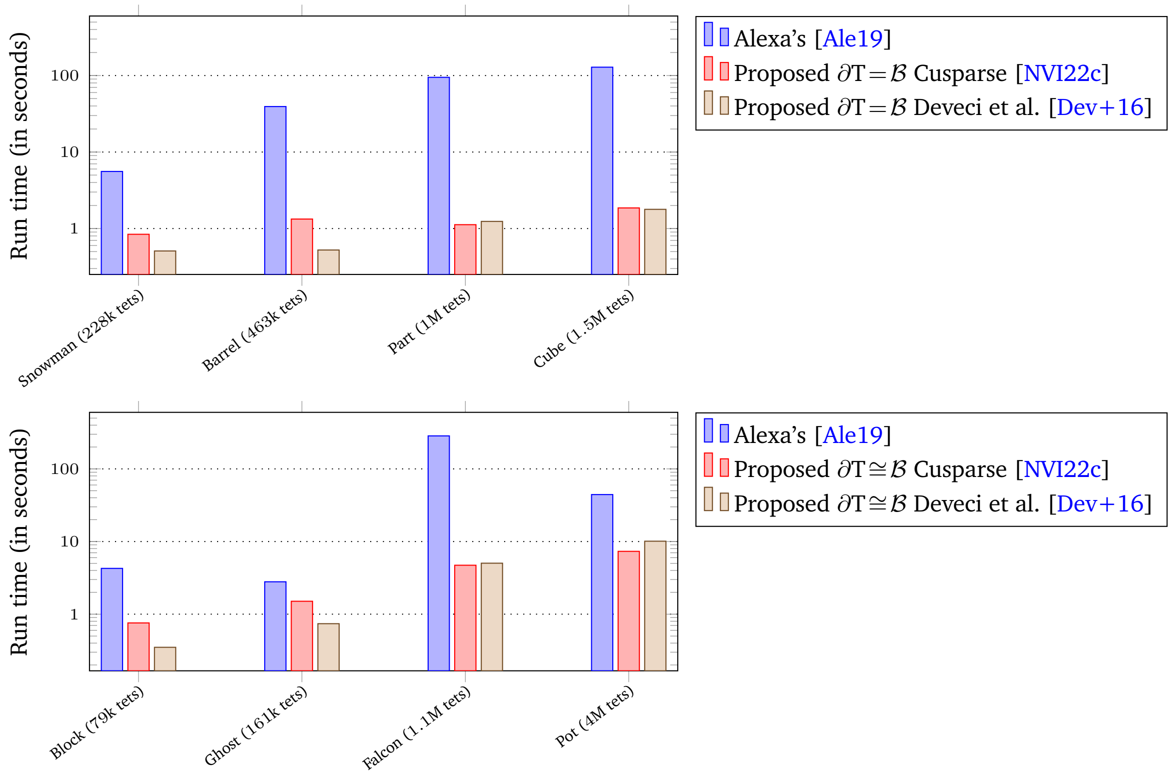


Figure 4.7.: Run times of Alexa's algorithm [Ale19] compared to the proposed algorithm for full boundary preservation (top) and approximate boundary preservation (bottom).

does no longer suffer from the issue that opposing gradients of adjacent vertices are conflicting. Another reason for the improved performance is that the proposed method does not require a reprojection step. The reprojection step needs to perform a spatial search for the closest boundary triangle that should be accelerated by spatial data structure. The proposed scheme performs gradient descent on the boundary and does need to construct a spatial data structure for the spatial search.

4.7. Summary

In summary, this chapter has presented a robust, massively parallel method for optimizing the quality of unstructured tetrahedral meshes. This optimization method couples vertex relocation with re-meshing using edge/face flips. The algorithm for massively parallel line searches in the interior of $\mathcal{M}$ as well as on the boundary $\mathcal{B}$ proved successful in achieving good run time performance and preventing element inversion. This chapter has presented a parallelization strategy for edge/face flips that has achieved a speedup of two orders of magnitude in the evaluation. Applying the proposed algorithm to $10\,\mathrm{k}$ of unstructured tetrahedral meshes has showed that it produces valid meshes. The evaluation has revealed substantial speedups of up to $84\times$ for full boundary preservation and of up to $60\times$ for approximate boundary preservation.

Besides proper parallelization and robustness for mesh optimization, this chapter has presented additional contributions. This chapter extended the harmonic triangulation by presenting how to calculate the harmonic gradient for a specific vertex of the tetrahedron τ using only face areas, face normals and the Jacobian of τ. Thus, one can easily use the harmonic gradient for optimization in a Gauß-Seidel iteration manner. Furthermore, this chapter has presented a method for performing gradient-descent iterations along a closed triangle mesh with good convergence properties.

In view of RQ1[1], RQ2[2] and RQ3[3], this chapter has allowed for answering these questions for the optimization of meshes. For answering RQ1, this chapter has led to two important findings:

- The parallelization of robust vertex relocation at both the interior and the boundary of the mesh achieves significant acceleration using the proposed algorithms.
- The parallelization of edge/face flips leads to substantial improvements of run time performance using the proposed parallelization strategy.

For answering RQ2 this chapter has shown that the proposed massively parallel optimization algorithms produce valid meshes on a large test set. Since mesh optimization and edge/face flips are an important part in mesh generation for VP purposes, the findings in this chapter confirm for RQ3 that the meshing step can benefit from the proposed algorithms. As edge/face flips are only a few among many re-meshing operations, the subsequent chapter will investigate massively-parallel re-meshing in more detail.

[1] RQ1: *How can the use of the GPU accelerate mesh optimization and re-meshing tasks for unstructured tetrahedral meshes?*

[2] RQ2: *How can massively parallel optimization and re-meshing of unstructured tetrahedral meshes robustly produce meshes of sufficient quality for numerical simulations?*

[3] RQ3: *How can massively parallel mesh processing be used for quick editing of unstructured tetrahedral meshes to accelerate VP cycles?*

5. Massively Parallel Collapsing of Edges of Unstructured Tetrahedral Meshes

The following paper contains the core content of this chapter:

[SSF23] **D. Ströter**, A. Stork, D. W. Fellner, “Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes”. In: *High-Performance Graphics - Symposium Papers*. Ed. by Jacco Bikker and Christiaan Gribble. Presented at High-Performance Graphics 2023. The Eurographics Association, 2023. DOI: 10.2312/hpg.20231139

In the preceding section, the use of massively parallel edge/face flips was enabled by proper conflict detection. Since the parallelization of edge/face flips alone does not address the comprehensive set of available re-meshing operators (cf. section 3.2.6), this chapter further investigates RQ1 and RQ2. In order to enable massively parallel re-meshing for more operators, this chapter attempts to devise a conflict detection for more general re-meshing of unstructured tetrahedral meshes. As it is one of the most challenging re-meshing operators to massively parallelize (see section 3.2.6), this chapter focuses on edge collapsing in unstructured tetrahedral meshes. A conflict detection for collapsing edges in unstructured tetrahedral meshes, should provide the following properties:

- enclose each edge with a large enough sub-mesh so that re-meshing produces a valid inversion-free mesh and can optimize, e.g., for mesh quality,
- at the same time, determine a dense set of conflict-free sub-meshes to enable good exploitation of the massively parallel processing power of a GPU,
- prioritize collapsing operations with the most desirable impact on the mesh, e.g., lowest loss of geometric detail or best element quality improvement.

A conflict detection that satisfies the above properties provides good means for massively parallel re-meshing of unstructured tetrahedral meshes, because many meshing operators can be parallelized with a conflict detection for collapsing edges [LM14]. Thus, attempting to establish such conflict detection would allow for profound means to discuss RQ1 and RQ2.

In the context of VP processes, high-resolution tetrahedral meshes can lead to substantial workloads for numerical simulation or post-processing applications. High-resolution meshes impose not only slow run time performance but also substantial memory occupation. Unfortunately, some meshing tools can be difficult to control for a target element size and it is generally difficult to predict how many elements are required to achieve sufficient accuracy. In addition, it can be interesting to re-purpose an already existing unstructured tetrahedral mesh to other applications that need a less fine-grained resolution for sufficiently accurate numerical simulation. For instance, it can be beneficial to re-purpose a mesh for a linear elasticity simulation for a fast heat simulation, while keeping the high-resolution geometric detail of the initial mesh. Therefore, mesh coarsening is interesting for improving run time performance and memory demands, which is related to RQ4. However, coarsening of unstructured tetrahedral meshes can be a compute intensive task. Especially for high-resolution meshes, the coarsening process can last several minutes or even hours [CDM04], because the removal of elements is concerned with several optimization constraints such as element quality and preservation of the boundary. Moreover, as the collapsing of edges is effective in eliminating low-quality elements [Lo14a], fast mesh coarsening can quickly improve meshes in time-critical applications such as fluid dynamics [Ala+06]. For better run time performance, the parallelization of mesh coarsening seems promising. Due to the sequential nature of optimization-driven re-meshing, the parallelization of mesh coarsening remains a challenging problem, although the academic community has devised many parallelization strategies for re-meshing operations (see section 3.2.6).

A massively parallel algorithm for collapsing edges appears in section 5.1. As mesh quality is important to numerical methods, section 5.2 presents a method to collapse edges to improve the quality of an unstructured tetrahedral mesh. Section 5.3 presents a tool that combines massively parallel re-meshing operators for mesh adaption. The evaluation in section 5.4 investigates the run time performance and robustness of the collapsing algorithm. Finally, section 5.5 summarizes the key conclusions of this chapter.

5.1. Collapsing Algorithm

This section presents an algorithm for massively parallel collapsing of edges. As this algorithm's mode of operation is supposed to be configurable, in order to support the many use cases of edge collapsing (see section 3.2.4), section 5.1.1 presents the design choices taken for the algorithm. Subsequently, this section successively details the individual steps of the algorithm, which is organized in three subsections.

Since collapsing edges requires to satisfy certain validity conditions (see, e.g., section 3.2.3), section 5.1.2 describes the process of ensuring the algorithm to perform only admissible edge collapse operations. Out of the admissible edge collapse operations, the algorithm performs the steps in section 5.1.3 to determine a dense set of non-overlapping sub-meshes for massively parallel re-meshing prioritized by a cost function. Finally, each sub-mesh includes an interior edge and the algorithm builds a new mesh following the steps described in section 5.1.4.

5.1.1. Algorithm Design

The design of the parallel edge collapsing method provides generic functions that can be specified to support specific use cases. The collapsing method depends on an array of vertices (3 float numbers per vertex) and an array of oriented tetrahedra (4 integers per tetrahedron). An outline of one collapsing iteration appears in algorithm 7. A placement strategy $\mathcal{P}$ specifies the point to which the edge is collapsed. As many applications enforce specific constraints for collapsing edges, a predicate $\mathcal{Q}$ protects edges from collapsing.

Typically, $\mathcal{Q}$ protects the boundary of the mesh. Many edge collapsing methods for surface meshes rely on quadric error metrics to approximately preserve the shape [GH97; KG03; LZ08; Gha+20], whereas the applications using unstructured tetrahedral meshes typically require exact boundary preservation to obtain accurate numerical simulations. Therefore, a set of rules restricts the collapsing of boundary edges. For this purpose, the massively parallel method in section 4.2 extracts the boundary and classifies vertices into face, ridge and corner using surface normals of surrounding boundary triangles. Only boundary edges along geometrical faces and ridges are collapsed (see fig. 5.1), while other features are protected. A face vertex can be collapsed along the surrounding geometrical faces. A ridge vertex can be collapsed along its two ridge edges.

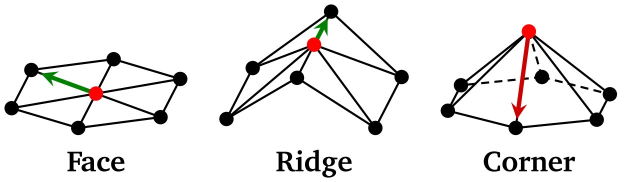


Figure 5.1.: Collapsing (arrows) face or ridge vertices preserves the boundary, while collapsing corner vertices leads to approximation errors.

Empirical investigation revealed that repeated collapsing of ridge edges can distort the boundary, because collapsing very small ridges can distort slightly curved surfaces over the course of many iterations. For this reason, an additional rule prevents the collapsing of very small ridge edges. This rule allows the collapsing of ridge edges, if its two adjacent triangle's normals dot product differs at least ε_R from 1. This thesis consistently uses $\varepsilon_R = 0.1$. Corner vertices cannot be collapsed without altering the boundary respecting ε_F and ε_R. In addition, collapsing an interior edge connecting two distinct boundaries can cause surface artifacts. For this reason, the collapsing algorithm detects boundary edges performing a parallel pass over triangles. If a triangle lies on the boundary, its three edges can be marked as boundary edges. If an edge contains boundary vertices but is not a boundary edge, the coarsening algorithm does not collapse this edge.

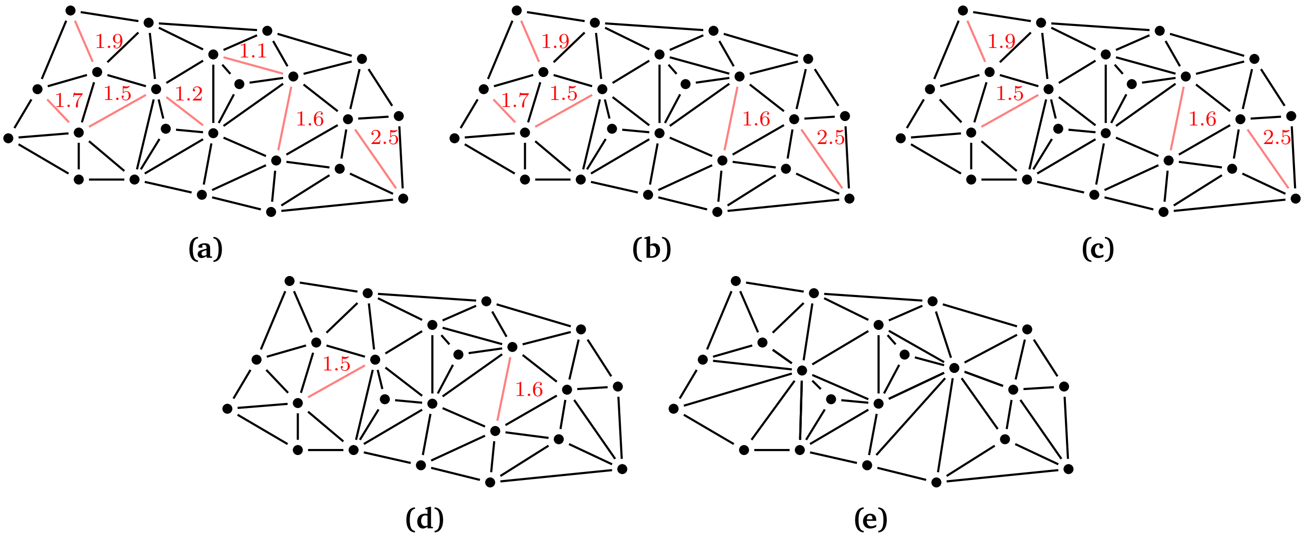
Algorithm 7 Edge collapsing iteration

```
 1: procedure COARSENTETMESH($\mathcal{M}$, $\mathcal{C}$, $\mathcal{P}$, $\mathcal{Q}$, $\varepsilon_c$, edgeMarking)
 2:     edgesMarked ← allocate($\mathcal{M}$.numEdges())
 3:     edgesMarked.fill(0)
 4:     for $i \leftarrow 0, \dots, \mathcal{M}$.numEdges() $- 1$ do                ▷ In parallel
 5:         if edgeMarking[$i$] then
 6:             edgesMarked[$i$] ← 1
 7:         end if
 8:     end for
 9:     $\mathcal{B}$ ← EXTRACTTETMESHBOUNDARY($\mathcal{M}$)
10:     costs ← allocate($\mathcal{M}$.numEdges())
11:     for $i \leftarrow 0, \dots, \mathcal{M}$.numEdges() $- 1$ do                ▷ In parallel
12:         costs[$i$] ← $\mathcal{C}$($\mathcal{M}$.edges[$i$])
13:     end for
14:     COLLAPSEEDGES($\mathcal{M}$, $\mathcal{B}$, costs, $\mathcal{P}$, $\mathcal{Q}$, $\varepsilon_c$, edgesMarked )
15: end procedure
```

Each edge is associated with a cost. The cost to collapse an edge is specified by the cost function $\mathcal{C}$ that is implemented in the system and can be parameterized to enable user specified inputs such as target edge length. With the specification of the cost function, one can control how collapse operations are prioritized. For each collapsing iteration, the algorithm evaluates the cost function in parallel over edges and saves the cost values in an allocated buffer (see lines 11-13 in algorithm 7). The collapsing algorithm performs iterations of parallel edge collapsing until no more collapse operations can be identified or the number of collapse iterations is lower than a user-specified threshold ε_c. The threshold ε_c enables to prevent the overhead of collapsing iterations, which change the mesh insignificantly due to performing a negligible number of edge collapses.

As collapsing edges depends on the mesh connectivity, the collapsing algorithm relies on data structures to lookup mesh connectivity. While the algorithm design is independent from the mesh data structure, it is advised to use a data structure designed for tetrahedral meshes such as OpenVolumeMesh [KBK13] or TCSR mesh (cf. section 2.2.3). This thesis uses TCSR mesh, because it is optimized for GPUs. Furthermore, the collapsing method marks mesh elements throughout the procedure. The marking of an element is represented by altering an entry in an array of marker values. The collapsing algorithm supports adaptive coarsening of tetrahedral meshes, because it collapses only edges that are marked for potential removal. A marker value is either 0 or 1. Specifically, the algorithm uses three arrays of marker values:

- `edgesMarked`: Indicates whether an edge is marked for collapsing "1" or unmarked "0".
- `verticesMarked`: Indicates whether a vertex remains in the mesh "1" or is removed "0".
- `tetrahedraMarked`: Indicates whether a tetrahedron remains in the mesh "1" or is removed "0".



(a) (b) (c) (d) (e)

Figure 5.2.: In (a), red edges with cost values are marked for potential collapsing. The geometrical and topological checks unmark some edges in (b). In (c), conflict detection compares the cost of adjacent edges and prioritizes edges with lower cost. As some simplices are still associated with several marked edges, the algortihm ensures that each simplex is affected by only one collapsed edge in (d). Finally, both edges can be collapsed simultaneously (see (e)).

In addition, the `vertAffectedByEdge` array indicates whether a vertex belongs to a collapsed edge “edge index” or not “-1”.

5.1.2. Finding Admissible Edges for Collapsing

In order to preserve the consistency of the input tetrahedral mesh, it is mandatory to check if a collapse operation produces an invalid mesh. A parallel pass over edges filters for admissible edge collapse operations. First of all, filtering checks if the edge is marked for potential collapsing. To preserve the topological type of the mesh an edge collapse is only admissible, if it satisfies the link condition of Dey et al. [Dey+99]. If the link condition is satisfied, the algorithm tentatively performs the edge collapse using the specified placement $\mathcal{P}$. As the edge collapse operation should not produce inverted elements, the algorithm computes the signed volume $v_\mathcal{T}$ of the resulting tetrahedra using the two one-rings of tetrahedra of the edge vertices. For evaluating the volume of the resulting tetrahedra, the algorithm replaces the positions of the vertices belonging to the collapsed edge with the position of the new vertex. If a tetrahedron contains both edge vertices it can be skipped, because the collapse operation removes the tetrahedron. If the volume of any of the resulting tetrahedra is lower than a threshold ε_v the edge collapse operation is not admissible, because it creates inverted or degenerate tetrahedra. This thesis consistently uses $\varepsilon_v = 2.\times 10^{-12}$ in the implementation. After the topological and geometrical checks, the algorithm evaluates $\mathcal{Q}$ for the edge and refuses the edge collapse operation in case $\mathcal{Q}$ is not satisfied. In this thesis, $\mathcal{Q}$ protects the boundary of the tetrahedral mesh from being altered (cf. section 5.1.1). If all of the checks succeed, the edge remains to be marked for collapsing (see fig. 5.2 (b)). Otherwise, the algorithm unmarks the edge in the `edgesMarked` array.

5.1.3. Finding Independent Sub-Meshes

The collapsing algorithm detects and resolves conflicts in two parallel passes. Since prior checks potentially unmark edges, the first pass over edges checks, if an edge is marked for potential collapsing. For an admissible collapse operation, the method searches for conflicts. The first step of conflict detection is checking adjacent edges. The conflict detection method uses the one-ring of adjacent edges for each of the two vertices of an edge. If any of the adjacent edges is also marked for collapsing and incurs a lower cost, conflict detection prioritizes the edge with the lower cost. In case of equal cost values, conflict detection prioritizes the edge with the lower index. Due to parallel processing over edges, any thread that unmarks an edge does not need to further check the adjacent edges, because this work is handled by other threads. If none of the adjacent edges incurs a lower cost, the edge remains to be marked for collapsing. As a collapse operation replaces an edge with a vertex, only one of the two edge vertices is removed. Tentatively, the algorithm sets the entry in `verticesMarked` of the edge vertex with the larger index to 0. The other edge vertex remains and its position is updated later, if the collapse operation is not rejected by the subsequent parallel pass. In addition, conflict detection records the index of the collapsed edge for both vertices by writing the edge index to `vertAffectedByEdge` at the position of the vertex indices. The recorded edge index entries cannot be overwritten by any other thread, because each thread checks the adjacent edges before writing.

As only checking adjacent edges for conflicts is not enough to prevent invalid collapses (see fig. 5.2 (c)), conflict detection performs a parallel pass over tetrahedra to detect the remaining conflicts. This parallel pass uses the recorded collapse operations in `vertAffectedByEdge` from the previous parallel pass. For each tetrahedron, conflict detection counts how many of the four tetrahedron vertices are marked for removal. Whenever a vertex is marked, conflict detection obtains the index of the to-be-collapsed edge from `vertAffectedByEdge` and writes it to a local stack. The local stack requires at most four entries. If only one of the four vertices is associated with an edge collapse operation no further checks are required. Otherwise, conflict detection potentially requires to resolve conflicts. In order to resolve conflicts, the method iterates over the edge indices recorded in the local stack. If two edge indices in the stack are different, a conflict is found. In this case, the algorithm resolves the conflict by evaluating the cost of both edges and prioritizing the edge with the lower cost. If the two conflicting edges share the same cost value, conflict detection prioritizes the edge with the lower index. The marking for the edge with the larger cost in the `edgesMarked` array is set to 0 and the entries in the `verticesMarked` for the two edge vertices are set to 1, because these vertices remain in the mesh. After the second parallel pass all marked edges can be safely collapsed without producing an invalid mesh (see fig. 5.2 (d) and (e)).

5.1.4. Collapsing Edges

After conflict detection determined a set of non-conflicting collapse operations, the collapsing algorithm builds a new mesh with collapsed edges. As conflict detection already established a valid marking for the `verticesMarked` array, a parallel exclusive prefix scan provides offset positions for vertices and the total number of remaining vertices. Subtracting the number of remaining vertices from the number of input vertices results in the total number of collapse operations. If this number is zero, i.e., no edge is collapsed, or lower than a user-specified threshold ε_c, the algorithm terminates returning the input mesh. Otherwise, the algorithm proceeds with building the resulting mesh with collapsed edges. Using the offset positions for vertices, the remaining vertices are copied to a newly allocated buffer. The next step is to determine a valid marking for the `tetrahedraMarked` array and collapse the marked edges. In a parallel pass over edges, each thread with a to-be-collapsed edge iterates over the tetrahedra containing the edge and sets their entries in the `tetrahedraMarked` array to 0. Subsequently, the thread compares the indices of the two edge vertices, in order to determine the removed vertex with the lower index and the remaining vertex with the larger index. The thread evaluates the placement strategy $\mathcal{P}$ to obtain the new coordinates for the remaining vertex and writes the coordinates to the newly allocated buffer for the remaining vertices using the offset positions. The offset position of the removed vertex is updated to the offset position of the remaining vertex, in order to prepare for building a valid triangulation.

After collapsing the edges in parallel, an exclusive prefix scan over `tetrahedraMarked` obtains offset positions and the number of remaining tetrahedra. The algorithm allocates an array for the tetrahedra of the resulting mesh. A parallel pass over tetrahedra updates the vertex indices of each marked tetrahedron using the offset positions for vertices. Each updated tetrahedron is copied to the array of remaining tetrahedra using the offset positions for tetrahedra. Finally, the remaining tetrahedra and the array of vertices represent the resulting mesh. Applications can then use the resulting mesh to select new edges for adaptive mesh coarsening and re-evaluate the cost function.

5.1.5. Determinism of Conflict Detection

The determinism of an algorithm is an important property, because for some use cases it is coveted to obtain reproducible results. Especially for debugging, the determinism of an algorithm is useful [Pan22], as it enables to trace the control path of an algorithm for multiple executions. For improved run time performance, the steps of the conflict detection in section 5.1.3 read from the same buffer of marking values as they write to the final marking of edges to-be-collapsed. This enables to find more edges for parallel collapsing, because the unmarking, i.e. rejection, of an edge for collapsing takes an immediate effect on the conflict detection. However, since the execution order of threads is not deterministic, it might happen in one execution that one edge is unmarked before checking an adjacent edge, while in another execution the adjacent edge is checked for collapsing before the other edge is unmarked. Therefore, the proposed conflict detection is indeterminstic when executed on massively parallel hardware.

In order to satisfy the potential demand for a conflict detection that finds an independent set of sub-meshes deterministically, this section briefly describes a simple modification of the presented conflict detection to obtain deterministic results. The key idea behind the modification is to introduce an additional, tentative buffer `edgesMarked_tmp` for marking to-be-collapsed edges so that threads can perform checks based on one buffer and write to the other. Thus, before checking adjacent edges (first parallel pass in section 5.1.3) the modified algorithm establishes `edgesMarked_tmp` as a copy of `edgesMarked`. The checks of adjacent edges then use the marking from `edgesMarked_tmp`, while the unmarking of an edge is only written to `edgesMarked`. After the parallel pass terminated, the modified algorithm prepares for the subsequent parallel pass copying the results from `edgesMarked` again to `edgesMarked_tmp`. Analogous to the first parallel pass, the second pass only reads marking values from `edgesMarked_tmp` and only writes marking values to `edgesMarked`. In this way, the unmarking of an edge does not take an immediate effect during the execution of one parallel pass. After the two parallel passes, the tentative buffer `edgesMarked_tmp` can be safely deallocated. As a downside, the modified conflict detection will find fewer edges for collapsing in some iterations, as unmarked edges are not immediately considered. This sometimes leads to up to 10% more iterations of parallel collapsing of edges.

5.2. Collapsing for Mesh Improvement

Since the edge collapse operation allows for the removal of low-quality tetrahedra [Lo14a], it is interesting to incorporate the massively parallel collapsing algorithm into the mesh optimization presented in chapter 4. In addition, harmonic mesh optimization with coarsening can reduce the element count substantially more compared to using harmonic flipping alone, while the optimization should not obtain a mesh of insufficient resolution for numerical accuracy. For this reason, the collapsing should primarily remove low-quality tetrahedra below a certain quality threshold.

For element shape improvement, one can specify the cost function $\mathcal{C}$ and the placement strategy $\mathcal{P}$ so that collapsing improves element quality. As the harmonic mesh optimization efficiently improves dihedral angles, η serves as the cost function for collapsing. Like the mesh optimization algorithm in section 4.3.1, the placement strategy $\mathcal{P}$ performs a line search finding the optimal position for the vertex replacing the collapsed edge. The line search optimizes the sum of η interpolating between the two edge vertices. The cost function $\mathcal{C}$ specifies the improvement in terms of η gained by the collapse operation, prioritizing larger improvement over smaller. While the vertex relocation in section 4.3.1 uses Brent's method [Pre+02], the line search for an collapsed edge uses quadratic fit search (QFS) [KW19], because QFS involves less branches than Brent's method improving parallelism. The key strategy of QFS is to fit a quadratic function to three points a, b and c along the search line, where $a < b < c$. Since η is typically a paraboloid function with one minimum along the search line, QFS is a viable choice. As the line search interpolates between the two edge vertices $\mathbf{v}_0$ and $\mathbf{v}_1$, the possible positions $\ell(x)$ along an edge can be expressed as:

$$\ell(x) = (1 - x) \cdot \mathbf{v}_0 + x \cdot \mathbf{v}_1, \qquad \text{where } x \in [0, 1]. \tag{5.1}$$

In order to prevent numerical robustness issues near the edge vertices, the implementation of the placement strategy consistently uses $a = 0.1$, $b = 0.5$, and $c = 0.9$ for the initial values. Empiric studies in this thesis have shown that a minimum on $\ell(x)$ is often near the midpoint, which QFS finds in few iterations, because it uses three bounds unlike Brent's method. The minimum determined by QFS is a good starting point for further gradient-based optimization with Brent's method. If a collapse operation does not lead to an improvement of η or produces inverted elements, it is inadmissible.

Like Cutler et al. [CDM04], the proposed algorithm couples edge collapse with other mesh improvement operations. The implementation combines GPU-efficient vertex relocation (cf. section 4.3.1) and face/edge swapping (cf. section 4.4) with collapsing edges. One improvement iteration relocates the vertices, finds beneficial face/edge swaps and collapses edges of tetrahedra with a dihedral angle lower than a predetermined threshold.

5.3. A Method for Massively Parallel Mesh Adaptation using Error Estimation

Since section 5.2 describes a way to combine mesh improvement (see chapter 4) with the collapsing algorithm, it is interesting to extend this combination by refinement, in order to obtain a mesh adaptation framework. The massively parallel tetrahedral subdivision refinement algorithm from the author's master thesis [Str19] is an apt choice for this framework, because it is highly efficient and adaptive. In a master thesis under the authors supervision, Stegemann [Ste24] presents a mesh adaptation framework using harmonic mesh optimization coupled with massively parallel coarsening and refinement. As the mesh resolution can significantly impact the accuracy of the simulation results, the focus of Stegemann's thesis [Ste24] lies in the reduction of the error of a numerical simulation.

One useful auxiliary tool to estimate the error of a simulation are a-posteriori error approximators, which perform calculations on the results of a numerical simulation to estimate the error [Lo14b]. Figure 5.3 visualizes an example for mesh adaptation based on a-posteriori error approximation. Stegemann [Ste24] incorporates the a-posteriori error approximation technique to adapt the mesh for a specific upper bound of the estimated error. The mesh adaptation method uses Kelly's estimator [Kel+83] or the estimator of Zienkiewicz and Zhu [ZZ87], because these are well established and their calculation schemes admit efficient, massively parallel computation. With the use of these estimators, the error is associated with the elements $\tau \in \mathrm{T}$. Thus, the a-posteriori error estimators are local, which enables to determine whether a certain element should be adapted or not. The adaptation method computes the refinement index ρ_τ for each $\tau \in \mathrm{T}$:

$$\rho_\tau = e_\tau / e_a,$$

where e_τ is the estimated error of the element τ and e_a is the specified target error. As it is numerically not feasible to obtain the target $\rho_\tau = 1$ for each $\tau \in \mathrm{T}$, the determination criteria for mesh adaptation includes safety bounds to prevent overrefinement. The method consistently refines elements with $\rho_\tau > \alpha_\rho$

and coarsens elements with $\rho_\tau < \beta_\rho$, where $\alpha_\rho \in [1.2, 2.5]$ and $\beta_\rho \in [0.45, 0.65]$. It is important to consider that the refinement of an element does not necessarily reduce ρ_τ. Therefore, a Laplacian smoothing step ensures that the refinement index is well-distributed among the mesh, in order to prevent overrefinement, e.g., at the the fixed boundaries of the mesh.

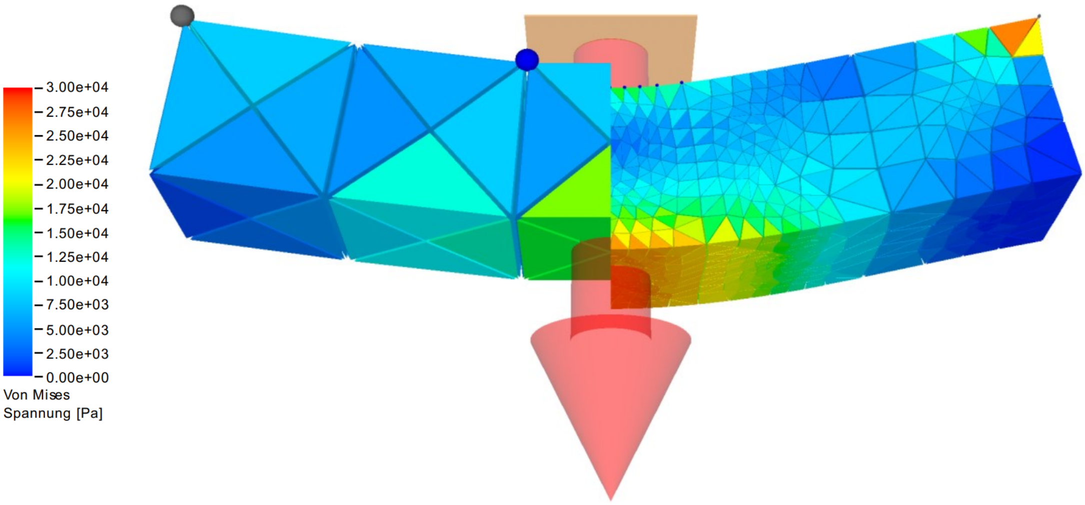


Figure 5.3.: Numerical simulation of the deformation of a bar using the FEM. The coarse mesh to the left leads to a significantly deviating result compared to the fine mesh to the right.

Besides using the error estimation, users like to specify a sizing field $S_T(x)\colon \Omega \to \mathbb{R}^+$ for mesh adaptation. The sizing field specifies the desired size of elements locally at any $\mathbf{x} \in \Omega$. There are various ways to specify a sizing field. Users could use a continuous function or interpolate discrete data. One could also transfer the estimated error to a sizing field [Arp+22]. In order to provide a practical interface for users, the method also supports sizing fields in an analog manner to the refinement index.

In order to optimize the mesh for mesh quality and the estimated error, an extended harmonic mesh optimization sets the function values of the discretized Dirichlet energy (cf. section 2.1.3) to the refinement index. This enables the optimization to scale the volume of elements so that the estimated error is reduced. For harmonic flipping, the evaluation of the harmonic index η uses the factor ρ_τ. For vertex relocation, the computation of the gradient uses the product rule to calculate the gradient of the weighted Dirichlet energy:

$$\frac{\partial \operatorname{tr}(\rho_\tau \mathbf{L}_\tau \rho_\tau)}{\partial \mathbf{x}} = \rho_\tau^2 \frac{\partial \operatorname{tr}(\mathbf{L}_\tau)}{\partial \mathbf{x}} + 2\rho_\tau \operatorname{tr}(\mathbf{L}_\tau) \frac{\rho_\tau}{\partial \mathbf{x}}.$$

While the left summand of the sum is already given by eq. (4.2), the left part of the equation requires the evaluation of $\partial\rho_\tau/\partial\mathbf{x}$. Since ρ_τ is defined on the tetrahedral elements but not on the vertices, Stegemann [Ste24] resorts to replacing $\partial\rho_\tau/\partial\mathbf{x}$ with the longest edge of τ and assumes that the optimization of ρ_τ is positively correlated with the longest edge length. It is worth noting that Alexa [Ale19] states that the derivative of the functional of the discretized Dirichlet energy is undefined on the sub-simplices of τ. Therefore, the replacement of $\partial\rho_\tau/\partial\mathbf{x}$ is an apt choice to optimize the vertex positions for the estimated error. Stegemann's thesis [Ste24] shows that the impact of weighting the Dirichlet energy with ρ_τ is significant for vertex relocation but does not significantly change the results of harmonic flipping.

5.4. Evaluation

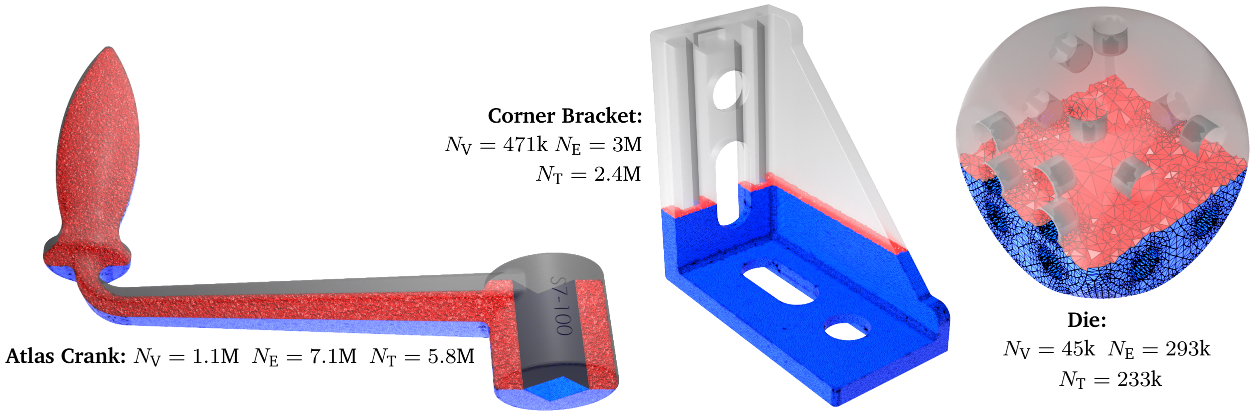


Figure 5.4.: Cross sections visualize the inner structures of the Atlas Crank, Corner Bracket and Die meshes. Labels provide the numbers of vertices (N_V), edges (N_E) and tetrahedra (N_T).

The evaluation in this section shows that the algorithm for massively parallel collapsing is robust and efficiently exploits parallel processing power. Firstly, an automated evaluation procedure applies the algorithm to a multitude of meshes in section 5.4.1 to validate its robustness and correctness. Subsequently, the evaluation focuses on efficiency. Section 5.4.2 and section 5.4.3 investigate the performance of collapsing edges using the Die [ZJ16] (file ID 128640), Corner Bracket [Uga22] and Atlas Crank [Has20] meshes (see fig. 5.4). For performance investigations, the evaluation selects all the edges with a length lower than a predetermined threshold for collapsing. The collapsing of edges terminates, when every edge exhibits a length larger than the threshold or is inadmissible for collapsing. The cost function calculates edge lengths prioritizing smaller edges. The following paragraphs briefly describe three competing algorithms, where sequential collapsing and Gautron et al.'s [GKN23] method are implemented using the TCSR mesh data structure (cf. section 2.2.3) as well:

Sequential collapsing Whenever an edge is admissible for collapsing, it is pushed to a priority queue that prioritizes by edge length. After sequential checking of all the edges, the algorithm pops the topmost edge from the queue until the queue is empty. For every edge, the sequential algorithm first checks if the collapse has been invalidated by a prior collapse. If the collapse has not been invalidated, it is performed. After collapsing the procedure sequentially builds the two arrays of tetrahedra and vertices that represent the collapsed mesh.

Jiang et al.'s [Jia+22] framework The CPU-parallel framework of Jiang et al. [Jia+22] can be used to coarsen tetrahedral meshes accelerated by Intel's Threading Building Blocks. This implementation performs the same admissibility checks as the proposed collapsing algorithm to filter for admissible collapses preserving the boundary. Their scheduler is set up to prioritize smaller over larger edges and terminates, when no more edge can be collapsed. As Jiang et al.'s [Jia+22] framework ships its own data structure, this implementation does not use TCSR mesh.

Gautron et al.'s [GKN23] method This GPU-parallel method performs the steps described in section 5.1.2 and section 5.1.4 but uses a different conflict detection approach. Two parallel passes over the edges determine a set of edges that can be collapsed in parallel. The first parallel pass initially checks if an edge was marked as admissible for collapsing. For admissible edges, the parallel pass creates the descriptors of each edge and propagates the descriptors using CUDA's `atomicMin`. The propagation involves the one-ring of tetrahedra for each of the two vertices of an edge. After edge descriptor propagation, another parallel pass over edges once again checks the one-ring neighborhood of both edge vertices and marks the edge for collapsing if each adjacent tetrahedron is associated with the edge descriptor of the edge.

Section 5.4.4 evaluates the performance impact of skipping collapse operations using ε_c. Section 5.4.5 evaluates the algorithm for mesh improvement presented in section 5.2. The evaluation machine is equipped with an Intel i9-11900K CPU and an NVIDIA RTX 3090 GPU. The implementations of the collapsing variants were compiled using Visual Studio 2022 and CUDA.

5.4.1. Robustness

In order to show that the proposed GPU-parallel collapsing method produces valid meshes, an automated evaluation procedure applied the method to all the $10\,\mathrm{k}$ meshes generated by Hu et al. [Hu+20]. As a large number of edges should be collapsed to evaluate the robustness of the algorithm, the evaluation procedure uses the median edge length of each mesh as the threshold for collapsing.

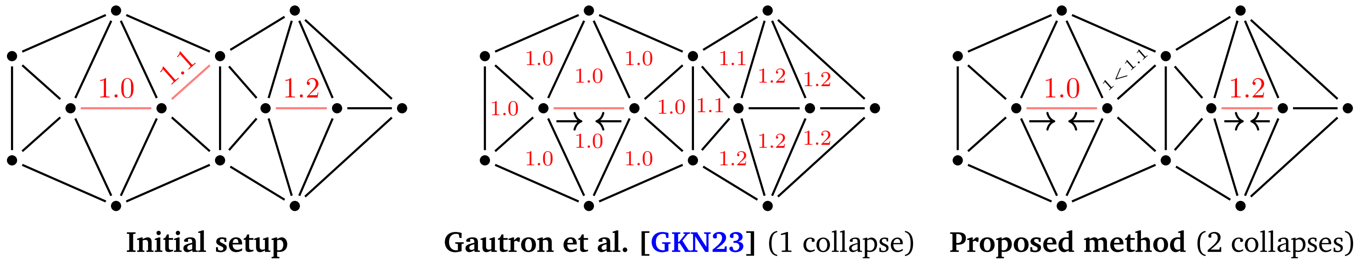
Table 5.1.: Average run times for coarsening the $10\,\mathrm{k}$ meshes of Hu et al. [Hu+20] until convergence ($\varepsilon_c = 0$).

Method	Run time (s)
Proposed	0.180
Gautron et al. [GKN23]	0.202
Jiang et al. [Jia+22] (16 threads)	0.368
CPU-sequential	1.500

The evaluation procedure performs several consistency checks on the resulting meshes. It includes topological checks. Each triangular face should be part of either one or two tetrahedra. In addition, if a triangular face is shared by two tetrahedra, these tetrahedra include this face in alternating orientations. None of the resulting meshes violates the topological checks. Besides the topological checks, the evaluation procedure includes geometrical checks. The procedure checks for inverted tetrahedra. As duplicated vertices pose a problem to many applications, the procedure also checks for duplicated vertices. In the evaluation procedure, two vertices are duplicates, if their pairwise coordinates differ by less than $1. \times 10^{-13}$ on every axis. The method did not produce duplicated vertices or inverted elements on any of the input meshes. Table 5.1 shows the average run times for coarsening the $10\,\mathrm{k}$ meshes with the competing methods. However, the majority of meshes generated by Hu et al. [Hu+20] only includes few elements. Thus, the evaluation measures run time performance on the larger meshes shown in fig. 5.4.

5.4.2. Conflict Detection

As conflict detection is an essential component of parallel re-meshing, the evaluation investigates the density of the resulting sub-meshes compared to the state of the art method of Gautron et al.'s [GKN23]. See fig. 5.5 for a schematic comparison with Gautron et al.'s [GKN23] method. The proposed conflict detection method (cf. section 5.1.3) benefits from the intermediate step unmarking adjacent edges with larger cost values. This intermediate unmarking significantly reduces the potential conflicts for the second conflict detection step. Thus, the second conflict detection step on the basis of tetrahedra finds a compact set of sub-meshes for re-meshing. The resulting sub-meshes can be locally adjacent, because they are non-overlapping.



Initial setup **Gautron et al. [GKN23]** (1 collapse) **Proposed method** (2 collapses)

Figure 5.5.: Initially, three edges are marked (shaded red) for collapsing with cost values (red). Gautron et al.'s [GKN23] method propagates the cost of 1.1 into the cavity of the edge with a cost of 1.2 finding only one collapse operation. The proposed method unmarks the edge with cost 1.1 when checking adjacent edges and finds two collapse operations.

Especially for tetrahedral meshes, the ability of the proposed conflict detection to find a more compact set of sub-meshes results in more parallelism, as conflicts frequently occur in the inner structures of the mesh. In order to show that the proposed conflict detection leads to more parallelism, fig. 5.6 plots the total number of collapsed edges throughout collapsing edges smaller than the median edge length ($\varepsilon_c = 0$) on the Atlas Crank and Corner Bracket meshes for all the three collapsing variants.

As can be seen in fig. 5.6, the proposed conflict detection results in significantly fewer collapsing iterations compared to the atomic operation-based propagation of Gautron et al. [GKN23]. Due to the more compact

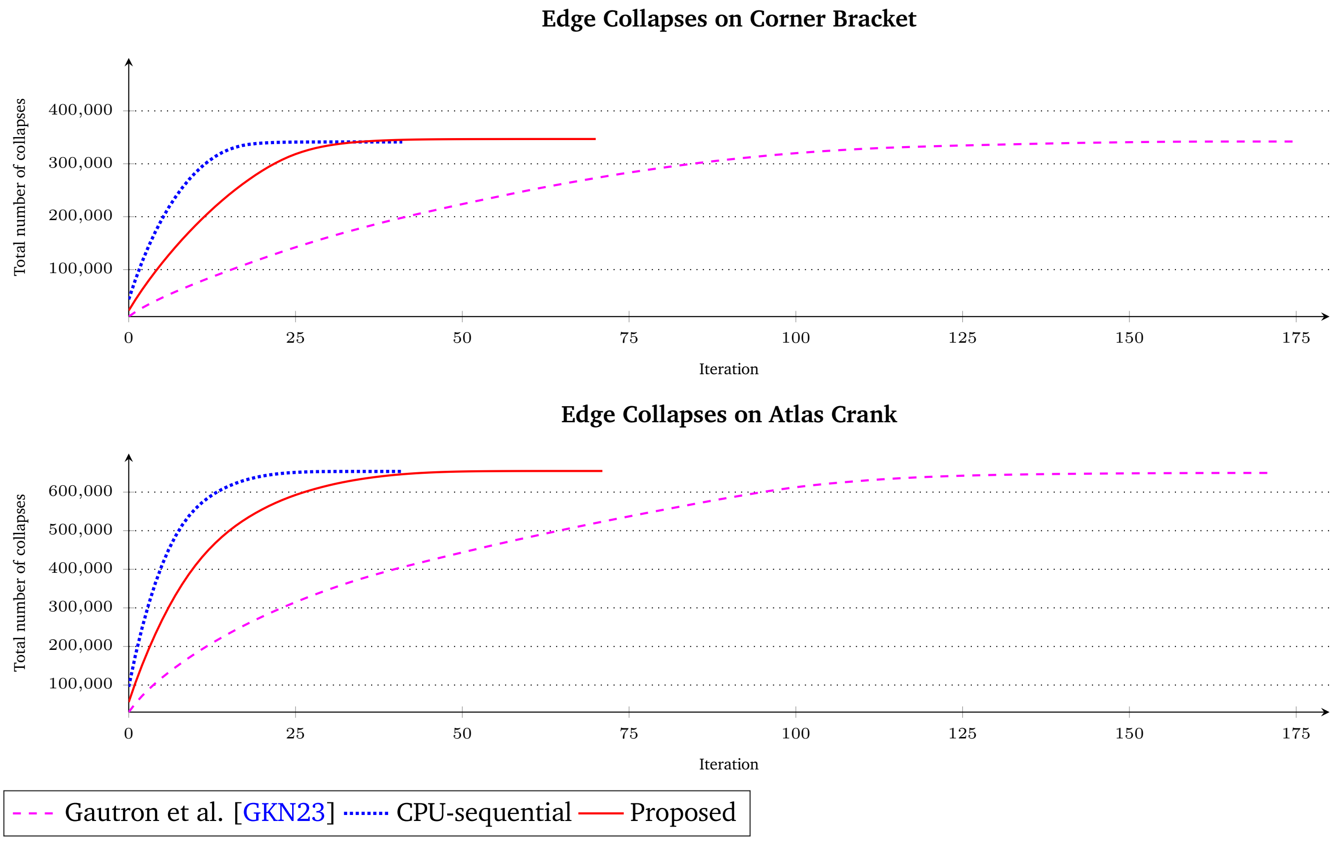


Figure 5.6.: The plots present how the total number of collapses per iteration develops throughout collapsing edges smaller than the median edge length.

set of conflict-free sub-meshes, more collapses can be performed in one single iteration. In fact, the plots exhibit convergence after up to $59\,\%$ fewer iterations compared to the conflict detection of Gautron et al. [GKN23]. Thus, the proposed conflict detection enables efficient exploitation of parallel processing power. As expected, the CPU sequential collapsing variant requires the fewest number of iterations for convergence, because parallel conflict detection tends to reject too many edges for collapsing. Nonetheless, the proposed massively parallel conflict detection achieves spatially dense sets of conflict-free sub-meshes. Compared to the sequential variant, the proposed conflict detection results in up to 1.7× more iterations, which means that only a small overhead of additional iterations is imposed by the decomposition into sub-meshes for parallelization. In contrast, the method of Gautron et al. [GKN23] results in up to 4.2× more iterations compared to sequential collapsing, which means that a substantial amount of additional iterations is imposed due to finding conflict-free collapse operations. The subsequent section 5.4.3 shows that the massive parallelization of the proposed method leads to superior run time performance than sequential collapsing and the method of Gautron et al. [GKN23]. A commonality of all the three collapsing methods is that they perform the bulk of the edge collapse operations in the initial iterations. Thus, after a certain number of iterations the mesh does not change significantly any more, as only few edges are collapsed in each iteration.

Thus, all of the methods tend to spend a significant workload on performing collapsing iterations without a significant effect on the mesh.

5.4.3. Run Time Performance

As the speedup depends on the number of simultaneously collapsed edges per iteration, the evaluation performs measurements for different edge length thresholds, interpolating between the minimal edge length and the median edge length. For each measurement, the evaluation procedure sets $\varepsilon_c = 0$ to collapse edges until no more admissible collapses can be found. As the TCSR mesh data structure (cf. section 2.2.3) requires rebuilding the connectivity relationships for every collapsing iteration, the evaluation includes overall run time measurements including the rebuilding of connectivity relationships and collapsing run time measurements that only involve the steps of collapsing edges (cf. section 5.1.2 to section 5.1.4) abstracting from the data structure of use. Each of the evaluated methods performs the checks described in section 5.1.2 to filter for edges admissible for collapsing. Additionally, measurements for massively parallel collapsing perform 20 repetitions and compute the median run time, because run times of parallel computations may exhibit a multimodal distribution [HB15].

The evaluation measures run times on the meshes shown in fig. 5.4 representing different mesh sizes. Figure 5.7 plots measured run times for edge length thresholds between the minimal and median edge lengths. The proposed massively parallel collapsing method outperforms the CPU-sequential variant by at least one order of magnitude on each of the three meshes. As the Die mesh is the smallest mesh, the proposed method achieves the lowest speedups for this mesh. The proposed collapsing method outperforms the framework of Jiang et al. [Jia+22] by up to 18× for the smaller thresholds and by 2.5× for the median edge length. The method of Gautron et al. [GKN23] exhibits only slightly slower run times than the proposed method, because the Die mesh is coarser in the interior and most conflicts occur near the mesh boundary. More compelling speedups can be found on the Corner Bracket and Atlas Crank meshes. For the Corner Bracket mesh, the proposed method outperforms the CPU-sequential method by 33×, the framework of Jiang et al. [Jia+22] by 7.4× and the method of Gautron et al. [GKN23] by 2.7×. In addition, the proposed method exhibits better scaling behavior due to improved conflict detection. On the Atlas Crank mesh, the proposed method outperforms CPU-sequential collapsing by 34×, the framework of Jiang et al. [Jia+22] by 4.4× and the GPU-parallel method by Gautron et al. [GKN23] by 2.5×. The speedups for the Atlas Crank mesh are slightly lower than for the Corner Bracket mesh, because the Atlas Crank mesh exhibits more thin and curved structures than the Corner Bracket mesh. As a result, the proposed method is significantly faster than the state of the

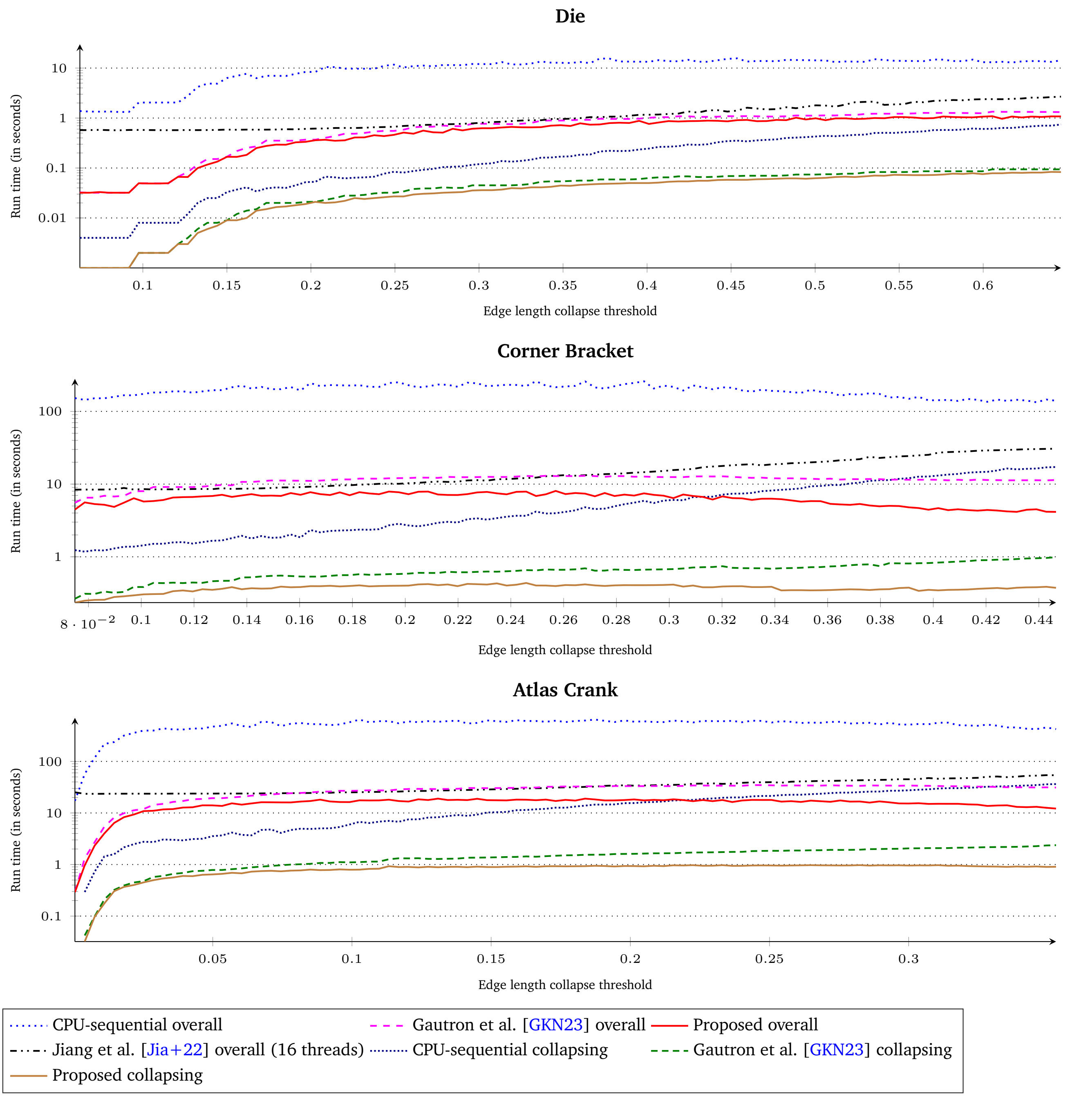
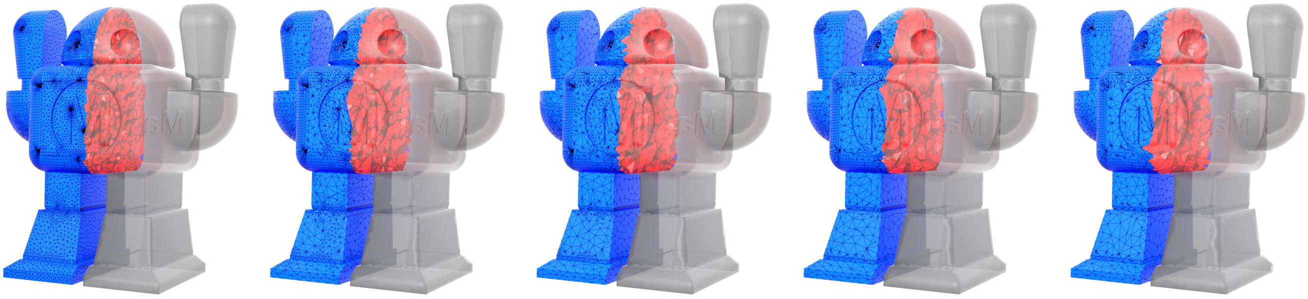


Figure 5.7.: The plots show run times for parallel edge collapsing. The Y-axis is scaled logarithmically.

art. An important commonality of the implementations of CPU-sequential collapsing, the method of Gautron et al. [GKN23] and the proposed method is that the major bottleneck is the rebuilding of connectivity relationships. Thus, a data structure that can be dynamically updated on the GPU would result in significantly improved performance enabling run times close to the run times for collapsing only in fig. 5.7.

5.4.4. Skipping Iterations that Collapse only Few Edges

Since many iterations collapse only a small number of edges (cf. section 5.4.2), the evaluation analyzes the run time performance of the proposed collapsing algorithm for different choices of the threshold ε_c. In order to aggressively coarsen the mesh and impose a considerable workload, the evaluation chooses the doubled median edge length as the threshold for collapsing. It uses the Robot mesh [ZJ16] (file ID 255172) for this evaluation, because it includes thin as well as large inner structures and flat as well as curved boundaries.



Original	$\varepsilon_c = 1000$	$\varepsilon_c = 700$	$\varepsilon_c = 100$	$\varepsilon_c = 0$
–	0.745 seconds	0.981 seconds	1.613 seconds	2.201 seconds
–	33 iterations	44 iterations	91 iterations	138 iterations
$N_\mathrm{T} = 503\mathrm{k}$	$N_\mathrm{T} = 258\mathrm{k}$	$N_\mathrm{T} = 190\mathrm{k}$	$N_\mathrm{T} = 115\mathrm{k}$	$N_\mathrm{T} = 109\mathrm{k}$
$N_\mathrm{E} = 645\mathrm{k}$	$N_\mathrm{E} = 339\mathrm{k}$	$N_\mathrm{E} = 255\mathrm{k}$	$N_\mathrm{E} = 161\mathrm{k}$	$N_\mathrm{E} = 153\mathrm{k}$
$N_\mathrm{V} = 101\mathrm{k}$	$N_\mathrm{V} = 55\mathrm{k}$	$N_\mathrm{V} = 42\mathrm{k}$	$N_\mathrm{V} = 28\mathrm{k}$	$N_\mathrm{V} = 27\mathrm{k}$

Figure 5.8.: Coarsening the Robot mesh with different values for ε_c results in different meshes and run times. The figure provides run times and numbers of tetrahedra (N_T), edges (N_E) and vertices (N_V) for the resulting meshes.

Table 5.2.: Run times and speedups (of the proposed method) for coarsening the Robot mesh [ZJ16] (file ID 255172) until convergence ($\varepsilon_c = 0$).

Method	Run time (s)	Speedup
Proposed	2.201	–
Gautron et al. [GKN23]	6.072	2.76×
Jiang et al. [Jia+22] (16 threads)	11.964	5.44×
Jiang et al. [Jia+22] (8 threads)	14.296	6.50×
CPU-sequential	29.488	13.40×

For $\varepsilon_c = 0$, Table 5.2 shows run times and speedups of the proposed method compared to the other methods. An overview on how the Robot mesh and the run time of the proposed method develops while increasing ε_c appears in fig. 5.8. As can be seen, the run time significantly increases for choosing a low number for ε_c. For the jump from $\varepsilon_c = 700$ to $\varepsilon_c = 100$, the run time of the proposed method almost doubles, meaning that many iterations only collapse hundreds of edges. Taking a close look on the boundaries for $\varepsilon_c = 700$ and $\varepsilon_c = 100$, it can be seen that these iterations primarily coarsen mesh regions with many short edges connected to each other. Conflicts frequently occur in these regions limiting the impact of parallel processing. Nonetheless, choosing $\varepsilon_c = 100$ eliminates only $75\,\text{k}$ more tetrahedra than choosing $\varepsilon_c = 700$. Thus, the impact on coarsening these local regions is not as substantial as coarsening the remainder of the mesh. The proposed method achieves a fast run time of 981 milliseconds (below one second) for choosing $\varepsilon_c = 700$, which provides means for immediate response times [New94]. The experiment validates that skipping collapsing iterations performing few collapses comes at low cost, because the bulk of the decimation happens in the initial iterations. For choosing $\varepsilon_c = 1000$, the proposed collapsing method achieve to halve the number of tetrahedra. For $\varepsilon_c = 700$, the proposed method can substantially reduce the number of tetrahedra.

5.4.5. Mesh Quality Improvement of Harmonic Optimization with Collapsing

Since section 5.2 proposes to combine the massively parallel collapsing method with the element quality optimization from chapter 4, the evaluation investigates the impact of this combination. While the resulting optimization algorithm has been tested for many models, the evaluation discusses the results for two meshes Die and Goyle, which are representative for the evaluation results. For each mesh, the optimization algorithm collapsed tetrahedra with a dihedral angle lower than $13\,^\circ$ until convergence ($\varepsilon_c = 0$). The Die mesh shows the benefits of combining vertex relocation, flipping and collapsing. Since limitations occur for models with highly-complex boundary features, the evaluation includes the Goyle mesh to show the limitations of the optimization method. The evaluation provides resulting mesh sizes, run times, and element quality in terms of the minimal dihedral angle φ_{min} and conformal AMIPS energy $\mathcal{D}^{\text{AMIPS}}$ (cf. section 3.2.2). Each measurement is repeated 20 times, in order to prevent outliers. If a method provides deterministic results, the measurement calculates the median run time. If a method is indeterministic, the evaluation determines the range of measured run times and element qualities. Figure 5.9 shows the results for both meshes.

For many meshes, the use of collapsing for mesh improvement is beneficial (see Die mesh in section 3.2.2). Collapsing edges additional to flipping elements results in a lower number of elements in the optimized mesh. While the harmonic mesh optimization significantly reduces the number of low-quality elements, the additional use of collapsing eliminates more low-quality elements. The more simple the surface features of the mesh are, the better is the benefit due to using collapsing. For meshes with a good portion of flat surface features like the Die mesh, the collapsing of low-quality elements frequently results in not only fewer low-quality elements but also better quality for the worst element. This is due to collapsing eliminates the elements with the worst quality and subsequent optimization steps obtain better element quality for the remaining elements. In some cases, collapsing even achieves to eliminate some low-quality elements at complex boundary features that cannot be easily optimized by vertex relocation and flipping alone. Mostly, the low-quality elements have an interior edge in these cases. The interior edge can then be collapsed so that the resulting element quality can be improved by the optimization. The run time performance of the element quality optimization frequently benefits from collapsing, if the number of low-quality elements in the mesh is much lower than the overall number of elements in the mesh. Otherwise, many iterations of parallel collapsing are necessary to eliminate all the low-quality elements. Due to the use of indeterministic coloring for vertex relocation, the exact run time of the optimization algorithm is difficult to predict. As different results for vertex relocation can lead to different choices of the collapsing algorithm, the use of edge collapsing further increases the variety of different run time behaviors.

While the additional usage of collapsing is beneficial for many meshes with flat surface features, the meshes with many complex and rounded surface features oftentimes benefit only a little by the usage of the collapsing method (see Goyle mesh in section 3.2.2). However, the combination of vertex relocation, flipping and coallpsing typically removes more low-quality elements than using vertex relocation and flipping alone. If a large portion of the elements in the mesh are of low shape quality, then collapsing edges for mesh improvements does not improve the run time performance of the optimization, since many collapsing operations are performed before termination. The use of collapsing for element quality optimization does not guarantee

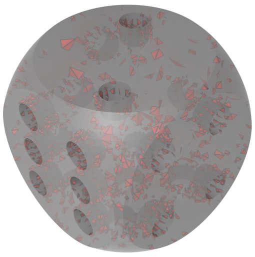
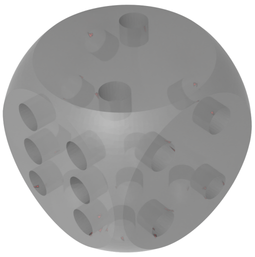
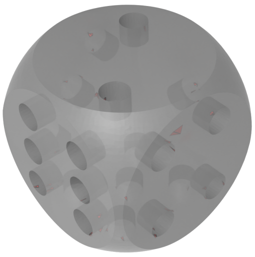
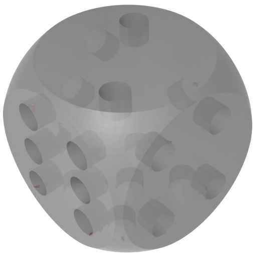
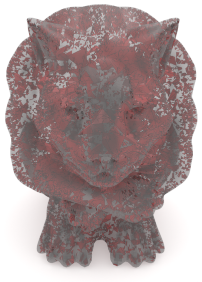
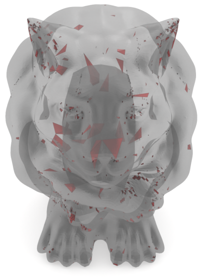
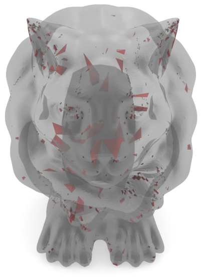
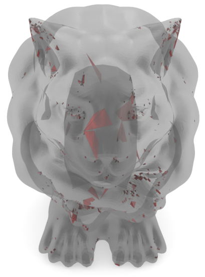
Original –	**Improved w\o collapsing** Deterministic coloring	**Improved w\o collapsing** Deveci coloring	**Improved w\ collapsing** Deveci coloring
			
2269 low-quality tets – $N_\mathrm{T} = 233\mathrm{k}$ $N_\mathrm{V} = 45\mathrm{k}$ $\varphi_{\min} = 2.07$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 28.51$	60 low-quality tets 808 ms $N_\mathrm{T} = 226\mathrm{k}$ $N_\mathrm{V} = 45\mathrm{k}$ $\varphi_{\min} = 7.59$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 20.14$	62 - 78 low-quality tets 264 ms - 601 ms $N_\mathrm{T} = 226\mathrm{k}$ $N_\mathrm{V} = 45\mathrm{k}$ $\varphi_{\min} = 7.29\text{-}7.67$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 19.55\text{-}19.97$	**27 - 34 low-quality tets** 267 ms - 513 ms $N_\mathrm{T} = 221\mathrm{k}$ $N_\mathrm{V} = 44\mathrm{k}$ $\varphi_{\min} =$ **7.87-9.38** $\mathcal{D}^{\mathrm{AMIPS}}_{\max} =$ **14.77-18.36**
			
20304 low-quality tets – $N_\mathrm{T} = 518\mathrm{k}$ $N_\mathrm{V} = 142\mathrm{k}$ $\varphi_{\min} = 0.50$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 283.50$	1530 low-quality tets 808 ms $N_\mathrm{T} = 503\mathrm{k}$ $N_\mathrm{V} = 142\mathrm{k}$ $\varphi_{\min} = 0.81$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 447.89$	1511 - 1612 low-quality tets 894 ms - 2.087 s $N_\mathrm{T} = 503\mathrm{k}$ $N_\mathrm{V} = 142\mathrm{k}$ $\varphi_{\min} =$ **0.99-1.77** $\mathcal{D}^{\mathrm{AMIPS}}_{\max} =$ **195.96-223.15**	**954 - 1180 low-quality tets** 4.453 s - 11.171 s $N_\mathrm{T} = 469\mathrm{k}\text{-}471\mathrm{k}$ $N_\mathrm{V} = 134\mathrm{k}\text{-}135\mathrm{k}$ $\varphi_{\min} = 0.10\text{-}1.25$ $\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 238.42\text{-}676.47$

Figure 5.9.: Elements (red) with a dihedral angle ϕ lower than 13° before and after improvement. For the indeterministic methods, the figures show the worst cases of remaining low-quality tetrahedra. Better element qualities are provided in boldface.

to provide the target element quality, because not every edge can be collapsed (cf. section 3.2.3), i.e., not every low-quality element can be eliminated. Furthermore, the optimization algorithm does not guarantee to improve the worst quality element, as the optimization consistently minimizes the sum of the harmonic indices and converges in a local minimum. Whenever an edge is collapsed, there is no guarantee that the subsequent optimization steps find a local minimum, where the worst quality element of the resulting mesh is better than before the collapsing. Therefore, it is in line with the expectations that element quality improvement with collapsing does not improve the worst quality elements in some meshes with complex boundary features. As high-resolution meshes with complex boundary features oftentimes require more iterations until a mesh optimization method converges, it can be expected that a larger set of re-meshing operations can impose reduced run time performance. It is possible to consistently improve element quality of complex meshes with the use of a roll back mechanism for re-meshing operations and performing exhaustive search until a sequence of re-meshing operations has reached a better global minimum [KS07]. This imposes the cost of substantially reduced run time performance. Consequently, this thesis does not investigate roll back of re-meshing operations for the quality improvement.

5.5. Summary

In summary, this chapter has introduced a robust, massively parallel and configurable method for collapsing edges in an unstructured tetrahedral mesh. The core of this method is a massively parallel conflict detection that determines densely packed sub-meshes for conflict-free re-meshing. This conflict detection satisfies all of the desired properties. The resulting sub-meshes:

- are non-overlapping so that massively parallel re-meshing produces valid meshes,
- can be spatially dense or even adjacent so that one massively parallel pass can perform many re-meshing operators, and
- are constructed with respect to a cost function $\mathcal{C}$ so that re-meshing operations with the most desirable impact on the mesh.

Evaluating the collapsing method on 10 k of unstructured tetrahedral meshes has shown the robustness of the proposed method, since the resulting meshes were all valid. Massively parallel collapsing of edges achieves a speedup of one order of magnitude over sequential collapsing. In addition, the evaluation reveals that the proposed collapsing method is significantly faster than state of the art parallel collapsing methods. The evaluation has shown that the main bottleneck is rebuilding the TCSR data structure, which suggests a dynamic GPU data structure for unstructured tetrahedral meshes for future work.

Returning to RQ1[1] and RQ2[2], this chapter allows further argumentation. As many re-meshing operators can be accelerated with the use of the proposed conflict detection, the goal of massively parallel re-meshing is achieved for foundational re-meshing operators. The method described in this chapter allows for quickly re-meshing unstructured tetrahedral meshes using the proposed algorithm, which is the acceleration of another important VP task that is concerned with RQ1. Thus, this chapter establishes further arguments for a positive answer to RQ1. However, the answer to RQ2 is not as simple. In the evaluation, the massively parallel algorithms have produced valid meshes on a large test set. The success of the proposed algorithms depends on the complexity of the mesh boundary. For complex boundaries, it is generally difficult to quickly optimize the mesh with a small number of operators and without backtracking. The proposed massively parallel algorithms inherit this issue (cf. section 5.4.5) and cannot guarantee good mesh quality, while they typically produce sufficient mesh quality on meshes with moderately complex boundaries.

Since mesh adaptation is a typical re-meshing task in VP cycles, the successful use of the proposed massively parallel algorithms for adapting meshes, e.g., demonstrated in section 5.3, allows to argue in the light of RQ3[3] that mesh adaptation in VP cycles can benefit from the proposed algorithms. Respective RQ4[4] one can claim that the coarsening of meshes reduces memory occupation for post-processing. Before this thesis evaluates the abilities of the proposed coarsening method to achieve memory-efficient post-processing in chapter 7, the discussion of RQ3 is extended to the customization of prototypes in chapter 6.

[1] RQ1: *How can the use of the GPU accelerate mesh optimization and re-meshing tasks for unstructured tetrahedral meshes?*

[2] RQ2: *How can massively parallel optimization and re-meshing of unstructured tetrahedral meshes robustly produce meshes of sufficient quality for numerical simulations?*

[3] RQ3: *How can massively parallel mesh processing be used for quick editing of unstructured tetrahedral meshes to accelerate VP cycles?*

[4] RQ4: *How can the massively parallel post-processing of unstructured tetrahedral meshes be implemented with efficient memory usage?*

6. User-guided Unstructured Tetrahedral Mesh Editing

The following papers contain the core content of this chapter:

[Str+21] **D. Ströter**, U. Krispel, J. Mueller-Roemer, D. Fellner, "TEdit: A Distributed Tetrahedral Mesh Editor with Immediate Simulation Feedback". In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Presented at SIMULTECH 2021. SciTePress 2013. SCITEPRESS - Science and Technology Publications, 2021, pp. 271–277. DOI: 10.5220/0010544402710277

[Str+23] **D. Ströter**, A. Halm, U. Krispel, J. S. Mueller-Roemer, D. W. Fellner, "Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation". In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Gerd Wagner, Frank Werner, Tuncer Oren, and Floriano De Rango. Springer International Publishing, 2023, pp. 24–42. DOI: 10.1007/978-3-031-23149-0_2

[Str+24] **D. Ströter**, J. M. Thiery, K. Hormann, J. Chen, Q. Chang, S. Besler, J. S. Mueller-Roemer, T. Boubekeur, A. Stork, D. W. Fellner, "A Survey on Cage-based Deformation of 3D Models". In: *Computer Graphics Forum* 43.2 (May 2024). Presented at EUROGRAPHICS 2024. DOI: 10.1111/cgf.15060

Subsequently to investigating low-level mesh editing (cf. section 3.1) in chapter 4 and chapter 5, this chapter presents methods for user-interactive editing of unstructured tetrahedral meshes. While the previously presented methods primarily addressed RQ1 and RQ2, i.e., the acceleration of optimization and re-meshing in the VP cycle, this chapter attempts to introduce methods that shorten the VP cycle by changing the tetrahedral mesh of the prototype without loop back to CAD. Therefore, this chapter investigates RQ3 beyond the acceleration of typical steps in the VP cycle in an attempt to establish high-level mesh editing of unstructured tetrahedral meshes. In order to extensively investigate the potential of high-level mesh editing for VP cycles, this chapter covers two of the well-established methods to interactively edit a mesh: feature-based editing (cf. section 3.3.1) and deformation-based editing (cf. section 3.3.2).

Feature-based editing should be compatible with the semantic features defined on the B-Reps in CAD, because this allows for a smooth transition from CAD to high-level mesh editing. However, editing the shape of the mesh invalidates this initial assignment of semantic features. Thus, a quick method to detect semantic features based on the mesh surface should be helpful to establish feature-based mesh editing. Mesh editing tools for professional VP purposes such as Autodesk's Fusion offer the determination of face groups [Aut24] to organize triangular surface faces of a mesh into semantic groups. However, the accurate determination of these face groups can impose slow run time performance, especially on high-resolution models. Therefore, this thesis attempts to introduce a massively parallel algorithms for fast run time performance of face group detection in sections 6.1.2 and 6.1.3. Additionally, this chapter attempts to extend the mesh editing based on face groups from surfaces to the editing of unstructured volumetric meshes for numerical simulation purposes. This requires measurements for providing valid meshes with sufficient element quality for numerical simulation. In order to establish proof of concept for feature-based editing of unstructured tetrahedral meshes, this chapter attempts to formulate two editing operations: hole close and erosion. The hole close operation enables the removal of co-planar holes in the mesh so that users can quickly customize models, e.g., for manufacturing purposes, and immediately evaluate the impact on simulation scenarios. The erosion operation enables the removal of thin user-selected model parts. As both of these operations should produce meshes for FEA, the element qualities of the resulting meshes are an important part of the evaluation. Since interactive editing is desirable, the use of the GPU to accelerate the mesh editing is also an interesting avenue that is explored in this chapter.

For addressing the deformation-based editing of unstructured tetrahedral meshes, cage-based deformation is an interesting candidate for unstructured tetrahedral mesh editing. Its deformation update scheme consists of linear affine sums (cf. section 2.3), which is suitable for massively parallel GPUs. In addition, a lot of advances in the past decade have significantly extended the facilities of cage-based deformation to provide shape preserving and robust deformation [Str+24]. Moreover, the possibilities to quickly generate a good cage for deformation control also improved in versatility and convenience. Although the cage-based deformation approach offers benefits over many well-established interactive deformation methods (cf. section 3.4), this approach has rarely been applied for VP purposes. For this reason, an extensive investigation of the possibilities of cage-based deformation in the field of 3D modeling can be helpful for finding out which methods can be applicable to shorten VP cycles with the cage-based editing of unstructured tetrahedral meshes. Therefore, this chapter extensively evaluates the facilities of cage-based deformation of 3D models.

The evaluated methods have been implemented in three different tools:

- **TEdit:** This tools enables the modification of unstructured tetrahedral meshes based on face tags that originate from CAD. After each modification a simulation feedback is triggered. For efficiency, a massively parallel FEM algorithm [Web+15] performs the numerical simulation.

- **TetMeshInteract:** This tool enables to quickly detect semantic features with a configurable detection algorithm using the surface of the unstructured tetrahedral mesh.

- **CageModeler:** This toolset provides cage-based deformation using the relevant approaches. The toolset is available at `https://github.com/DanStroeter/CageModeler`.

As interaction with the mesh typically relies on the boundary, section 6.1 presents the concept of face groups that represent semantic features. Subsequently, section 6.2 presents editing operations on the basis of these face-groups. A critical evaluation of these editing operations appears in section 6.3. To evaluate the capabilities of different cage-based deformation approaches section 6.4 discusses and evaluates the approaches for cage-based deformation in the context of 3D models. Finally, section 6.5 summarizes the key conclusions of this chapter.

6.1. Face Groups for Interactive Mesh Modification

Given an unstructured tetrahedral mesh, a face group represents a set of adjacent boundary faces for the user to select with a single mouse click. For interactive mesh editing, face groups enable users to select parts of the model with only few interactions. The selected face groups represent a semantic feature that the user intends to modify. Editing operations are applied to the selected faces of the face groups. Figure 6.1 shows the interaction with face groups in TEdit.

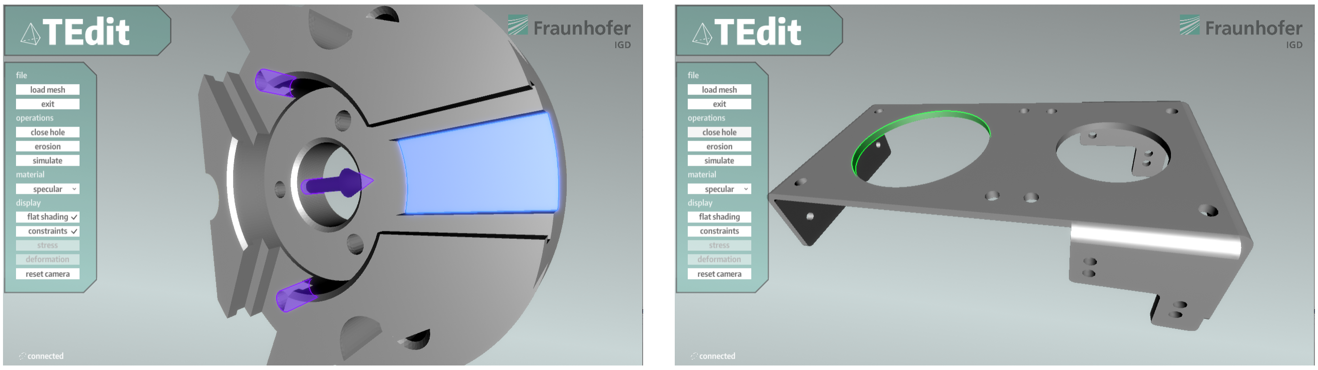


Figure 6.1.: When the cursor hovers over a face group, a blue shaded region appears on the mesh to indicate a selectable face group (left). A selected face group is colored green (right).

The concept of face groups generally applies to surface meshes, whereas unstructured tetrahedral meshes include many interior faces. Consequently, an editing system needs to extract boundary faces from the input mesh to construct face groups. Section 6.1.1 describes the extraction of faces associated with CAD surfaces. For situations where these tags are unavailable, section 6.1.2 presents a massively parallel algorithm to extract face groups from surface features in a parametrized manner.

6.1.1. Assigning Face Groups using Annotation from CAD

Virtual prototyping oftentimes starts in CAD, where several CAD faces typically constitute a continuous representation of the model's surface. For the extraction of face groups, it is practical to exploit that generating volumetric meshes from CAD faces frequently follows a hierarchical scheme. Hierarchical mesh generation from CAD surfaces can be (over-)simplified as follows:

1. Generate a surface mesh that forms an accurately enough representation of the model shape.
2. Tag surface polygons with the ids of CAD faces they originate from.
3. From the resulting surface mesh, generate a volumetric mesh with boundary faces tagged by the ids of CAD faces.

One of the most frequently used mesh file formats used for maintaining and exchanging unstructured tetrahedral meshes is the MSH format [GR23]. The MSH format provides optional tag fields. Usually, the optional tag field includes the identifier of the geometrical entity a mesh element belongs to. Hierarchical meshing tools such as GMSH [GR09] enable to provide the generated mesh in MSH format tagging the boundary faces with the CAD faces they belong to.

6.1.2. Extracting Face Groups from the Surface Geometry

If the data format of the mesh does not include face tags from CAD faces, the modeling application cannot provide interactive editing with semantical face groups. In order to enable interactive editing in such a situation, this thesis presents a method to quickly group boundary faces based on the surface geometry. As this method extracts face groups from a boundary mesh, the term "face group extraction" denotes this method in this thesis. The face group extraction is the result of a practical lab by González and Licheva [GL22] under the supervision of the author of this PhD thesis.

The key idea of the face group detection is to analyze the surface curvature of a model and assign faces to separate groups, if the curvature on the shared edge exceeds a user-specified ridge angle γ_r. As a result of the user-guided parametrization, the user can generate the face groups, which are suitable for the user-intended editing operations. For interactive run time performance, the face group detection performs calculations in sequences of massively parallel passes over the surface mesh elements using the TCSR data structure (cf. section 2.2.3). To ensure robustness, the face group extraction uses the robust singularity analysis of the quadric metric tensor presented by Jiao [Jia06], which is amenable to massively parallel execution. Nonetheless, the face group detection can be implemented with any method to analyze surface curvature, though the method of choice should provide reasonable efficiency to provide interactivity.

The remainder of this section describes the overall method of the face group extraction. The fist step is to classify boundary vertices into face, ridge and corner. For this purpose, one can either compare the normals of surrounding boundary triangles as described in section 4.2 or classify the vertices using the quadric error metric approach from Jiao [Jia06]. The implementation uses the method of Jiao [Jia06] as it enables classification based on user-defined angle thresholds, which allows for user-guided control of the resulting face groups. The quadric metric tensor $\mathbf{A_v}$ for a vertex $\mathbf{v}$ is assembled of weights and the normals of surrounding surface triangles:

$$\mathbf{A_v} = \mathbf{N_v}^{\top}\mathbf{W_v}\mathbf{N_v}, \tag{6.1}$$

where $\mathbf{N_v}$ is composed of surrounding surface face normals and $\mathbf{W_v}$ is a diagonal matrix of weights for each face. Let us suppose that $N_t^{\mathbf{v}} \in \mathbb{N}$ denotes the number of surface faces surrounding $\mathbf{v}$. The size of these matrices can be expresses as $\mathbf{W_v} \in \mathbb{R}^{N_t^{\mathbf{v}} \times N_t^{\mathbf{v}}}$, $\mathbf{N_v} \in \mathbb{R}^{N_t^{\mathbf{v}} \times 3}$, and therefore $\mathbf{A_v} \in \mathbb{R}^{3\times 3}$. As the number of available registers per thread is limited (cf. section 2.2.1), the implementation does not assemble $\mathbf{A_v}$ using eq. (6.1), but takes advantage of a rewritten form:

$$\mathbf{A_v} = \mathbf{n}_1 \cdot w_1 \cdot \mathbf{n}_1^{\top} + \mathbf{n}_2 \cdot w_2 \cdot \mathbf{n}_2^{\top} + \ldots + \mathbf{n}_{N_t^{\mathbf{v}}} \cdot w_{N_t^{\mathbf{v}}} \cdot \mathbf{n}_{N_t^{\mathbf{v}}}^{\top},$$

where w_i and $\mathbf{n}_i$ represent the weight and normal of the i-th face, respectively. This way, each thread allocates a 3×3 matrix and adds the respective normals multiplied by weights until $\mathbf{A_v}$ is assembled.

As this thesis focuses on virtual prototyping, the calculation of weights follows the recommendation of Jiao [Jia06] for applications in VP and sets w_i to the angle at $\mathbf{v}$ to its i-th incident face. In order to classify $\mathbf{v}$, a singular value decomposition of $\mathbf{A_v}$ is calculated [Jia06]. The implementation uses the Jacobi method from the book of Press [Pre07], because $\mathbf{A_v}$ is a real symmetric positive semi-definite matrix with real, non-negative eigenvalues. The resulting Eigenvalues λ_1, λ_2 and λ_3 and corresponding Eigenvectors e_1, e_2 and e_3 are ordered such that $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Subsequently, the classification of the vertices can be inferred from ratios of Eigenvalues [Jia06]. This enables to classify vertices into the categories discussed in section 4.2:

- **CORNER**: It holds that $\lambda_3/\lambda_1 \geq \chi_c$ or $360° - \sum_{i=1}^{N_t^{\mathbf{v}}} w_i \geq 90°$.
- **RIDGE**: It holds that $\lambda_2/\lambda_1 \geq \chi_r$ or $e_1^{\top}\mathbf{n}_i \leq 0$ for any $i = 1, 2, \ldots, N_t^{\mathbf{v}}$.
- **FACE**: None of the above conditions are satisfied.

The thresholds $\chi_c = 2\tan(\gamma_c/2)^2$ and $\chi_r = \tan(\gamma_r/2)^2$ are calculated using the corner angle γ_c and the user-specified ridge angle γ_r. As γ_c does not affect the classification into face groups, the face group extraction constantly uses $\gamma_c = 45°$. Conceptually, γ_c and γ_r are thresholds on the opening angle and the dihedral angle of triangle normals, respectively. A small $\chi_r \approx 0$, i.e., $\gamma_r \approx 0°$, leads to virtually every vertex being classified as a ridge vertex. Analogously, a large $\chi_r \approx 1$, i.e., $\gamma_r \approx 90°$, leads to virtually none of the vertices being classified as ridge vertex. The face group extraction performs the vertex classification step in one massively parallel pass over surface vertices. If the input is a tetrahedal mesh, the procedure in section 4.2 can be used to extract the boundary, where the classification step is replaced with the eigenvalue-based scheme to allow for configuration with an input ridge angle.

In order to determine the face groups on the basis of the classified vertices, a propagation algorithm is used. To prepare for the propagation, the neighboring faces are determined by lookup of the shared edges. The key idea is to initially assign each triangle a unique face group and propagate over shared edges, while ridge edges are ignored. A ridge edge connects two vertices that are not classified as face vertices. As two adjacent triangles can be coplanar, while the shared edge connects two ridge vertices, it is not sufficient to define ridge edges only on the basis of classified types of their vertices. Therefore, an additional criterion for a ridge edge is that the angle between the two triangles is larger than γ_r to ensure that the triangles are not co-planar:

$$\text{isRidgeEdge}(\mathbf{v}_1, \mathbf{v}_2, t_1, t_2) = \textit{type}(\mathbf{v}_1) \neq \textbf{FACE} \wedge \textit{type}(\mathbf{v}_2) \neq \textbf{FACE} \wedge \textit{angle}(t_1, t_2) > \gamma_r,$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the two vertices of the edge shared by triangles t_1 and t_2. Thus, in parallel over triangles, the face group extraction checks whether each of the three triangle edges are ridge edges and stores the index of the three adjacent triangles. This results in a triplet of integers for each triangle. If a triangle edge is not a ridge edge, the respective integer is the index of the adjacent triangle. Otherwise, the respective integer is set to -1 indicating that propagation at that edge is prohibited. In addition, the algorithm constructs a marking array that identifies ridge edges with a marking value of 1. Non-ridge edges are identified with a marking value of 0. The marking array allows to quickly extract the ridge edges using a stream compaction (cf. section 2.2.2).

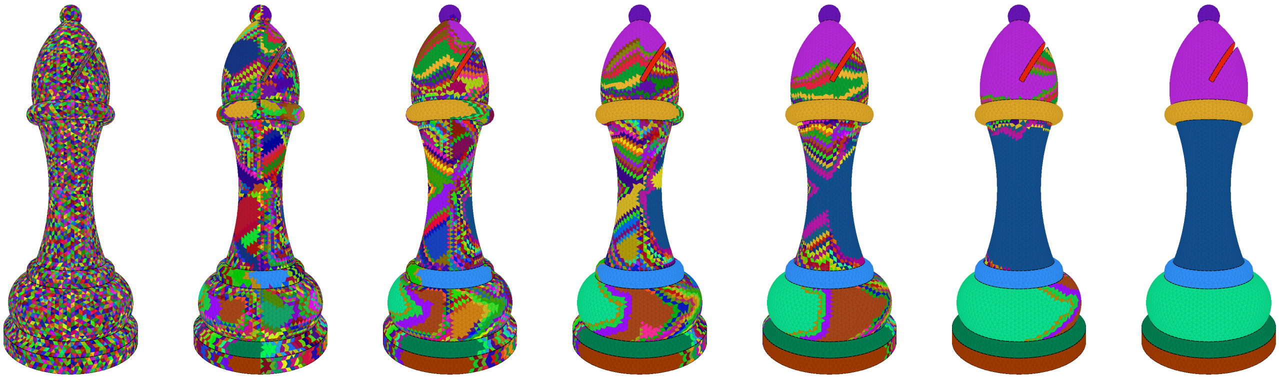


Figure 6.2.: Visualization of the face group id propagation on a chess bishop model.

The subsequent propagation phase determines the face groups. Initially, each triangle is assigned its own unique face group that is associated with its index. Throughout propagation, each triangle writes its face group id to its neighbors, but only if its own face group id is larger than the id of the neighbor. Consequently, all the triangles of one face group eventually will be associated with the same id. Since writing beyond ridge edges is prohibited, each face group is eventually represented as a group of triangles sharing the same id. A visualization of the face group id propagation appears in fig. 6.2. One propagation pass processes all the triangles of the surface mesh in parallel. The face group extraction performs propagation passes until no change in the face group assignment occurs in one pass.

6.1.3. Finding Feature Edges between Face Groups

While the algorithm in section 6.1.2 determines ridge edges to extract face groups, it does not provide the exact edges separating the extracted face groups, because the propagation is only restricted by closed loops of ridge edges. However, for visualization it can be difficult for users to separate many face groups only by color. Additionally, mesh editing needs to be restricted to the user-selected parts of the mesh. Thus, this thesis includes as algorithm to extract the ridge edge loops that separate face groups and ignores the strings of edges inside only one single face group.

The algorithm to determine the edge loops between face groups depends on the surface mesh of the model and the marking of ridge edges calculated in section 6.1.2. In order to prepare for loop finding, a stream compaction (cf. section 2.2.2) first extracts all the indices of ridge edges from the mesh as well as the tuples of a ridge edge's two adjacent face groups. Each tuple (g_1, g_2) of two face group indices $g_1 \in \mathbb{N}$ and $g_2 \in \mathbb{N}$ is ordered internally so that $g_1 < g_2$. The algorithm keeps the exclusive prefix sum calculated during the stream compaction for later lookup purposes. Subsequently, the algorithm sorts the edge indices by their respective face group tuples. The parallel sort by key algorithm of the Thrust library [NVI23c] executes the sort by tuples, where the following relationship applies:

$$(g_1, g_2) < (g_1', g_2') \iff g_1 < g_1' \lor (g_1' \geq g_1 \land g_2 < g_2').$$

After the sorting algorithm terminated, a parallel pass over the sorted entries establishes an integer array that represents the permutation of sorted elements to invert the prior sorting operation. This enables to retrieve the original indices of edge indices before the sorting operation so that the exclusive prefix sum can be used again.

After the preparation, the first step of determining the feature edges is to calculate the number N_G of different face group tuples in the sorted list. This number can be obtained by incrementing a counter, whenever the sorted array of face group tuples exhibits different consecutive tuples. To calculate this counter value, the algorithm initializes an array with a zero entry for each ridge edge and a parallel pass over the sorted face group tuples sets the respective entry of a tuple to 1 if the consecutive face group tuple is different. An exclusive prefix sum calculates N_G. Subsequently, the feature edges identification allocates an integer array of the size N_G and fills it with the ridge edge indices where different consecutive face group tuples were found using the exclusive prefix sum results. As some ridge edges may be adjacent to more than two ridge edges, the next step checks in parallel if ridge edges include vertices that are endpoints of several feature edges. For each edge this parallel pass retrieves all the edges connected to the edge vertices and uses the prefix sum of ridge vertices and the inverse sorting order to obtain the face group tuples of adjacent ridge edges. Whenever adjacent edges exhibit the same face group tuple, a counter is incremented. If any of the adjacent ridge edges is associated with a different face group tuple, the ridge edge is marked to contain an endpoint of a feature edge. With the analog steps for calculating the upper bound of feature edges, the feature edge identification computes the upper bound of endpoints for feature edges and establishes an array of index positions of edges containing endpoints.

Using the pre-calculated endpoints, a sequential depth first search determines the feature edges as lists of consecutive edges separating the face groups. This depth first search iterates over the N_G sequences of ridge edges that are all associated with the same face group tuples. If a sequence of ridge edges is associated with a face group tuple (g_1, g_2) with $g_1 = g_2$, then this sequence of ridge edges does not separate two distinct face groups and is skipped. Otherwise, the depth first search extracts feature edges from the list of edges. The key strategy is to use the previously recorded endpoints to infer the number of feature edges associated with the sequence of ridge edges. If within a sequence of ridge edges there is only one single endpoint, the sequence of edges forms a loop. In this situation, the depth search writes the sequence of consecutive edges in a list that represents a feature edge. If a sequence of ridge vertices contains many (potentially duplicated endpoints) the depth first search writes consecutive ridge edges for each endpoint to a list representing the feature edges. As a result, the feature edges contain sequences of consecutive ridge edges that separate the face groups.

6.2. Volumetric Mesh Editing Operations

In order to allow for quick model customization using only the volumetric mesh, this section presents two volumetric mesh editing operations. These operations are volumetric hole closing and erosion. After each editing operation, a new numerical simulation can be automatically executed to immediately present the impact of the changes to the user. Both editing operations build upon the concept of face groups (cf. section 6.1) to enable convenient user interaction. In order to achieve interactive editing times, this section presents massively parallel algorithms for the most run time expensive steps. The algorithm fir closing co-planar holes appears in section 6.2.1 and section 6.2.2 presents the algorithm for erosion.

6.2.1. Volumetric Hole Filling

Mechanical parts often include holes, e.g., for mounting or weight relief, that users may want to remove in a modification step. With the use of face groups, the user can select the inner lateral surface of the cylindrical hole to-be-closed. As can be seen in fig. 6.3, the hole closing can then detect the boundaries of the face group to perform a meshing steps that fills the hole. Beyond closing the hole at both ends of the cylinder formed by the hole's inner surface, the hole in the tetrahedral mesh must be filled volumetrically. Therefore, the meshing of the hole extracts several PLCs (cf. section 2.1.2) and proceeds in a hierarchical order. First, the meshing closes the hole surface using 2D constrained Delaunay meshing. Thereafter, the a 3D meshing step fills the hole volumetrically using 3D constrained Delaunay meshing on the resulting PLC. To sketch the procedure, the hole closing algorithm performs the following four steps:

1. Search the triangular mesh of the face groups representing the hole' cylindrical lateral surface for boundary loops

2. For each loop: Close the loop performing planar constrained meshing using Shewchuck's Triangle library [She96]

3. Fill the resulting manifold by volumetrically meshing it using TetGen [Si20]

4. Merge the resulting tetrahedral mesh with the tetrahedral mesh of the model

The experiments have revealed that sequential execution on the CPU provides sufficiently fast run times for as quickly perceived responses. The number of triangles in the lateral surface of a cylindrical hole is typically small enough such that parallelization is not expected to achieve a significant impact on the perceived response time. In the case of a lateral surface mesh with many triangles, the tetrahedral meshing step (step three) may be worth parallelizing, but no GPU-parallel constrained tetrahedral meshing algorithms are available yet. Consequently, a sequential volumetric hole filling algorithm is used. Nonetheless, the use of GPU-accelerated boundary extraction (see section 4.2) leads to a substantial speedup.

As user-friendly maintenance of face groups requires reasonable assignment of newly added boundary faces to either persisting or new face groups, the hole filling algorithm checks if the newly added faces shall be assigned to a surrounding face group or not. Due to the hierarchical nature of our hole closing algorithm, which generates a surface mesh for each boundary loop individually, it is known which sets of newly added boundary faces are potentially added to an existing face group. The algorithm iterates through the boundary faces obtained by meshing each boundary loop and marks all the edges of the newly added surface triangles. Another loop iterates through all previously existing boundary triangles and checks, if a triangle contains any of the marked edges. If a triangle contains a marked edge, a list records the face group of the triangle. After this loop, if the list contains only one face group, we add the triangles closing the boundary loop to this face group. Otherwise, the newly added boundary triangles represent a new face group. Although parallelization

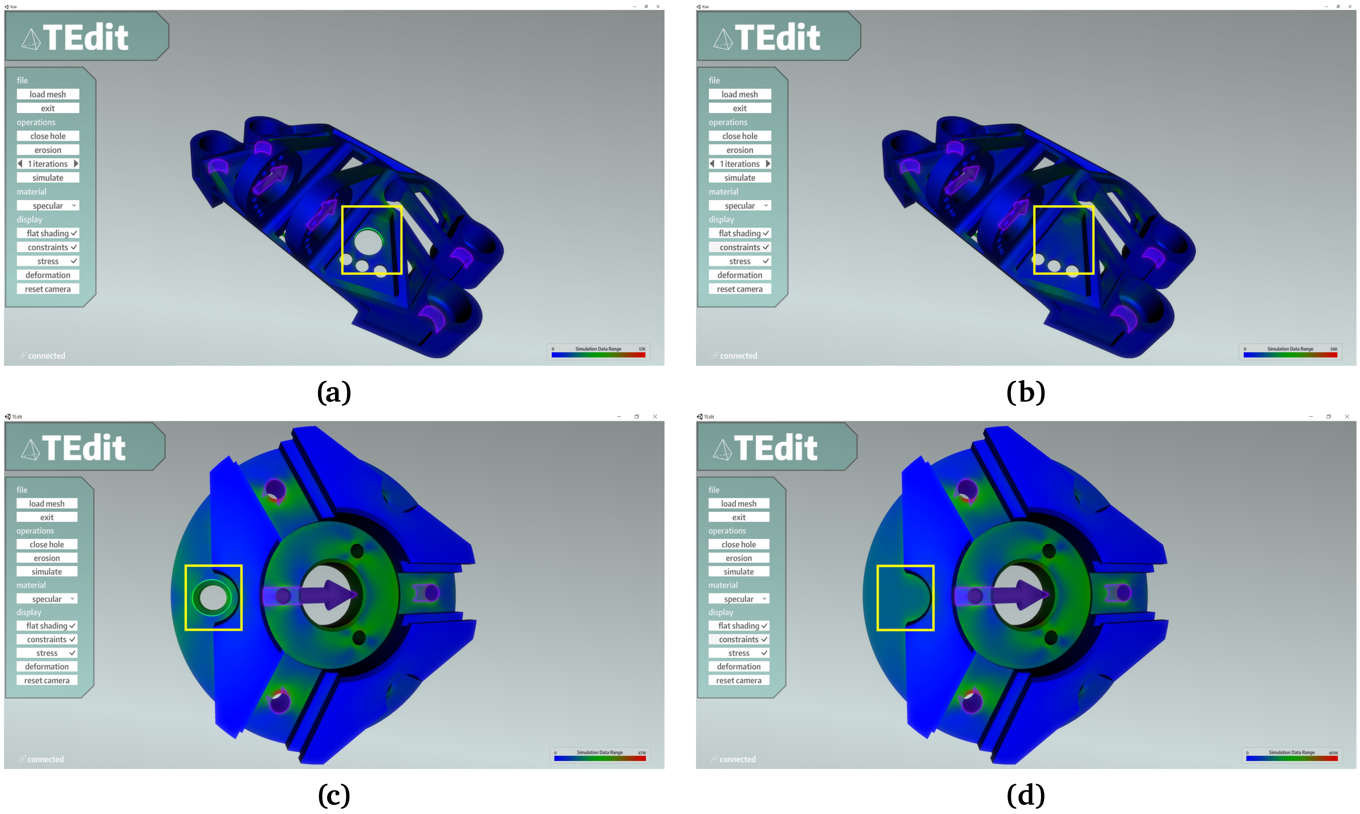


Figure 6.3.: Experiments for hole closing. The yellow rectangles indicate the area of the to-be-closed hole. Figure (a) shows the selected hole on the "SimJEB 633" model and Figure (b) shows the results. Figure (c) shows the selected hole of a mechanical part and (d) the resulting model.

of our face group assignment check algorithm is straightforward, the experiments did not reveal any situation in which sequential execution of face group assignment was the bottleneck.

6.2.2. Volumetric Mesh Erosion

The erosion operation can be used to remove parts of the model. See fig. 6.5 for examples of model customization through erosion. An overview on the steps of mesh erosion appears in fig. 6.4. The erosion method receives a set of boundary vertices belonging to the user selected surface triangles as an input. The implementation uses the TCSR mesh data structure (cf. section 2.2.3), as it enables massively parallel computation of connectivity relationships with efficient memory use. A marking identifies the remaining part of the mesh after erosion by marking tetrahedra or vertices as 0 or 1, denoted as *to-be-removed* and *remaining*, respectively. Initially, every tetrahedron of the mesh is unmarked. The first step of the erosion method is to perform a parallel pass over the input boundary vertices and mark every tetrahedron including any of the input vertices as *to-be-removed*. While users intend to thin down or remove a part of the model and expect erosion to return a consistent mesh, direct removal of every marked tetrahedron can leave "islands" of tetrahedra that are not connected to the rest of the mesh. In order to prevent this issue, a flood fill method is used to identify the to-be-removed parts of the mesh.

The flood fill propagates the *remaining* marking through the mesh starting from the unselected surface triangles performing parallel passes on the GPU. To initiate the propagation, a parallel pass over boundary triangles marks the tetrahedron associated with each boundary triangle. If the tetrahedron is not marked as

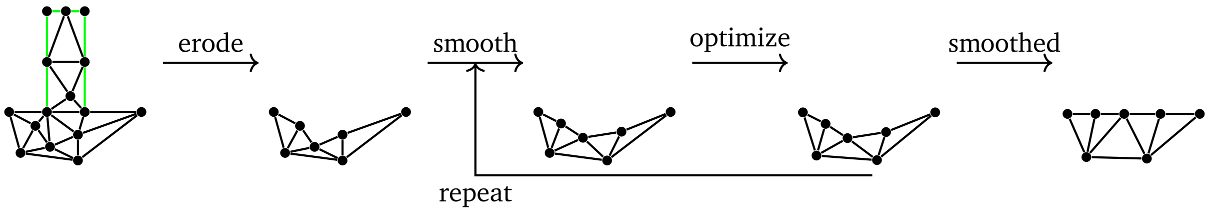


Figure 6.4.: Overview on the steps of the erosion method: First the user selects boundary triangles of the mesh (green). Then a part of the mesh is eroded. Alternating steps of smoothing and element quality optimization produce a smoothed surface.

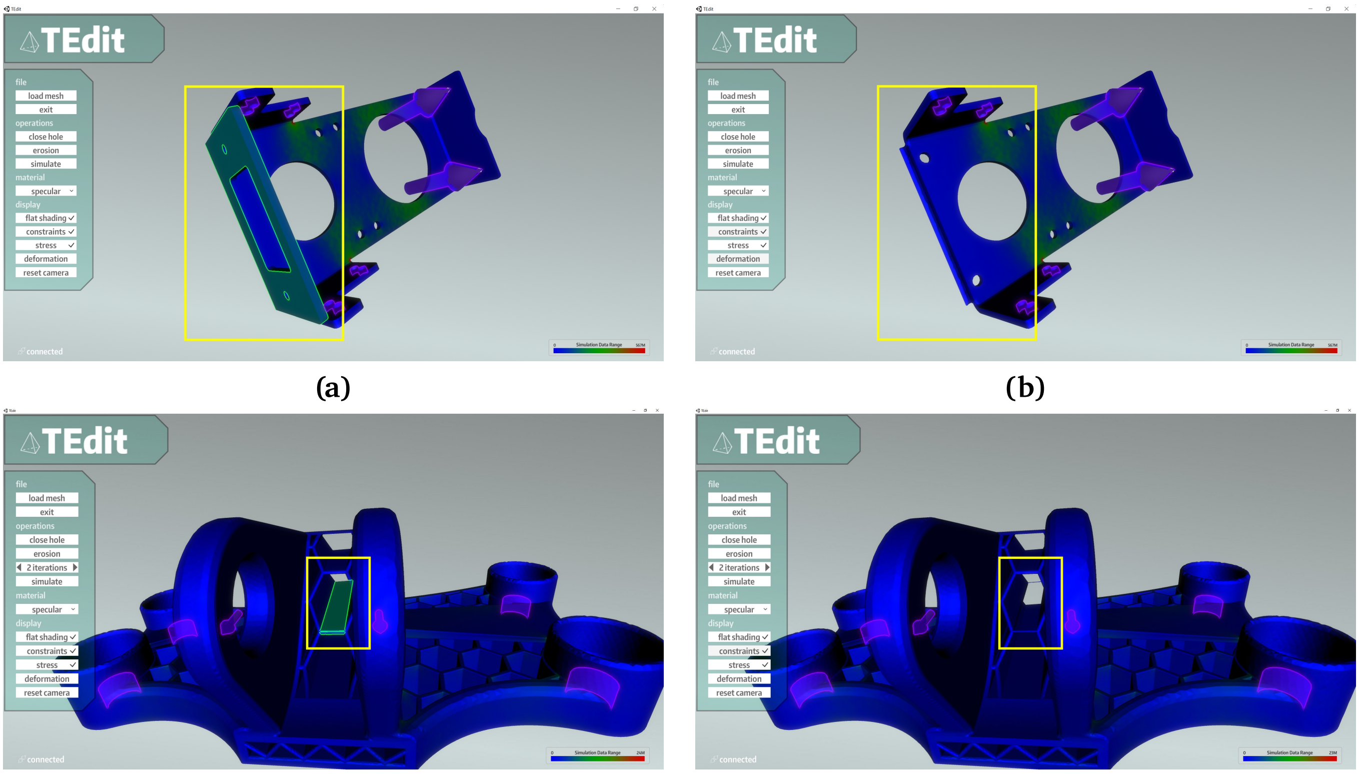
to-be-removed already, it is marked as *remaining*. Subsequently, a flood fill propagates the *remaining* marking by advancing into the interior of the mesh. To this end, the propagation checks the tetrahedra sharing an interior face with a currently unmarked tetrahedron in parallel on the GPU. If an unmarked tetrahedron is connected to a tetrahedron marked as *remaining*, the propagation marks it as *remaining* as well. The propagation performs several parallel passes advancing the remaining part until no further tetrahedron can be marked as *remaining*. Every tetrahedron without a marking is not connected to the mesh and is marked as *to-be-removed*. With the use of the resulting marking, the mesh erosion method writes the remaining tetrahedra and vertices to a buffer using Wald's [Wal21] GPU-parallel re-indexing method.

After the user-specified part of the mesh is removed, a coarse "zigzag" surface appears at the surface newly introduced due to erosion. However, users expect a smooth surface without protruding triangles. Therefore, the erosion not only removes parts of the mesh but also involves GPU-accelerated surface smoothing. The smoothing of the new surface uses the discrete Laplace-Beltrami operator [Nea+06], as it is efficient and provides a unique solution for a compact surface [SCV14]. As simultaneous update of every vertex is more efficient on the GPU than updating every vertex individually, the smoothing relocates all surface vertices simultaneously.

The smoothing pass first calculates the Laplace-Beltrami gradient for each surface vertex in parallel. Since the mesh editing algorithms aim to produce models suitable for the FEM, they must prevent element inversions. Similar to the vertex relocation in section 4.3.1, a binary search finds a step size λ such that relocating the vertices along their gradients does not produce any inverted tetrahedra. As soon as such a λ is determined, parallel Laplace-Beltrami smoothing relocates every vertex of the new surface. After a surface smoothing step, the element quality of boundary tetrahedra typically deteriorates. In order to improve the element quality of the tetrahedral mesh and enable successive smoothing passes without introducing inverted elements, an optimization step improves mesh quality after each surface smoothing step. For fast run times, the GPU-parallel mesh optimization algorithm presented in chapter 4 is used. For user control of workloads, smoothness and quality, the user is able to specify the number of alternating passes of smoothing and element quality optimization.

6.3. Evaluation of Tetrahedral Mesh Editing based on Face Groups

This section presents critical evaluation of the proposed methods to edit unstructured tetrahedral meshes on the basis of face groups. Section 6.3.1 evaluates the proposed face groups detection algorithm. An evaluation of the proposed hole closing algorithm appears in section 6.3.2. Section 6.3.3 evaluates the proposed hole erosion algorithm.



(a)

(b)

(c)

(d)

Figure 6.5.: Experiments for erosion. The yellow rectangles indicate the to-be-eroded parts of the model. Figure (a) shows the selected surfaces of a bracket and (b) shows the resulting model. Figure (c) shows the selected strut of the "SimJEB 220" model and (d) shows the results.

6.3.1. Evaluation of Face Group Detection

Throughout the research for this PhD thesis, many students and colleagues applied the face group detection algorithm to a multitude of meshes. The evaluation discusses the observations of the evaluation on 12 unstructured tetrahedral meshes with different geometric features. These meshes and the resulting face groups (separated by color) can be seen in fig. 6.6. The specifics about these meshes, input setups such as γ_r, and the run times appear in table 6.1.

With setting a suitable ridge angle γ_r, the face group detection is able to organize many boundary triangles into face groups. For meshes with surface features separated by ridges, the proposed detection algorithm yields meaningful face groups that can be associated with semantic features such as a drill hole. The algorithm's applicability is not limited to meshes with only flat boundary features, e.g., Fan guard and Shutter, but can also be used for meshes approximating smooth curves, e.g., Vase and Bishop. Thus, the user only needs to find a suitable ridge angle γ_r, which can admittedly require several executions, if the user is not aware of the boundary curvature specifics of the model at hand. This also means that smooth curves should be approximated by a reasonable number of boundary triangles for achieving practical results with the proposed face group detection algorithm. Since the entire partitioning of the surface into face groups depends on one homogeneous ridge angle γ_r, the face group detection is not adaptive to the local surface curvature. Therefore, it might happen that the intended face groups cannot be determined with a global ridge angle. Nonetheless, the user can re-run the face group detection, whenever the current face groups do not allow for specifying the current editing operation in mind. In addition, the user can select several face groups and apply the editing operation to the selected groups, as if these groups were one single face group.

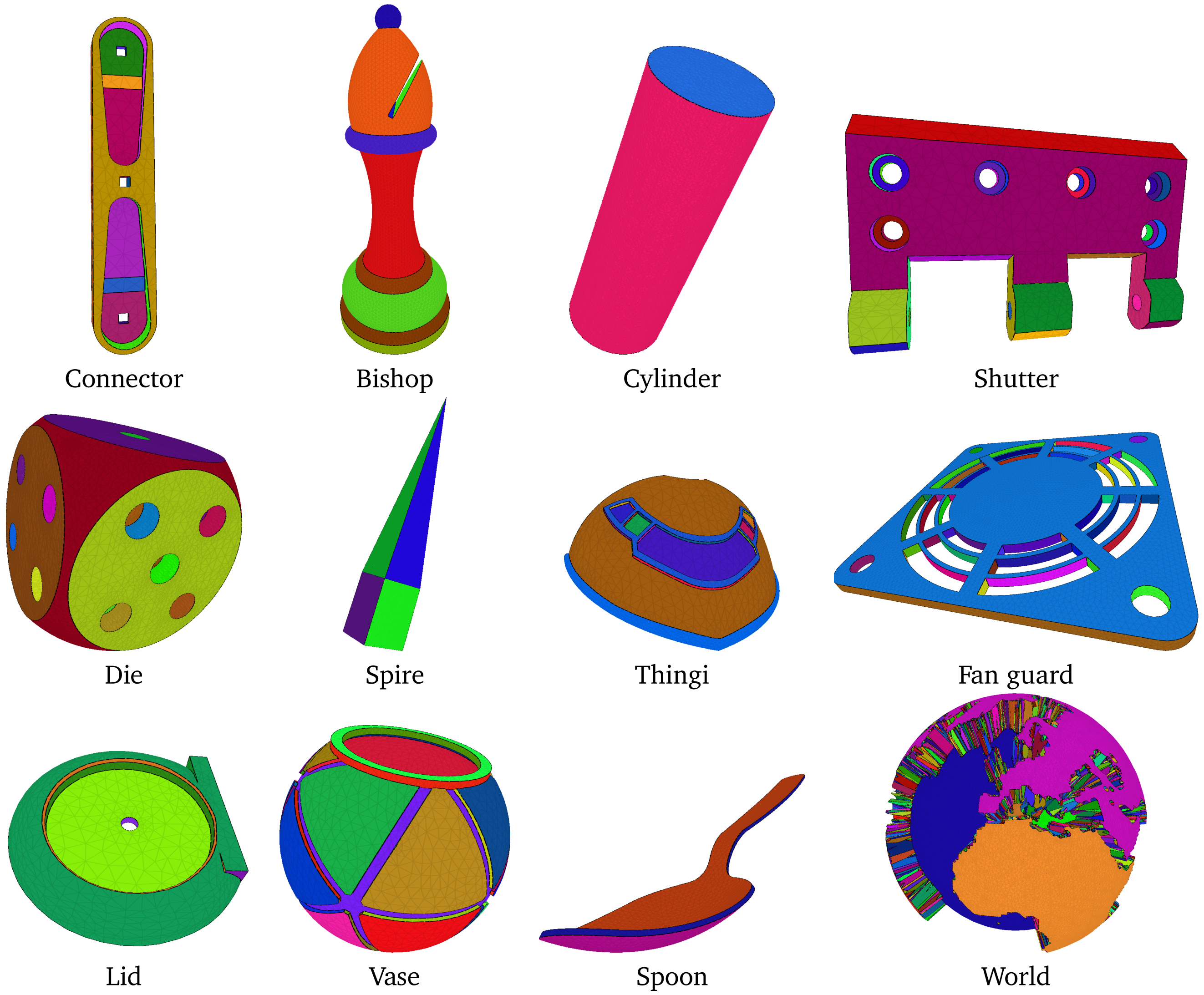


Figure 6.6.: For each tetrahedral mesh boundary, the resulting face groups are separated by colors.

As the face group detection is supposed to be used for interactive editing, the efficiency of the algorithm is an important property. In order to evaluate the efficiency, the evaluation includes several run time measurements for unstructured tetrahedral meshes with mesh sizes ranging from 2 k to more than 1 M of elements. As can be seen in table 6.1, the face group detection only requires a few milliseconds for all the evaluated models, which is acceptable for a user-configured pre-process. For most models, the run times do not admit a frame rate of 30 frames per second. Thus, the face group detection does not achieve real-time performance for large models. However, in the context of a user-configured process, it is sufficient to enable presentation of results in below one second [New94], though real-time performance of face group detection can be interesting for dynamically varying geometry in a simulation. High-resolution models tend to demand longer run times, though it is not generally true that a tetrahedral mesh with more elements leads to slower run time performance. As the run time of the connectivity lookup of the TCSR mesh data structure (cf. section 2.2.3) scales with the complexity of the mesh, it is reasonable that meshes with more elements tend to require longer run times. Another factor that influence the run time performance, is the number of boundary triangles in a tetrahedral mesh. The more boundary triangles the mesh includes, the more boundary facets need to be extracted and classified by the

Table 6.1.: Run times of face group detection on an NVIDIA RTX 3090 GPU for different tetrahedral meshes ordered by N_T.

	Input				**Run time** (ms)			
Name	N_T	N_V	$N_{\partial\mathrm{T}}$	γ_r	Total	Boundary extraction	Face groups detection	Feature edges detection
Connector	2086	707	1286	15°	37	25	8	3
Lid	4332	1346	2244	20°	37	25	5	7
Spoon	7849	2765	5520	20°	30	27	2	1
Thingi	15 649	3939	4692	15°	34	23	8	3
Shutter	17 974	4782	6306	25°	39	27	7	5
Vase	20 865	6771	12 860	25°	45	28	12	4
Fan guard	82 902	19 674	24 440	20°	58	28	19	11
Cylinder	172 441	34 145	25 976	15°	39	30	6	3
Bishop	177 744	32 336	16 310	25°	43	30	10	3
Die	232 767	44 676	30 136	15°	37	32	11	4
Spire	247 476	47 065	31 924	2°	57	31	17	9
World	1 786 620	367 573	335 564	17°	133	63	44	26

boundary extraction procedure. Therefore, the run time of boundary extraction increases with the number of tetrahedral elements N_T and the number of boundary triangles $N_{\partial\mathrm{T}}$. For many meshes, the boundary extraction is the step with the largest portion of the total run time for face group detection.

The detection of face groups and feature edges benefit from the pre-calculated connectivity lookups in the boundary extraction step. Therefore, these two steps typically exhibit faster run times than the boundary extraction. As these steps work on the mesh surface, the number of boundary triangles is an important factor for the run time performance. However, the workload for extracting face groups and feature edges also depends on the complexity and number of surface features. For instance, the surface of the Fan guard model consist of many face groups, which leads to significantly longer run times for extracting face groups and feature edges compared to simpler models such as the Cylinder. Additionally, face groups that include many boundary triangles require more propagation steps for face group detection, which increases run times. As the extraction of feature edges needs to perform sequential depth first search, the number of separate face groups significantly influences the run time of this step, as can be seen for meshes with a very large number of face groups such as the World. However, the extraction of feature edges is typically the cheapest step, because most edges on the mesh surface are filtered out by the classification steps before.

6.3.2. Run Time Performance and Element Quality for Hole Closing

In order to critically review the capabilities of the hole closing operation in VP, the evaluation covers run time performance and element qualities of resulting meshes. As it is typically most promising to accelerate the bottleneck of a process, the evaluation of the hole closing operation begins with an analysis of bottlenecks. For this purpose, an evaluation of the relative run times for each step of the sequential hole closing operation reveals overheads. The sequential hole closing operation can exhibit slow run time performance, if the boundary extraction runs sequentially on the CPU. This is especially an issue, if boundary triangles shall be mapped to face tags originating from CAD, because the boundary needs to be extracted and mapped, whenever the shape of the volumetric mesh changes. For instance, compare the relative run times shown in fig. 6.7 for purely sequential hole closing operation shown in fig. 6.3 (a) and (b).

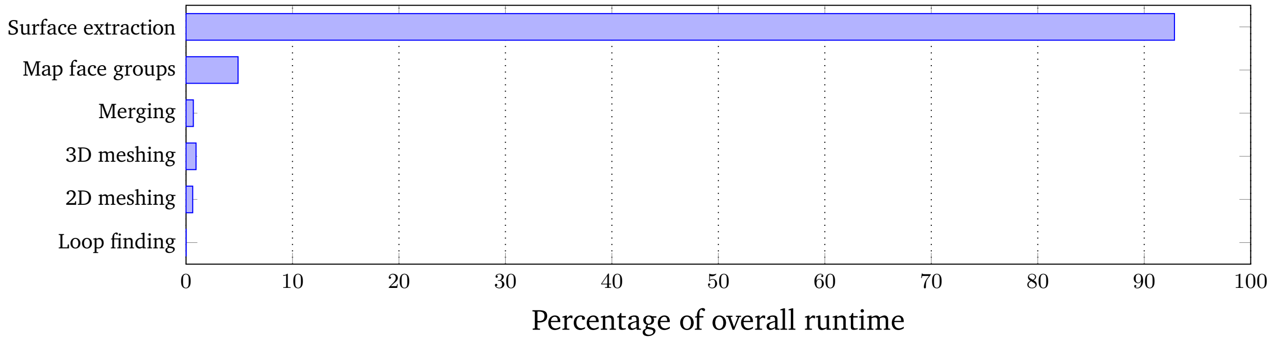


Figure 6.7.: Relative run times of the individual steps of the hole closing operation in fig. 6.3 (a) and (b). The diagram shows that surface mesh extraction is the bottleneck.

Therefore, fast boundary extraction is a necessity for achieving fast editing of volumetric meshes. If the face groups are loaded from face tags originating from CAD, then it is possible to quickly extract the boundary faces with the algorithm from section 4.2. For the examples in fig. 6.3, the hole close operation can be accelerated significantly (see fig. 6.8). If the face groups are detected using the algorithm in section 6.1.2, then the massively parallel boundary extraction from section 4.2 is applied as a part of the face group determination.

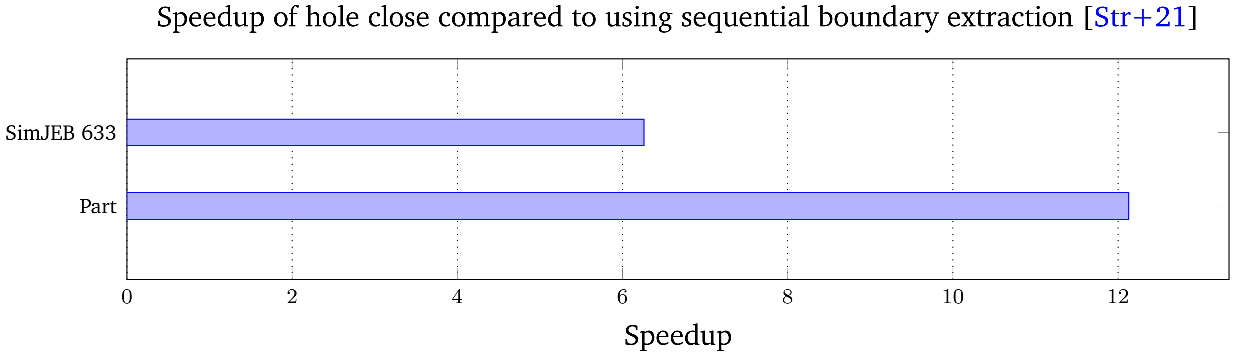
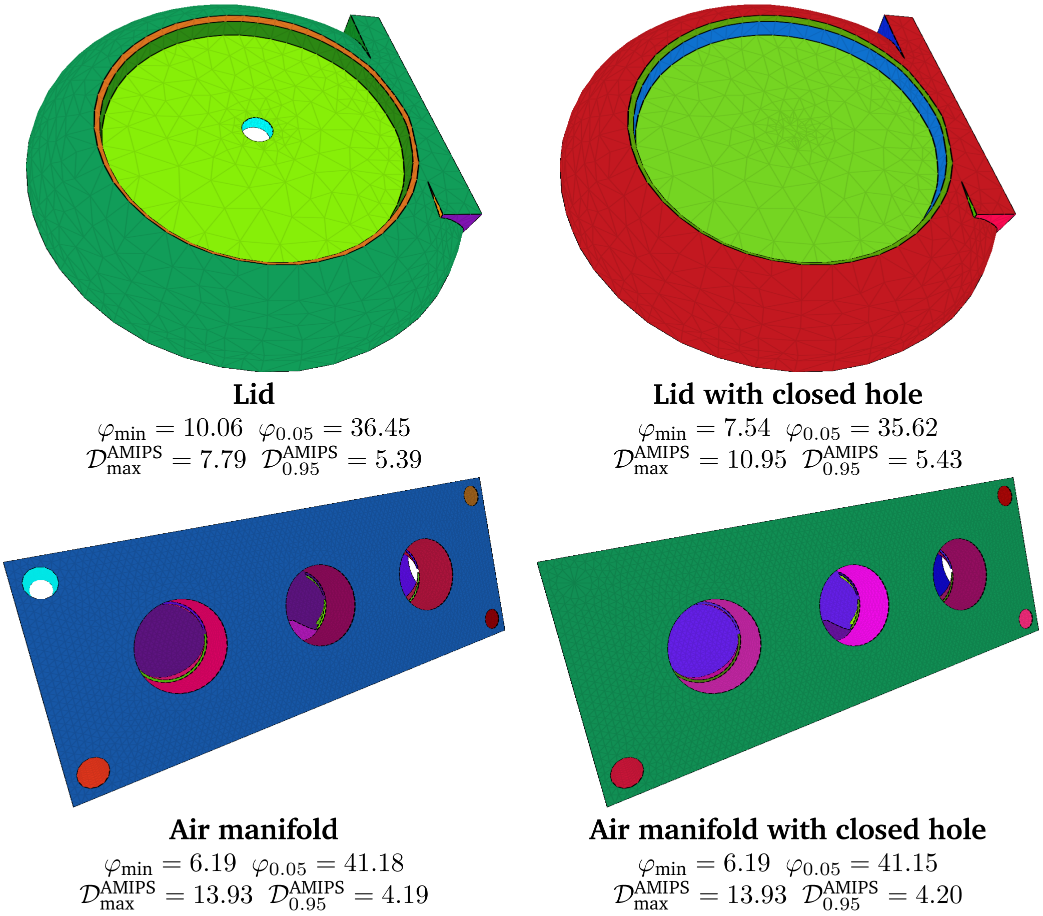


Figure 6.8.: This figure shows the speedups due to the application of massively parallel boundary extraction compared to the hole closing algorithm executed sequentially on the CPU [Str+21]. The evaluation machine is equipped with an Intel i7-3930K CPU and an NVIDIA RTX 3090 GPU. The models SimJEB 633 and Part appear in fig. 6.3 (a) and (c), respectively.

As the edited meshes should be of sufficient element quality for numerical simulation, the evaluation involves a comparison of tetrahedral element qualities before and after hole close. Two examples are the Lid and the Air manifold that appear in fig. 6.9. In order to evaluate if the element quality is suitable, the evaluation determines the worst element quality in the mesh and the quantiles for the worst 5% of elements. Element quality is provided in terms of the dihedral angle φ and the scale-invariant AMIPS conformal energy $\mathcal{D}^{\text{AMIPS}}$ (cf. section 3.2.2). Typically, the resulting mesh after closing a coplanar hole is of sufficient element quality for numerical simulation. The hole closing operation adds new tetrahedra to the mesh that fill the lateral surface of the selected hole with a planar boundary. As an enclosing PLC is ensured by prior 2D meshing, a clean volumetric closure of the hole is produced. The volumetric Delaunay meshing step (cf. section 2.1.2)



Lid

$\varphi_{\min} = 10.06 \;\; \varphi_{0.05} = 36.45$
$\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 7.79 \;\; \mathcal{D}^{\mathrm{AMIPS}}_{0.95} = 5.39$

Lid with closed hole

$\varphi_{\min} = 7.54 \;\; \varphi_{0.05} = 35.62$
$\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 10.95 \;\; \mathcal{D}^{\mathrm{AMIPS}}_{0.95} = 5.43$

Air manifold

$\varphi_{\min} = 6.19 \;\; \varphi_{0.05} = 41.18$
$\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 13.93 \;\; \mathcal{D}^{\mathrm{AMIPS}}_{0.95} = 4.19$

Air manifold with closed hole

$\varphi_{\min} = 6.19 \;\; \varphi_{0.05} = 41.15$
$\mathcal{D}^{\mathrm{AMIPS}}_{\max} = 13.93 \;\; \mathcal{D}^{\mathrm{AMIPS}}_{0.95} = 4.20$

Figure 6.9.: Resulting boundaries and tetrahedral element qualities for the hole close operation applied to the Lid (top) and the Air manifold models.

can then be executed on the resulting PLC to ensure the absence of inverted elements. As Tetgen enables to preserve the input PLC triangles, the produced tetrahedralization's boundary triangles are matching to the lateral surface of the cylindrical hole, which provides a valid tetrahedral mesh. However, the meshing step can add elements of lower quality than the input elements to the mesh, which can be seen for example with the Lid mesh. Thus, the hole close operation can potentially degrade element quality. Typically, the degradation of element quality does not interfere the numerical simulation, because the meshing tools optimize for element quality. Otherwise, one can perform quick optimization, e.g., using the massively parallel algorithms in chapter 4 or section 5.2.

As an additional evaluation result, the run time performance of the hole close operation typically enables to produce the edited mesh in a couple of milliseconds. The run times and mesh sizes for the experiments in fig. 6.9 appear in table 6.2. For small meshes such as the Lid, the time for meshing the hole part is oftentimes lower than the time for extracting the boundary and detecting new face groups. For larger meshes such as the Air manifold, the time for meshing is predominantly governed by the number of triangles N_{lateral}. The volume of the hole, i.e. the target resolution of the tetrahedral elements, is also a factor that governs the run time of meshing. Since the hole close operation is a quick modification of a mock up, the meshers are not configured to additionally refine the mesh of the lateral surface after creating a tetrahedralization of good quality. Additional refinement can complicate the merging step with the input mesh. If the simulation accuracy demands a finer resolution for more precision, mesh adaptation tools such as the tool discussed in section 5.3 can be applied after hole closing. The run time for re-detecting the face groups after the hole close operations is oftentimes longer than the time for hole meshing, when it comes to high-resolution meshes such as the Air manifold. This is an expected observation, because TCSR mesh needs to re-compute connectivity information and the the face group detection extracts and classifies the newly created surface.

	Input			**Run time** (ms)		
Name	N_T	N_V	N_lateral	Total	Hole close	Face groups determination
Lid	4332	1346	270	36	22	14
Air manifold	495 850	93 830	3132	292	74	218

Table 6.2.: Run times on an NVIDIA RTX 3090 GPU and an Intel i9-11900K CPU for hole close operations shown in fig. 6.9.

6.3.3. Run Time Performance and Element Quality for Erosion

In order to evaluate the applicability of the erosion algorithm in section 6.2.2, the evaluation investigates its run time performance and the element quality of the resulting meshes. As the use of a massively parallel algorithm is supposed to accelerate the erosion operation, the evaluation compares the sequential erosion variant from the paper introducing TEdit [Str+21] to the massively parallel variant [Str+23]. Figure 6.10 plots the speedup for the Bracket and SimJEB 220 models shown in fig. 6.5. The evaluation reveals significant speedups of 4.55× and 5.52×. Thus, the use of massively parallel flood fill, smoothing and mesh optimization (see chapter 4) leads to significantly faster run time performance.

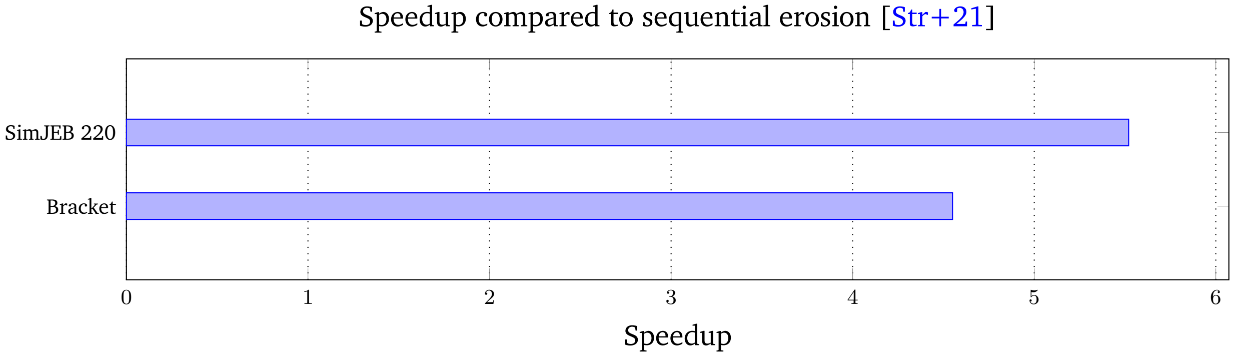


Figure 6.10.: This figure shows the speedups due to the application of massively parallel boundary extraction compared to the erosion algorithm executed sequentially on the CPU [Str+21]. The evaluation machine is equipped with an Intel i7-3930K CPU and an NVIDIA RTX 3090 GPU. The models Bracket and SimJEB 220 appear in fig. 6.5 (a) and (c), respectively.

As a typical example, the evaluation shows the run time and element qualities for the erosion of the bracket in fig. 6.5 (a) and (b). Since the number of iterations for surface smoothing and element quality optimization has a substantial impact on the results, the evaluation presents the results for performing none to six iterations. Table 6.3 presents the medians of the measured run times and the resulting element qualities.

Iterations	Element quality $\varphi_{\min}$	$\varphi_{0.05}$	$\mathcal{D}^{\text{AMIPS}}_{\max}$	$\mathcal{D}^{\text{AMIPS}}_{0.95}$	Run time (ms) Total	Connectivity lookup	Generate new mesh	Smooth & optimize
None	11.21	37.41	8.34	4.68	175	92	83	0
1	13.81	41.43	8.17	4.11	282	92	83	107
2	8.94	41.43	11.15	4.12	332	92	83	157
3	5.61	41.42	18.49	4.12	378	92	83	203
4	3.78	41.43	28.68	4.12	422	92	83	247
5	2.68	41.41	35.86	4.13	484	92	83	309
6	1.51	41.40	58.51	4.13	511	92	83	336

Table 6.3.: Element qualities and run times for varying iteration counts of smoothing and quality optimization for eroding the bracket in fig. 6.5 (a) and (b). The tetrahedral mesh of the bracket consists of $N_{\text{T}} = 108\text{k}$ tetrahedra and $N_{\text{V}} = 31\text{k}$ vertices. For run time measurements, the median run times of several executions have been determined. The evaluation machine is equipped with an NVIDIA RTX 3090 GPU and an Intel i9-11900K CPU.

Eroding the selected parts without smoothing the resulting surface exhibits a fast run time performance but leaves a coarse surface. Performing one smoothing iteration already increases the run time significantly. The mesh quality is significantly improved, because the mesh optimization algorithm in chapter 4 improves the quality of every element in the mesh and the smoothing, which counteracts the degradation of shape quality due to smoothing. After performing a second iteration of smoothing and element quality optimization, the element quality is worse than the input, while the element quality still is sufficient for numerical simulations. The performance of the second iteration exhibits only half the run time compared to the first iteration. This observation is expected, because smoothing only changes a small subset of vertices, while the remaining vertices still reside in a local minimum. Thus, the second optimization iteration exhibits faster convergence. For each additional smoothing iteration, the mesh quality further degrades and the run time further increases. Until after six iterations, the mesh quality has substantially degraded but still allows for numerical simulation with the simulation algorithm of use, as our experiments revealed. The performance for erosion with six iterations of smoothing and element quality optimization exhibits a run time of 551ms, which is still a sufficient time for interactive editing. Thus, the erosion scheme exhibits sufficient results for use cases like presented in fig. 6.5.

6.4. Evaluating the Capabilities of Cage-based Deformation

The previous section 6.3 investigated the editing of face group-based mesh editing, which is intended for mechanical parts. The surface of a mechanical part frequently includes sharp features. The face group detection from section 6.1.2 can take advantage of the sharp features to obtain a meaningful separation of the mesh. However, important VP applications such as 3D printing [Bad+16; Pop+20] or shape optimization [PTA20] involve geometries with smooth and organic-like surfaces. As these geometries oftentimes include semantic features that are not separated by sharp changes in curvature, which complicates the usage of face groups for mesh editing. Editing geometries with smooth and organic-like surfaces oftentimes requires the deformation of the shape with a smooth deformation approach. Therefore, this chapter evaluates the capabilities of cage-based deformation (cf. section 2.3), in order to investigate a mesh editing approach for smooth and organic-like geometries.

Many advances in the past decade have improved the capabilities of cage-based deformation. A survey paper [Str+24] about these advances has been published and includes many of the following evaluations.

In order to evaluate the advances of cage coordinate types, the CageModeler tool set implements the most relevant coordinate types and provides an interactive application for cage-based deformation. In addition, CageModeler allows for cage-based deformation of unstructured tetrahedral meshes to support the shortening of VP cycles. In the light of VP, especially important are the advances in cage generation (see section 6.4.1), local deformation control (see section 6.4.2), freedom of cage connectivity (see section 6.4.3), and preservation of volume as well as shape (see section 6.4.4). Since cage-based deformation enables the use of massively parallel GPUs for calculating the deformed geometry (see section 6.4.5), it provides efficient mesh editing. For VP applications, the use of cage-based deformation can extend the capabilities of mesh editing in two ways:

- deform an unstructured tetrahedral mesh so that the development team saves the overhead of changing CAD geometries and re-meshing, and
- obtain smooth deformation of organic-like shapes, which can be useful for artistic designs or shape optimization applications.

As the cage-based deformation of unstructured tetrahedral meshes for VP purposes depends on mesh quality, the evaluation involves an investigation of tetrahedral element quality in section 6.4.6. Finally, section 6.4.7 presents a comparison of the overall coordinate types.

6.4.1. Comparing Cage Generation Methods

The evaluation discusses the methods for cage generation and presents a systematic comparison in table 6.4. The first automatic cage generation methods fall into the offset surface simplification category. Progressively collapsing edges is inherently a sequential procedure, which imposes low run time performance for cage generation. Self-intersections are either resolved within the local topological vicinity or globally across $\mathcal{C}$. The avoidance of global self-intersections or intersections with $\mathcal{M}$ requires intersection tests for all cage faces [DLM11] or the application of sophisticated offset surface handling [SVJ15], which significantly reduces run time performance. Thus, these methods are not suitable for applications, where users expect a cage immediately after loading a high-resolution model. Especially for high-resolution models, situations may occur, where the simplification scheme is not able to decimate the fine-grained details of the offset surface leading to robustness issues [CB17]. Nonetheless, offset surface simplification methods are successful in achieving a low number of control vertices leading to improved run time performance at bind time. Consequently, cage simplification is an important optimization step in many current cage generation methods.

The voxelization-based approaches offer simple and efficiently parallelizable methods for cage generation. Self-intersections and intersections with $\mathcal{M}$ can be avoided easily, as the cage is formed from non-intersecting voxel faces respecting an offset distance to $\mathcal{M}$. Without post-optimization the resulting cages are even globally free of self-intersections. Users can control the resolution of the cage by specifying the voxel size. As it is not intuitively predictable which voxel size leads to the intended results, users typically need to run the cage generation method multiple times to obtain a suitable cage. Additionally, the majority of the fully automatic methods only allow for a homogeneous voxel size, whereas some details might need finer sampling. Thus, voxelization-based methods excel at quick and robust cage-generation but lack intuitive control.

The template-based methods are intended specifically for reusing skinning configurations across several animated characters. The key advantage of template-based methods is that they assist users in posing the model, because users only need to perform minor posing for the motion details not covered by the template. Template-based methods come with the drawback that users require a large library to find similar characters with suitable templates. Moreover, template-based methods inherently depend on a skeleton for cage generation. Cage generation constructs an offset surface along the skeleton curve, which is prone to self-intersections, unless the methods of Yang et al. [Yan+12] are applied. For the typical scenarios of VP, the

Table 6.4.: Comparison of cage generation methods ordered by category (top to bottom: offset surfaces simplification, voxelization-based, template-based, and interactive) and year of publication. Being conservative expresses the ability to generate $\mathcal{C}$ not intersecting with $\mathcal{M}$.

method	no self-intersections	conservative	tightness	symmetry	comment
[San+00]	Local	✘	✘	✘	Basic building block for cage generation
[SOS04]	Resolution-dependent	Resolution-dependent	✔	✘	Relies on iso-surfaces
[BWG09]	Local	✔	✔	✘	Intended for deformation transfer
[DLM11]	Global	Vertices of $\mathcal{C}$ only	✔	✘	Features post-hoc cage repair
[XZG13]	Local	Vertices of $\mathcal{C}$ only	✔	✘	Semantic mapping
[SVJ15]	Global if the input has no self-intersections	✔	✔	✘	Generates a sequence of nested cages
[XLG09]	Global w/o smoothing local with smoothing	Vertices of $\mathcal{C}$ only	✔	✘	Smooth and tightly fitting cages
[Nes+09]	Global	✔	✘	✘	Intended for linear elasticity deformable objects
[XLG11]	Global w/o improvement local with improvement	✔	✔	✘	Quick generation of coarse cages for complex models
[XLX15]	Global w/o collapsing local with collapsing	✔	✔	✘	Provides a high-quality cage for a suitable voxel size
[Ju+08]	Local	✔	✔	✘	Usability depends on a large template library
[Yan+12]	Global if input is of genus 0	✔	✔	✘	Intended for muscle design
[CF14]	Local	✔	✔	✘	Generates a sequence of cages from skeletons
[LD17]	Local due to using unbounded cut slides	✔	✔	✔	Expressive interaction through user-defined cuts
[CB17]	Global	✔	✔	✔	Highly efficient and robust
[Cas+19]	Global for a sufficient number of bending nodes	✔	✔	✔	Generates a cage for a skeleton by bending node control

template-based approaches are not useful, because skeletons are not a common data structure for engineering design. Thus, engineers would have to create a suitable skeleton in the first place, which is an undesirable overhead. An exception is, when the prototype is effectively a character model that resembles typical characters for animation applications. In such a case, template libraries like the one in Adobe's Mixamo [Ado24] can be used.

Since the cages generated by automated methods typically need manual adjustment to be suitable for the intended deformation, the interactive cage generation methods are most useful. Especially for VP use cases, the engineers can specify the semantic parts of a model that they intend to edit. This provides them cages adapted to the intended deformation task and reduces the overhead of adjustment of cages. While the fully automatic methods typically do not guarantee to provide symmetrically structured cages for symmetric features, many interactive methods provide symmetric cages. The methods by Chen and Feng [CF14] and Casti et al. [Cas+19] additionally depend on a skeleton, which renders them inapt for most VP scenarios. Among the interactive methods, Le and Deng [LD17] and Casti et al. [Cas+19] provide the most fine-grained user control of the cage generation. As Calderon and Boubekeur [CB17] offer a heterogeneous voxel size that can be interactively controlled, their method benefits from the efficiency and robustness of parallel voxelization.

6.4.2. Locality of Coordinate Types

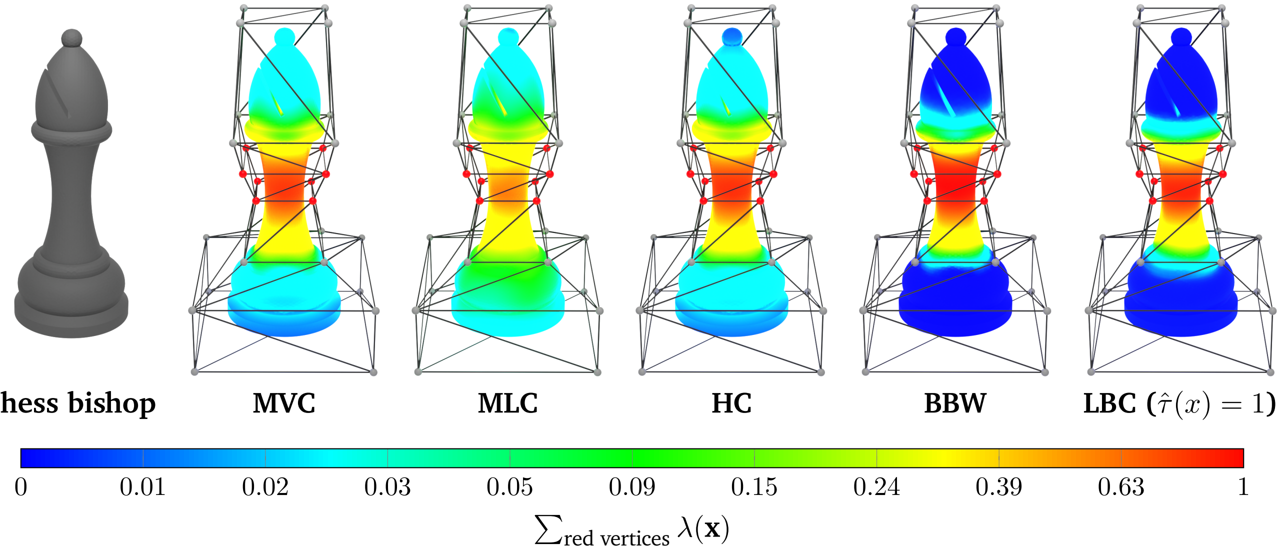

chess bishop
MVC
MLC
HC
BBW
LBC ($\hat{\tau}(x) = 1$)
0
0.01
0.02
0.03
0.05
0.09
0.15
0.24
0.39
0.63
1
$\sum_{\text{red vertices}} \lambda(\mathbf{x})$

Figure 6.11.: Visualization of the local influence of red control vertices for different coordinate types with a logarithmically scaled color map.

The use of cage-based deformation in VP requires protecting the global shape of the model when deforming specific local features of a prototype. In addition, the influence of each cage vertex should be local for intuitive deformation control. Thus, the evaluation of coordinate types includes the locality, i.e., the local influence, of cage vertices. For this evaluation, the CageModeler tool set enables to plot the local influence of a set of cage vertices. Prototypical for many experiments, fig. 6.11 plots the local influence of various GBC types for a selected set of control vertices. Generally, all the coordinate types for cage-based deformation provide local deformation but the degree of the locality highly varies across coordinate types.

MVC provide high influence near the control vertices, while the GBC functions decay only slowly in the more distant regions. The probability-based coordinates such as MLC are less local and decay slower than MVC. EMC (cf. section 3.6.2) exhibit the most local deformation control. HC are more local than MVC. A significant increase of locality is achieved using BBW for cage-based deformation. The use of LBC (with $\hat{\tau}(x) = 1$) inherently provides the most local set of GBC. Our experiments confirm that even more local GBC are obtained by setting $\hat{\tau}$ to a monotonically increasing function. The locality of coordinates with normal control is more difficult to plot, because face normals and stretch factors also influence the locality of deformation. Typically, these coordinates provide less locality than the other coordinate types.

6.4.3. Cage Connectivity Support of Coordinate Types

As quad-layouts are more common in industry, many applications use quads for cages. Due to the frequent use of VP for industrial applications, the support of quad-layouts for shape deformation is important in VP processes. A cage needs to be triangulated, if the coordinate type only supports triangles. However, deforming a model with a triangulated cage can lead to artifacts (see fig. 6.12). Instead of triangulating cage polygons, one should use suitable coordinates to avoid artifacts in the deformed geometry.

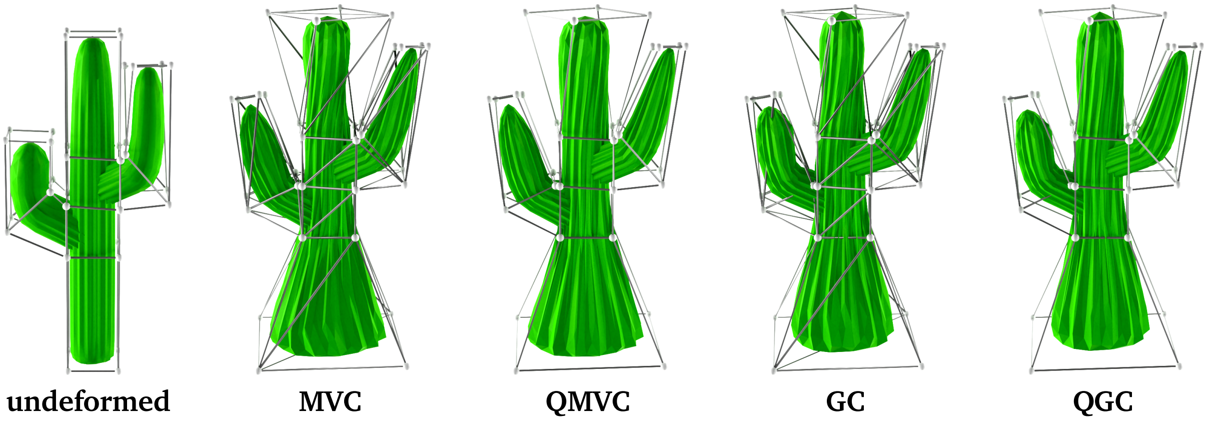


Figure 6.12.: Cage triangulation can lead to artifacts. The use of tri-quad enabled coordinates such as QGC resolves this issue.

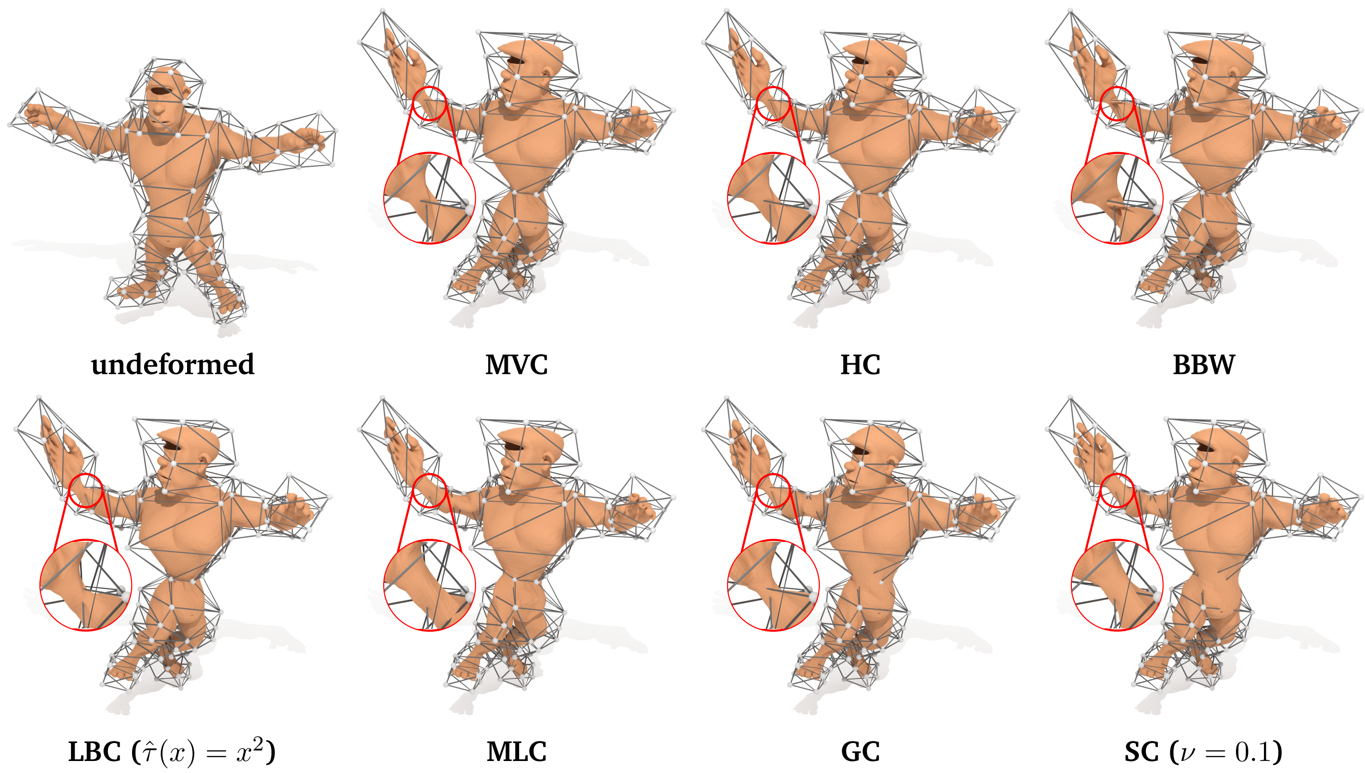
The probability-based coordinate types conceptionally support pointwise evaluation for arbitrary polygon cages, while MLC has only been evaluated for triangle cages up to now. Local deformation for planar polygon cages can be obtained with the use of HC, though the numerical solver of use needs to provide an implementation for the polygon type of the cage. Considering that the most commonly used solvers for HC are implemented for triangular cages, users cannot easily benefit from the support of arbitrary planar polygons. Users can easily deform models with quad or tri-quad cages using QMVC or QGC and benefit from simple and efficient pointwise evaluation. An advantage of QGC extrapolation capabilities, which enable users to only wrap the model parts of interest with cages while each cage still deforms the exterior parts.

6.4.4. Shape Preservation of Coordinate Types

As users typically wish to preserve surface features, cage-based deformation should avoid significant distortion of surface features. However, large deformation can lead to unintended shape distortion. Especially in VP processes the preservation of volume and shape under large deformation is important to avoid unintended artifacts in the geometry of the prototype. Thus, the evaluation discusses the ability of different coordinate types to preserve the input shape under large deformation. Figure 6.13 presents the resulting geometries by substantially deforming an Ogre model.

The application of simple MVC to large deformation can impose sharp artifacts, because MVC is prone to inflating the volume more at the parts near the cage vertices. This can lead to asymmetric shapes such as the chest of the Ogre and sharp bumps such as the distortion at the Ogre's wrist and shoulder. Since the locality of MVC is governed by the euclidean distance between a point $\mathbf{x} \in \Omega$ and a cage vertex, the occurrence of these artifacts is difficult to avoid. Due to their increased locality, the EMC suffer from the same problem. Significant artifacts are observed with the use of BBW, whereas one can easily preserve certain areas by incorporating an additional energy term, e.g., preserving the Ogre's right wrist. Among the EMC, the application of LBC seems to most reliably prevent artifacts, because their calculation effectively minimizes total variation and the locality is controlled by geodesic distances.

Deformation using MLC provides good shape awareness in the interior of the cage, which results in good volume preservation. For instance, see the Ogre's right arm after deformation with MLC. However, the probability-based GBC also incur artifacts especially at the regions close to the cage boundary. The best feature and volume preservation is provided by the cage coordinates with normal control, because they deform the model considering the shape and scaling of cage faces besides the cage vertex positions. As a result, coordinates with normal control better avoid the introduction of asymmetries and sharp features (see the Ogre's chest and shoulder). While GC provide good feature preservation in general, improved control of



undeformed **MVC** **HC** **BBW**

LBC ($\hat{\tau}(x) = x^2$) **MLC** **GC** **SC** ($\nu = 0.1$)

Figure 6.13.: Comparison of the results of large deformation of an Ogre model using different coordinate types.

volume and shape preservation can be achieved using SC. Cage deformation using SC can restrict the volume inflation of the model (see the Ogre's right hand). The only drawback of coordinates with normal control is that after large deformation the cage potentially intersects the model. In such a situation, the influence of the cage on the model is not as intuitive and the user needs to gauge which cage vertices to relocate to achieve the intended deformation.

6.4.5. Computational Cost for Coordinate Computation

The coordinate types for cage-based deformation differ in their computational overheads at bind time. The EMC impose the most pre-calculation demands. One needs to obtain a sufficient embedding $\mathcal{E}_\mathcal{C}$ (see section 3.5.6) to initiate the calculation of coordinates. Additionally, a numerical scheme is required to calculate the coordinates, which typically cannot be trivially prallelized. As a result, the calculation of high quality EMC such as BBW or LBC can impose low run time performance. For instance, the calculation of BBW or LBC for an $\mathcal{E}_\mathcal{C}$ with over one million tetrahedra took several days on an Intel i9-11900K CPU and an NVIDIA RTX 3090 GPU. Thus, it is advised to offload the calculation of EMC on a remote machine and save the coordinates for later use.

The other coordinate types can be calculated efficiently for high-resolution models in a couple of seconds or even milliseconds. The actual run time performance depends on the implementation. Especially, the GBC with explicit formulas and the coordinates with normal control admit efficient parallelization on massively parallel GPUs. The probability-based coordinates require an additional numerical optimization step that can be performed efficiently. After coordinate computation, the deformation update either requires only the evaluation of eq. (2.16) or the additional evaluation of an affine sum of cage face normals and the computation of stretch factors. Each of these schemes to calculate the deformed mesh can be implemented efficiently for massively parallel processing so that fast deformation update is achieved for high-resolution models.

6.4.6. Element Quality of Deformed Tetrahedral Meshes using Cage-based Deformation

As numerical simulation for VP purposes requires sufficient element quality, we evaluate the preservation of element quality using dihedral angles φ and AMIPS (cf. section 3.2.2). While fast optimization algorithms like massively parallel harmonic mesh optimization (cf. chapter 4 and section 5.2) can improve element quality after deformation, this evaluation investigates if shape quality of tetrahedral elements can be preserved by cage-based deformation. In order to present a prototypical example for VP with many rounded and sharp features, the evaluation scenario covers the modeling of a door handle, where a flat handle shall be customized to be rounded. In the scenario, the engineer uses either a triangular or a quadrilateral cage for deformation control and relocates control vertices along the middle of the handle in order to create a rounded handle.

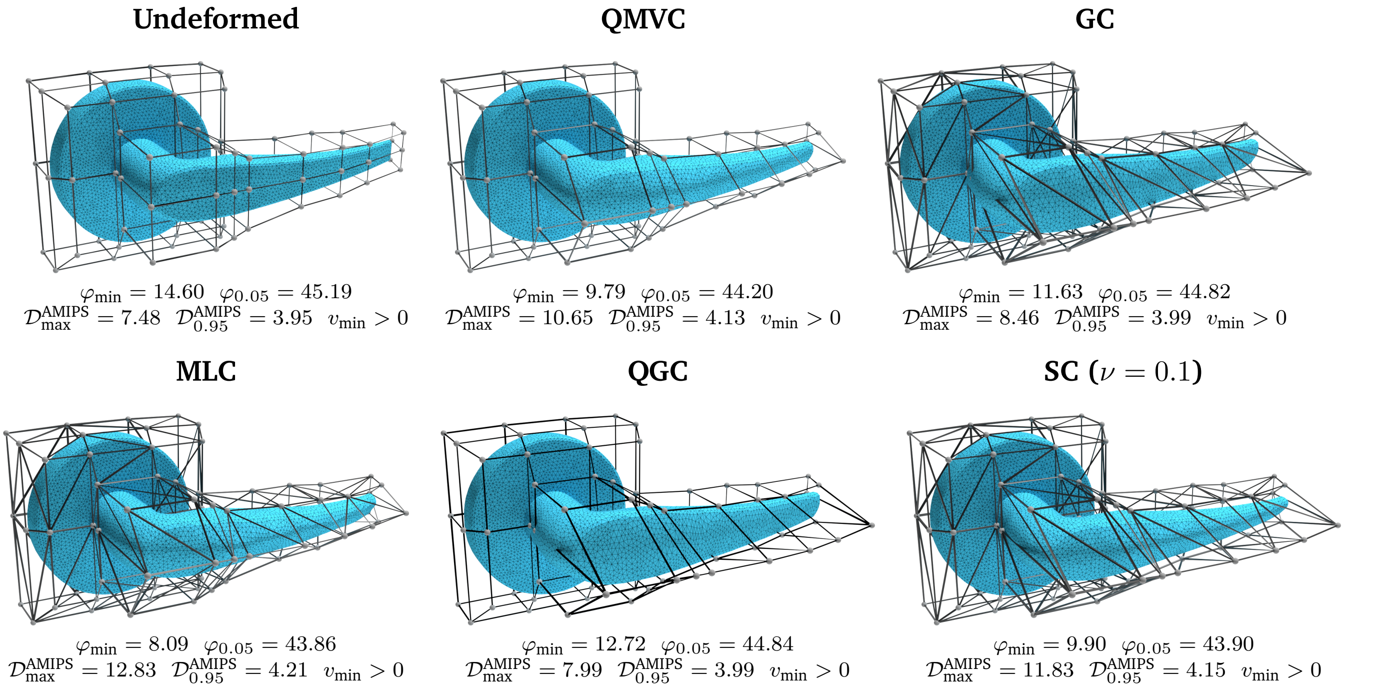

Undeformed
QMVC
GC
$\varphi_{\min} = 14.60 \quad \varphi_{0.05} = 45.19$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 7.48 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 3.95 \quad v_{\min} > 0$
$\varphi_{\min} = 9.79 \quad \varphi_{0.05} = 44.20$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 10.65 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 4.13 \quad v_{\min} > 0$
$\varphi_{\min} = 11.63 \quad \varphi_{0.05} = 44.82$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 8.46 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 3.99 \quad v_{\min} > 0$
MLC
QGC
SC ($\nu = 0.1$)
$\varphi_{\min} = 8.09 \quad \varphi_{0.05} = 43.86$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 12.83 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 4.21 \quad v_{\min} > 0$
$\varphi_{\min} = 12.72 \quad \varphi_{0.05} = 44.84$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 7.99 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 3.99 \quad v_{\min} > 0$
$\varphi_{\min} = 9.90 \quad \varphi_{0.05} = 43.90$
$\mathcal{D}^{\text{AMIPS}}_{\max} = 11.83 \quad \mathcal{D}^{\text{AMIPS}}_{0.95} = 4.15 \quad v_{\min} > 0$

Figure 6.14.: Deformed meshes and element qualities for GC and QGC. For element qualities, the figure provides maximal (worst) AMIPS $\mathcal{D}^{\text{AMIPS}}_{\max}$, the 95-percentile of AMIPS $\mathcal{D}^{\text{AMIPS}}_{0.95}$, minimal dihedral angle $\varphi_{\min}$, the 5-percentile of dihedral angles $\varphi_{0.05}$ and the sign of the minimal volume (absence of inversions).

As EMC need special investigation due to their dependency on an embedding, the evaluation first investigates the mesh quality using the other coordinate types. Figure 6.14 shows the results for the prototypical door handle model, while many models have been evaluated. Like many tasks in geometry modeling, the deformation of the door handle should preserve the symmetry of the surface features. For this reason, the evaluation uses a quad cage, if it is allowed by the coordinate type of use. Otherwise, the evaluation uses a triangulated cage to enable cage-based deformation for the coordinate type. None of the coordinate types in fig. 6.14 introduce inverted elements. The deformation with QMVC yields a smooth result with good preservation of mesh quality, while the symmetry of the door handle is not well preserved. The use of GC yields better element quality and preservation of the symmetry. However, the deformed geometry abounds from the cage, because GC are not interpolatory. The deformation with MLC introduces bumps at the handle, as MLC are not robust near the cage boundary. This leads to significantly more degradation of element quality compared to GC. Due to the support of quad cages, the deformation with QGC provides the best symmetry preservation. As the resulting geometry best preserves the shape of the input model, the use of QGC provides the best preservation of element quality. However, the deformed mesh abounds the cage, because QGC is like GC not interpolatory. For $\nu = 0.1$, the use of SC incurs less swelling of the volume compared to GC and QGC. This comes at the expense of less symmetry preservation, which results in more degradation of element quality. For all of the evaluated coordinate types, the deformation degrades tetrahedral element quality. Therefore, the use of cage-based deformation before a numerical simulation depends on the element quality of the undeformed tetrahedral mesh. After large deformation, it is potentially necessary to optimize element quality.

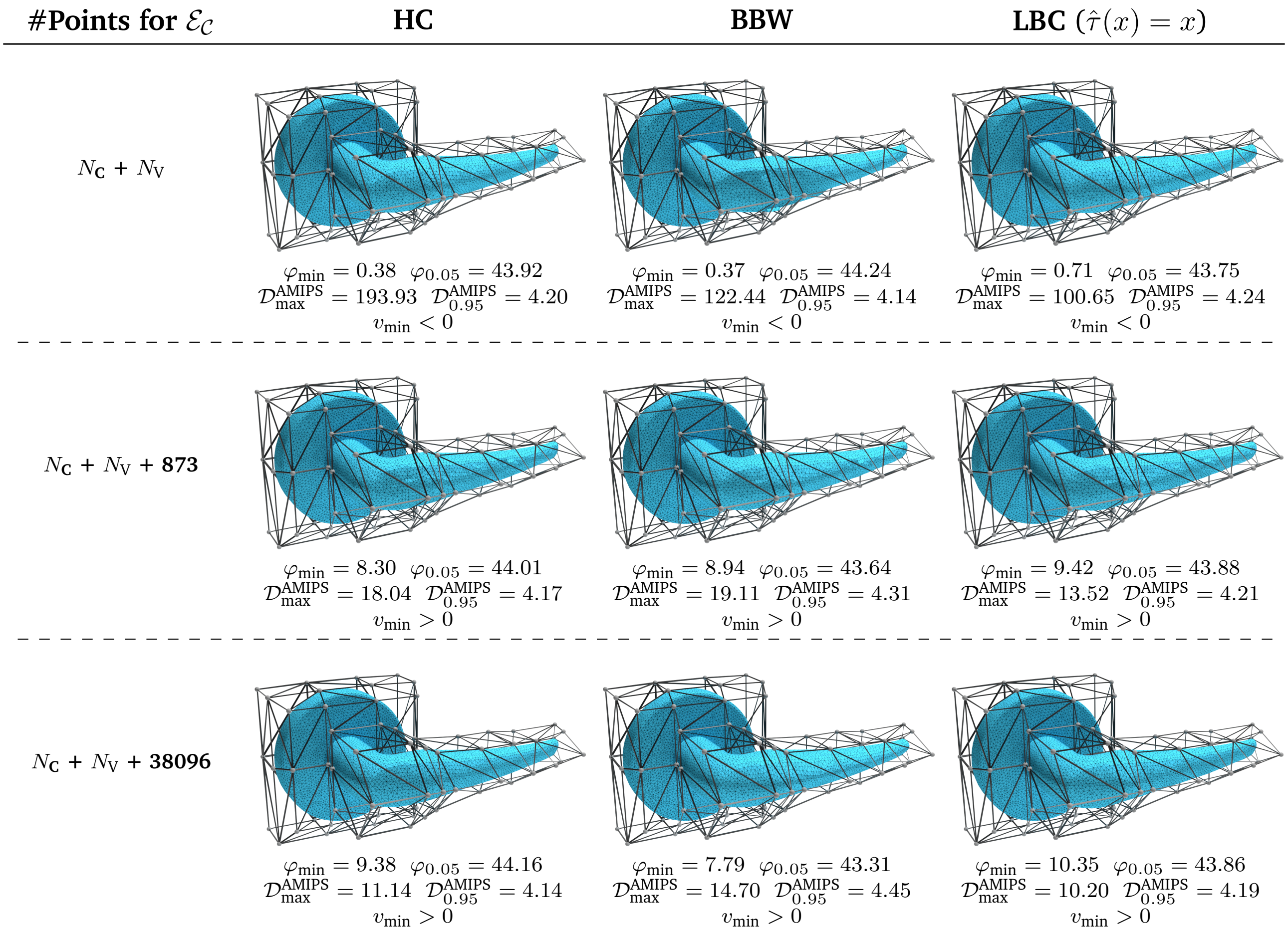

#Points for $\mathcal{E}_\mathcal{C}$
HC
BBW
LBC ($\hat{\tau}(x) = x$)
N_C + N_V
$\varphi_{\min} = 0.38$ $\varphi_{0.05} = 43.92$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 193.93$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.20$ $v_{\min} < 0$
$\varphi_{\min} = 0.37$ $\varphi_{0.05} = 44.24$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 122.44$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.14$ $v_{\min} < 0$
$\varphi_{\min} = 0.71$ $\varphi_{0.05} = 43.75$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 100.65$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.24$ $v_{\min} < 0$
N_C + N_V + 873
$\varphi_{\min} = 8.30$ $\varphi_{0.05} = 44.01$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 18.04$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.17$ $v_{\min} > 0$
$\varphi_{\min} = 8.94$ $\varphi_{0.05} = 43.64$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 19.11$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.31$ $v_{\min} > 0$
$\varphi_{\min} = 9.42$ $\varphi_{0.05} = 43.88$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 13.52$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.21$ $v_{\min} > 0$
N_C + N_V + 38096
$\varphi_{\min} = 9.38$ $\varphi_{0.05} = 44.16$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 11.14$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.14$ $v_{\min} > 0$
$\varphi_{\min} = 7.79$ $\varphi_{0.05} = 43.31$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 14.70$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.45$ $v_{\min} > 0$
$\varphi_{\min} = 10.35$ $\varphi_{0.05} = 43.86$ $\mathcal{D}^{\text{AMIPS}}_{\max} = 10.20$ $\mathcal{D}^{\text{AMIPS}}_{0.95} = 4.19$ $v_{\min} > 0$

Figure 6.15.: Resulting surfaces and element qualities of the deformed door handle model using EMC, i.e., HC, BBW, and LBC, with different numbers of points for $\mathcal{E}_\mathcal{C}$.

In terms of element quality, the evaluation reveals similar deformation results for the cage-based methods using a volumetric embedding $\mathcal{E}_\mathcal{C}$ (cf. section 3.6.2). Figure 6.15 compares resulting deformations of the door handle using EMC, i.e., HC, BBW as well as LBC for different resolutions of $\mathcal{E}_\mathcal{C}$. For the generation of $\mathcal{E}_\mathcal{C}$, Tetgen [Si20] receives as input the triangular cage, the vertices of the mesh $\mathcal{M}$ and sizing parameters (-a and -q switches). Deforming the door handle with HC, BBW or LBC achieves sufficient element quality for numerical simulations. However, when constructing $\mathcal{E}_\mathcal{C}$ only of the cage vertices and the vertices of $\mathcal{M}$, the HC, BBW and LBC coordinate types produce meshes of low-quality even including inverted elements. This is primarily because of the low number of sampling points defined on the boundary of the cage. In order to achieve better results, one needs to include more sampling points on the boundary of the cage. These additional sampling points need to be expressed as affine sums of the cage vertices to properly construct boundary conditions for numerically computing EMC.

The evaluation investigates the effect of adding additional points for $\mathcal{E}_C$, in order to investigate how many additional points are required for achieving sufficient element quality for numerical simulation. Typically, few points need to be added to achieve better element quality. For the door handle, empirical tests revealed that for adding only 873 points to $\mathcal{E}_C$ deformation using EMC already results in significantly better element quality. Generally, our evaluation shows that LBC tends to produce better element quality than HC and BBW, i.e., is more robust to the use of a low-resolution embedding. Although adding 873 points improves tetrahedral element qualities the boundaries of the resulting meshes are not smooth. Consequently, a finer $\mathcal{E}_C$ is necessary to achieve smoother surfaces. Throughout the evaluation it was difficult to predict how to setup the meshing tool such that a good trade-off between number of used elements and resulting deformation quality.

Consequently, the next step is to evaluate the results when using a high-resolution $\mathcal{E}_C$ with 38096 additional points, which results in smooth surfaces and good mesh quality for all the coordinates types of EMC. Taking a closer look at the resulting surfaces, a small bump appears in the deformed mesh, where three closely spaced control points are located. Due to the strong local influence of EMC, the spatial proximity of control vertices leads to an accumulation of the influence on $\mathcal{M}$ for the deformation. This leads to a larger deformation for the parts of $\mathcal{M}$ close to spatially dense control vertices, while smaller deformations occur for parts of $\mathcal{M}$ more distant to the control vertices. Among HC, BBW and LBC, the resulting bump is the largest when using LBC and the smallest when using HC.

6.4.7. Comparing Coordinate Types

As the various coordinate types possess different advantages and disadvantages, this thesis provides a comparison. Table 6.5 presents a comparative overview of the presented coordinate types.

The barycentric coordinates with explicit formulas offer a simple construction that is easy to apply to various use cases, whereas these coordinates impose several limitations. Many of these limitations have been overcome by other coordinate types. Especially the deformation artifacts for non-convex cages are an issue for VP applications. Thus, barycentric coordinates with explicit formulas should only be used for simple deformation that does not require non-convex cages.

The EMC offer improved locality for deformation control and should be used for applications where small deformation is needed to adjust local details while preserving the global shape. This can be useful in VP, because a high degree of locality ensures that the global shape of the model is well-preserved and only the local to-be-deformed features of the mesh are altered. Moreover, suitable applications for EMC are tolerant to long pre-calculation times. Therefore, one should prepare the use of EMC a considerable time before the deformation of high-resolution models.

For quick and shape preserving modeling applications in VP, the coordinates with normal control are most shape-preserving under large deformation. In particular the symmetry preservation of QGC can be useful to model prototypes with symmetric features. However, the coordinates with normal control are not interpolatory, which means that a swelling of the volume can occur. This can be mitigated with the use of SC, while this coordinate type currently does not support quad-layouts for cages.

The probability-based coordinates offer pointwise evaluation for topologically arbitrary cages, while MLC have only been evaluated for triangle cages up to now. The freedom of cage design can be interesting for modeling prototypes with irregular topology so that the cage can adapt well to the surface structure with arbitrary polygons that do not need to be triangulated. However, they do not offer the locality of EMC or the shape preservation of coordinates with normal control. While MLC exhibit good shape awareness, the cages need to be designed such that the distance between the to-be-deformed geometry and the cage is sufficiently

Table 6.5.: Comparison of cage coordinate types ordered by category (top to bottom: GBC with explicit formulas, EMC, probability-based coordinates, and coordinates with normal control) and year of publication.

coordinates	Lagrange property	pointwise evaluation	face type		extrapolation	comment
			triangle	quad		
MVC [FKR05]	✔	✔	✔	✘	✔	Simple and efficient
SBC [LBS06]	✔	✔	✔	planar	✔	Support for planar n-gon faces
PMVC [Lip+07]	✔	✔	✔	✘	✘	Non-negative MVC on the GPU
QMVC [TMB18]	✔	✔	✔	✔	✘	Use of tri-quad cages
HC [Jos+07]	✔	✘	✔	planar	✘	Local deformation influence
BBW [Jac+11]	✔	✘	✔	✘	✘	Enables the joint use of cages, skeletons, and point handles
LBC [Zha+14]	✔	✘	✔	✘	✘	Enables to control locality
MEC [HS08]	✔	✔	✔	✔	✘	Direct evaluation for arbitrary polygon-cages
MLC [CDH23]	✔	✔	✔	✘	✘	Better shape awareness than MEC
GC [LLC08]	✘	✔	✔	✘	✔	Conformal mapping
QGC [TB22]	✘	✔	✔	✔	✔	Symmetry preserving
SC [CDD23]	✘	✔	✔	✘	✔	Volume control

large to avoid deformation artifacts.

It is worth to highlight, that the recent advances in better preservation of shape features, locality and support for quad-layouts provides good means for usage of cage-based deformation to model the geometry of smooth and organic shapes, because these traits enable the modification of a prototype without introducing unintended deformation artifacts.

6.5. Summary

In summary, this chapter has presented methods for user-interactive feature-based editing and deformation of unstructured tetrahedral meshes. Both types of methods have validated as useful for editing meshes to save loop back to CAD in VP cycles. In addition, both types of methods cannot guarantee sufficient element shape quality for numerical simulation.

For feature-based editing of unstructured tetrahedral meshes, this chapter has presented face groups that allow to select a large part of the model surface at once. The face groups can either originate from CAD or they can be determined by the curvature of the surface. With the use of the face groups, the user can specify a part of the model, which is modified by an editing operation. On the basis of face groups, this chapter has presented two editing operations: hole closing and mesh erosion. For both editing operations, this chapter has introduced algorithms that perform expensive steps on the GPU. This has led to fast execution times, which provides good means for interactive mesh editing. The quality of the resulting meshes was sufficient for numerical simulation so that a mesh editing operation could be followed by FEA to inspect the impact of the model customization. However, for both operations, the evaluation has revealed that element quality can degrade. Therefore, complex mesh optimization and re-meshing might be necessary after mesh editing.

For deforming the shape of a non-mechanical prototype, this chapter has investigated the facilities of cage-based deformation to deform smooth and organic-like geometries. Among the available methods to quickly generate a cage for deformation control, the interactive cage generation is the most useful, because it enables the users to specify the semantic parts of the model so that the generation places control vertices, where the user intends to deform the model. Many methods are suitable for interactive mesh modeling of prototypes (cf. section 6.4.7). The most relevant coordinate types have been implemented in the publicly available CageModeler tool set. The CageModeler application also enable the cage-based deformation of unstructured tetrahedral meshes. The set of available methods for cage-based deformation includes one method for the most desirable design goals, e.g., MVC for simplicity of implementation, LBC for neat local influence of cage vertices or QGC for proper shape preservation. Each method has its own advantages and shortcomings. Thus, there is not one method that ultimately trumps the others. Once the cage is constructed, the run time performance of cage based deformation allows for interactive deformation with the exception of computing EMC. The evaluation of element shape quality has revealed that tetrahedral element quality preservation benefits from the smoothness properties of deformation, while the element quality can degrade throughout the deformation process. Thus, deforming the mesh for VP should be coupled with fast mesh optimization.

In view of RQ3[1], this chapter has shown that quick interactive editing of unstructured tetrahedral meshes is possible with algorithms that are amenable to massively parallel processing. The resulting meshes are typically suitable for downstream FEA such as a massively parallel FEM algorithm [Web+13; Web+15]. This enables fast customization and analysis of prototypes without loop back to CAD. Similarly to morphing [Sta+11], the proposed algorithms can lead to degradation of element quality and in some cases require intervention with element quality optimization, in order to provide FEA of the prototype. While this chapter has addressed the modeling of prototypes for numerical, the subsequent chapter is concerned with the visual analysis of the simulation results.

[1] RQ3: *How can massively parallel mesh processing be used for quick editing of unstructured tetrahedral meshes to accelerate VP cycles?*

7. Massively Parallel Post Processing of Unstructured Tetrahedral Meshes for Analysis

The following papers contain the core content of this chapter:

[Str+20] **D. Ströter**, J. S. Mueller-Roemer, A. Stork, D. W. Fellner, "OLBVH: octree linear bounding volume hierarchy for volumetric meshes". In: *The Visual Computer* 36.10-12 (July 2020). **Honorable mention from Fraunhofer IGD for best papers in the category "Impact on Science"**, Presented at Computer Graphics International 2020, pp. 2327–2340. DOI: 10.1007/s00371-020-01886-6

[SSF23] **D. Ströter**, A. Stork, D. W. Fellner, "Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes". In: *High-Performance Graphics - Symposium Papers*. Ed. by Jacco Bikker and Christiaan Gribble. Presented at High-Performance Graphics 2023. The Eurographics Association, 2023. DOI: 10.2312/hpg.20231139

[Bue+24] M. v. Buelow, **D. Ströter**, A. Rak, D. W. Fellner, "A Visual Profiling System for Direct Volume Rendering". In: *Eurographics 2024 - Short Papers*. Ed. by Ruizhen Hu and Panayiotis Charalambous. The Eurographics Association, 2024. DOI: 10.2312/egs.20241030

In addition to addressing the editing of meshes in chapters 4 to 6, this chapter presents massively parallel post-processing methods of unstructured tetrahedral meshes to investigate RQ4. In order to allow for quick spatial search, this chapter presents a memory-efficient spatial data structure for various post processing tasks on unstructured tetrahedral meshes. The proposed data structure is denoted as octree linear bounding volume hierarchy (OLBVH), because the bounding volumes of primitives define an octree-partition of the space and nodes are aligned in a linear order in memory. An earlier version of the OLBVH was developed in the author's master thesis [Str19]. For this PhD thesis, several significant improvements of the OLBVH and new algorithms using the OLBVH have been developed. Besides minor optimizations such as more sophisticated usage of GPU-primitives (cf. section 2.2.2), this PhD thesis presents a significantly extended version of the OLBVH. The following list presents the most relevant improvements to the OLBVH since the master thesis:

- The data structure layout uses a prefix sum for primitive offsets (one integer per node) instead of ranges (two integers per node).
- Instead of approximating the number of primitives for each potential tree node, a heuristic based on the average edge length determines the maximum tree level.
- The construction determines offsets PO for primitives at the split recording step.
- On hierarchy construction, the PO are used to infer the child node indices of a tree node.
- Short stack traversal allows for leveraging faster shared memory (see section 7.1.4).

In order to enable post-processing applications to benefit from the memory-efficient OLBVH, this chapter presents several massively parallel algorithms, which use the OLBVH to provide fast post-processing. As DVR enables exploration of the simulation results in the interior of the unstructured tetrahedral mesh (cf. section 2.4), this chapter introduces an algorithm for DVR using the OLBVH. Unlike structured data such as voxel grids, the DVR of unstructured tetrahedral meshes still poses a challenge to memory management, because it needs to manage the topology, geometry, and scalar field to locate and render tetrahedral elements [Sar+23]. Besides the memory efficiency, the run time performance of DVR is important for interactive exploration of simulation results. Therefore, the design decisions of the intended DVR algorithm should not only optimize memory usage but also run time performance.

As memory efficiency of DVR not only depends on the spatial data structure of use, this chapter also attempts to use the re-meshing methods presented in chapter 5 to compress meshes for DVR. Potentially, high-resolution meshes can be compressed for DVR, which can significantly reduce memory consumption. However, the compression of the meshes should preserve the fidelity of the simulation results, because users require an as truthful as possible rendering of the simulation results.

Another algorithm presented in this chapter attempts to facilitate the 3D printing of a prototype for analysis. This provides a physical prototype so that product development can benefit from real-world tests of the prototype during the VP process.

The OLBVH spatial data structure is presented in section 7.1. A description of the DVR algorithm appears in section 7.2. In order to improve memory efficiency beyond usage of the OLBVH, section 7.3 describes how to coarsen meshes for DVR. A critical evaluation of the proposed DVR methods appears in section 7.4. For 3D printing of the prototype, section 7.5 describes and evaluates an algorithm for conservative slicing. Finally, section 7.6 summarizes the key conclusions of this chapter.

7.1. Octree Linear Bounding Volume Hierarchy

This section describes the basic concept of the OLBVH data structure and details the implementation (see sections 7.1.1 and 7.1.2) of the data structure as well as construction (see section 7.1.3) and traversal algorithms (see section 7.1.4).

7.1.1. Quantization of AABBs along the Morton Curve

Like previous LBVH variants (see section 3.8.1), the OLBVH relies on approximate spatial sorting of elements ordered by Morton code [Mor66]. This quantization encodes the input coordinates x, y, and z as l-bit integers $\hat{x} = (x_{l-1}, x_{l-2}, \ldots, x_0)$, interleaving their bits.

$$m(\hat{x}, \hat{y}, \hat{z}) = (x_{l-1}, y_{l-1}, z_{l-1}, \ldots, x_0, y_0, z_0)$$

The spatial domain of the quantization is the enclosing axis aligned bounding box (AABB) $H(\mathcal{M})$ of the volumetric mesh $\mathcal{M}$:

$$H(\mathcal{M}) = \left((x_{\text{min}}^{\mathcal{M}}, y_{\text{min}}^{\mathcal{M}}, z_{\text{min}}^{\mathcal{M}})^\mathsf{T}, (x_{\text{max}}^{\mathcal{M}}, y_{\text{max}}^{\mathcal{M}}, z_{\text{max}}^{\mathcal{M}})^\mathsf{T}\right) = \left(\mathbf{x}_{\text{min}}^{\mathcal{M}}, \mathbf{x}_{\text{max}}^{\mathcal{M}}\right).$$

A key difference to previous LBVH variants is that the OLBVH does not encode the centroids of AABBs. Instead, the quantization relies on the fact that Morton codes inherently span an axis-equidistant grid. The resolution of this grid depends on the number of bits l used for quantization. The cell size of the Morton grid can be calculated as:

$$\mathbf{s}^l = (s_x^l, s_y^l, s_z^l)^\mathsf{T} = \frac{\mathbf{x}_{\text{max}}^{\mathcal{M}} - \mathbf{x}_{\text{min}}^{\mathcal{M}}}{2^l - 1}. \tag{7.1}$$

Using this size, quantization and reconstruction, respectively, follow the equations below:

$$q(x) = \left\lfloor \frac{x - x^{\mathcal{M}}_{\text{min}}}{s^l_x} + \frac{1}{2} \right\rfloor = \hat{x}$$

$$q^{-1}(\hat{x}) = x^{\mathcal{M}}_{\text{min}} + \hat{x} \cdot s^l_x$$

The quantized AABB $\hat{H}(\mathsf{p})$ of a primitive $\mathsf{p} \in \mathcal{M}$ is defined by two quantized points:

$$\hat{H}(\mathsf{p}) = \left(\hat{\mathbf{x}}^{\mathsf{p}}_{\text{min}}, \hat{\mathbf{x}}^{\mathsf{p}}_{\text{max}}\right).$$

For each primitive $\mathsf{p} \in \mathcal{M}$, the quantization generates the Morton codes enclosed by the primitive's quantized AABB $\hat{H}(\mathsf{p})$. The set of generated Morton codes for a primitive $\mathsf{p} \in \mathcal{M}$ is given by:

$$M(\hat{H}(\mathsf{p})) = \left\{ m(\hat{x}', \hat{y}', \hat{z}') \mid \bigwedge_{c \,\in\, \{x,y,z\}} \hat{c}^{\mathsf{p}}_{\text{min}} \leq \hat{c}' \leq \hat{c}^{\mathsf{p}}_{\text{max}} \right\}.$$

Combined with a heuristic to determine the tree depth $L = l + 1$ (the root level is present, even for 0-bit Morton codes), the OLBVH construction uses $M(\hat{H}(\mathsf{p}))$ to split primitives a priori and eliminate looseness, i.e., sibling cells never overlap. Additionally, the OLBVH stores boundary flags at every node of the tree, allowing for early termination at nodes containing only interior cells when only determining if a point or bounding box is inside or outside the mesh $\mathcal{M}$. As interior primitives near the boundary must share points with boundary primitives and the sets $M(\hat{H}(\mathsf{p}))$ are inclusive, cells that contain empty space must always contain boundary primitives provided that no primitive has a negative signed volume.

7.1.2. Data Structure Layout

The OLBVH data structure consists of six arrays containing:

1. primitive indices $\mathsf{P}[N_m]$ in $[0, N_p)$ sorted by Morton code,
2. tree node bounding volumes $\mathsf{BV}[N_n]$,
3. child node offsets $\mathsf{CO}[n_{n_i} + 1]$ in $[0, N_n)$,
4. primitive index offsets $\mathsf{PO}[N_n + 1]$ in $[0, N_m]$,
5. boolean boundary flags $\mathsf{BF}[N_n]$,
6. per-level node offsets $\mathsf{NO}[L]$ in $[1, N_n]$,

where N_p is the number of primitives in $\mathcal{M}$, N_m is sum of the numbers of Morton codes per primitive, N_n is the number of tree nodes, and N_{n_i} is the number of internal tree nodes excluding leaves.

As the bounding volume of a primitive may intersect with several spatial cells of the Morton grid, P may contain duplicate indices. The bounding volumes of tree nodes BV are laid out linearly in memory following a breadth-first traversal order. In order to allow for top-down traversal, CO stores child offsets for the tree nodes. Every tree node is associated with a maximum of eight children. Due to the contiguous levelwise order in memory, the CO array enables retrieval of the child node indices C_i of a given tree node i:

$$C_i = \{j \mid \mathsf{CO}[i] < j \leq \mathsf{CO}[i+1]\}. \tag{7.2}$$

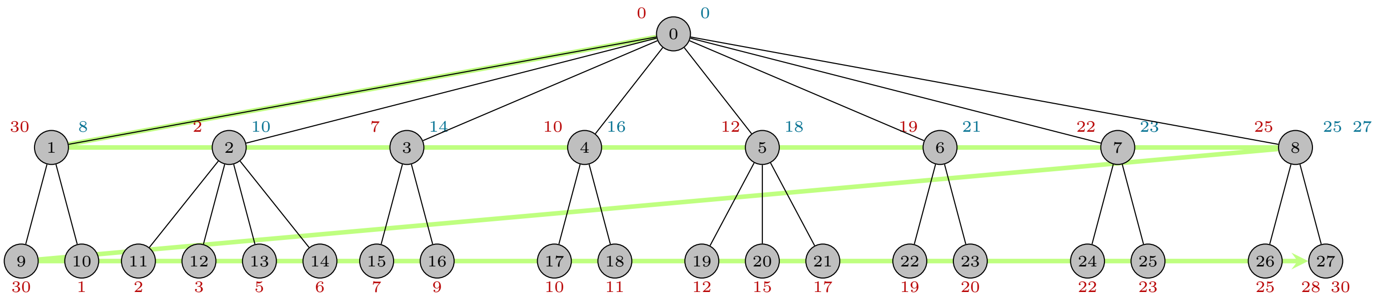


Figure 7.1.: This figure shows the data layout of the CO and PO arrays for a sample tree. The gray circles represent the tree nodes. The blue numbers above the tree nodes represent entries of the CO array and the red numbers represent entries of the PO array. The green arrow in the background indicates the in-memory order.

In order to manage memory efficiently, the CO array only contains entries for internal, i.e., non-leaf, nodes. As a result of the chosen data structure, the node index of an internal node's rightmost child is equal to the child offset of its succeeding node in memory, i.e,

$$\text{Node } i \text{ is } j\text{'s rightmost child} \implies \mathsf{CO}[j+1] = i. \tag{7.3}$$

This property is useful while constructing the internal hierarchy levels starting from the leaves.

The OLBVH also incorporates a primitive offsets array PO to infer the primitive indices in P are associated with node i. As nodes reside in memory in breadth-first order, it is necessary to handle adjacent nodes of different hierarchical levels:

$$P_i = \begin{cases} \{\mathsf{P}[j] \mid 0 \le j < \mathsf{PO}[i+1]\}, & \text{if } \mathsf{PO}[i] > \mathsf{PO}[i+1] \\ \{\mathsf{P}[j] \mid \mathsf{PO}[i] \le j < \mathsf{PO}[i+1]\}, & \text{otherwise.} \end{cases} \tag{7.4}$$

As any rightmost child has the same upper offset as its parent node, it holds that:

$$\text{Node } i \text{ is } j\text{'s rightmost child} \implies \mathsf{PO}[i+1] = \mathsf{PO}[j+1]. \tag{7.5}$$

The memory layout of the CO and PO arrays is illustrated for a sample tree in fig. 7.1. The boundary flag array BF stores a boolean boundary flag for each tree node i. The final array NO stores the node index offsets for each level. It is used to determine the hierarchical level of a node given its index.

7.1.3. Construction

The OLBVH construction receives any volumetric mesh $\mathcal{M}$ with marked boundary primitives as input, the following construction is designed with unstructured tetrahedral meshes in mind. However, OLBVH is applicable to general polyhedral meshes. The construction relies on the TCSR mesh data structure (cf. section 2.2.3) to efficiently store and process tetrahedral meshes on the GPU, and to extract the mesh boundary (cf. section 4.2). If $H(\mathcal{M})$ is not known beforehand, a parallel reduction (cf. section 2.2.2) on the mesh's vertices calculates $\mathbf{x}_{\text{min}}^{\mathcal{M}}$ and $\mathbf{x}_{\text{max}}^{\mathcal{M}}$. The construction is performed in four steps:

1. Determine the tree depth heuristically
2. Calculate and sort Morton codes of primitive AABBs
3. Record at which levels the sorted Morton codes split
4. Bottom up construction based on split positions

Algorithm 8 Leaf node bounding volume and boundary flag determination.

```
procedure GENERATELEAFNODES($l$)
    for all $i \in [\mathtt{NO}[l-1], \mathtt{NO}[l])$ do                ▷ In parallel
        $p \leftarrow \mathtt{P}[\mathtt{PO}[i+1]-1]$                    ▷ Pick any entry in $P_i$
        $m_{x,y,z} \leftarrow \mathtt{MC}[p]$                            ▷ Calculate AABB
        $min \leftarrow q^{-1}(m^{-1}(m_{x,y,z}))$
        $max \leftarrow min + s^l$
        $\mathtt{BV}[i] \leftarrow (min,\ max)$
        for all $\mathsf{p} \in P_i$ do                                  ▷ Calculate boundary flag
            if $\mathsf{p}$ is marked as boundary primitive then
                $\mathtt{BF}[i] \leftarrow true$
                return
            end if
        end for
        $\mathtt{BF}[i] \leftarrow false$
    end for
end procedure
```

In the initial stage, the construction determines the number $l \leq l_{\max}$ of bits to use for quantization, and therefore the depth of the tree, according to element size. The maximum possible number of quantization bits is $l_{\max} = 10$ in the implementation, as it uses 32-bit integers to store Morton codes. In parallel over primitives, the construction estimates the tree level by first computing the binary logarithm ld of the largest tetrahedron AABB axis in relation to the maximum grid resolution:

$$l_\alpha(\mathsf{p}) = \max\left(\left\lfloor \mathrm{ld} \left\lfloor \frac{\mathbf{x}^{\mathsf{p}}_{\max} - \mathbf{x}^{\mathsf{p}}_{\min}}{\mathbf{s}^{l_{\max}}} \right\rfloor_{\max} \right\rfloor, 0\right),$$

where the division is performed per component. The binary logarithm can be efficiently implemented using a count leading zeros instruction (the `__clz` intrinsic in CUDA). Subsequently, the construction procedure chooses

$$l = \mathrm{clamp}\left(\left\lfloor 10.5 - \left(\underset{\mathsf{p}\in\mathcal{M}}{\mathrm{average}}\left(l_\alpha(\mathsf{p})\right) + l_\alpha\right)\right\rfloor, 0, 10\right)$$

by performing a parallel reduction on $l_\alpha(\mathsf{p})$, where $l_\alpha \in [-10, 10]$ is a tuning parameter. Though it is of course possible to choose a negative l_α, caution is advised, as the resulting Morton grid would be more fine grained than most primitive AABBs. This may lead to oversampling situations and increased memory consumption. Therefore, we define $l_\alpha \in [0, 10]$. Increasing l_α results in a coarser Morton grid, and thereby a faster construction and reduced use of memory, while decreasing l_α leads to slower construction and more memory consumption while potentially improving rendering performance.

Since the maximum level is heuristically determined, the tree construction proceeds with calculating the Morton codes in the second step. The size of the Morton grid cells $\mathbf{s}^l$ can be computed using eq. (7.1). On the basis of $\mathbf{s}^l$, the construction allocates a temporary array of Morton codes MC. As the overall number of Morton codes N_m is not known a priori, one parallel passes over primitives determines N_m to allocate MC. A subsequent parallel pass over primitives fills MC. Each thread calculates $\left|M(\hat{H}(\mathsf{p}))\right|$ for one primitive. A parallel exclusive prefix sum determines the array offsets the Morton codes of each primitive are written to. By performing the prefix sum for N_p+1 elements, the final offset corresponds to N_m and additional branching is avoided. A second pass over all primitives generates the sets $M(\hat{H}(\mathsf{p}))$ and writes the Morton codes into MC at the computed offsets, while also writing each primitive's index to P at the corresponding offset.

With a parallel sort, the primitive indices P are sorted using the Morton codes in MC as keys. As a result, the primitive indices are ordered along the Z-curve. Primitive indices are potentially duplicated, because

$M(\hat{H}(\mathtt{p}))$ may include several Morton codes. Additionally, the root node of the OLBVH is constructed at this point. Because the root node encloses all primitives $\mathtt{p} \in \mathcal{M}$, its AABB is set to $H(\mathcal{M})$. Moreover, the two initial entries of PO are set to 0 and the overall number of Morton codes N_m, respectively. Straightforwardly, the first entry of CO is 0, the first entry of NO is 1, and the last is N_n. Additionally, the root boundary flag $\mathtt{BF}[0]$ is set to *true*.

Algorithm 9 Construction of internal nodes of level l_i.

```
procedure GENERATEINTERNALNODES($l_i$)
    for all $i \in [\mathtt{NO}[l_i], \mathtt{NO}[l_i+1])$ do                          ▷ In parallel
        $\mathtt{CI}[\mathtt{PO}[i+1]] \leftarrow i$
    end for
    $\mathtt{CI}[0] \leftarrow \mathtt{NO}[l_i] - 1$
    for all $i \in [\mathtt{NO}[l_i-1], \mathtt{NO}[l_i])$ do                          ▷ In parallel
        $(begin,\ end) \leftarrow (\mathtt{PO}[i], \mathtt{PO}[i+1])$
        if $begin > end$ then                                    ▷ Calculate primitive range
            $begin \leftarrow 0$
        end if
        $\mathtt{CO}[i] \leftarrow \mathtt{CI}[begin]$
        $c_{\text{begin}} \leftarrow \mathtt{CO}[i] + 1$
        $c_{\text{end}} \leftarrow \mathtt{CI}[end]$
        if $i = \mathtt{NO}[l_i] - 1$ then                                   ▷ Set last child offset
            $\mathtt{CO}[i+1] \leftarrow c_{\text{end}}$
        end if
        $\mathtt{BV}[i] \leftarrow \mathtt{BV}[c_{\text{begin}}] \cup \cdots \cup \mathtt{BV}[c_{\text{end}}]$      ▷ Merge AABBs
        $\mathtt{BF}[i] \leftarrow false$
        for $j \leftarrow c_{\text{begin}}, \ldots, c_{\text{end}}$ do                                   ▷ $j \in C_i$
            if $\mathtt{BF}[j]$ then                                   ▷ Calculate boundary flag
                $\mathtt{BF}[i] \leftarrow true$
            end if
        end for
    end for
end procedure
```

The third construction step determines the indices at which the Morton codes in MC indicate splits between BVH nodes. As the OLBVH is an octree, the construction compares Morton codes advancing in 3-bit steps beginning from the most significant bit. Thus, each split is associated with a level $l_c < L$ and an index referring to a Morton code in the MC array. As the split positions refer to the Morton codes of primitives, the split generation procedure writes these positions to the PO array. As in the case of Morton code generation, the construction first determines the number of splits between neighboring codes. A parallel scan is then used to determine the size of and offsets into the corresponding arrays. As no split is recorded for the last Morton code, a split is appended sequentially by the CPU for each level. This can be done concurrently while filling the remainder of the PO. A parallel stable sort by key procedure sorts PO by split level resulting in the intended primitive offset order. As the splits are sorted by level, the remaining entries of the NO array can be calculated by exploiting the per-level contiguous order. A parallel pass over splits searches for adjacent splits of different level. If two such splits are found at positions i and $i+1$, a node level offset was found and thus $\mathtt{NO}[splits[i].l_c] = i + 1$.

The final step relies on NO and PO to construct the OLBVH in a bottom-up manner, i.e., starting at the leaf level. As the data layout does not allow for immediate inference of the parent node for a given node, it is not possible to construct the hierarchy in a single kernel launch using atomic flags as in Apetrei's [Ape14] agglomerative LBVH builder. Nonetheless, it is feasible to construct the OLBVH's topology in separate kernel launches.

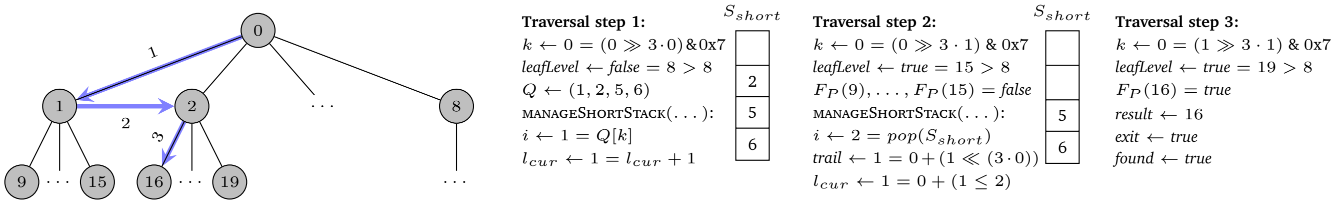
Firstly, algorithm 8 calculates the leaf node AABBs and boundary flags BF in parallel over leaf nodes.

As all Morton codes associated with a leaf node are equal, the minimum AABB coordinates are obtained by calculating $q^{-1}(m^{-1}(m_{x,y,z}))$ for an arbitrary $m_{x,y,z}$ associated with the leaf node. Furthermore, the maximum AABB coordinates are calculated by adding the grid size $\mathbf{s}^l$ to the minimum coordinates. Therefore, leaf node AABBs can be calculated in constant time. The boundary flag $\mathtt{BF}[i]$ of a leaf node is set if any primitive in P_i is tagged as a boundary element. Therefore, boundary flag calculation for a leaf node is $\mathcal{O}(|P_i|)$.

The generation of internal tree nodes relies on an auxiliary array CI of the same size as the overall number of generated Morton codes N_m. Algorithm 9 outlines internal node generation. Algorithm 9 writes node indices to the primitive offset positions of CI for each level. On construction of the parent hierarchical level, algorithm 9 exploits eq. (7.5) to lookup the rightmost child indices for internal nodes at the upper primitive offset positions of the CI array. To reduce the number of kernel launches, the computation of CI can be merged with GENERATELEAFNODES for the first iteration. Between internal levels, a separate kernel must be used to prevent conflicting reads and writes. CI and PO are used to determine the child offsets CO according to eq. (7.3). Finally, internal node AABBs and boundary flags are determined from their children.

7.1.4. Traversal

As the efficient usage of the OLBVH depends on fast traversal, this section presents a method for GPU-parallel traversal of the OLBVH. Algorithm 10 outlines the OLBVH traversal approach used in this thesis. Figure 7.2 presents a sample traversal execution. Traversing the OLBVH for input geometric predicates F_N for nodes and F_P for primitives proceeds by pushing child nodes to a traversal stack. If a full traversal stack is used, its size is bounded by $7 \cdot l + 8$ node indices. This typically exceeds the amount of available fast on-chip shared memory, imposing the use of slower cache-backed local memory (cf. section 2.2). Therefore, OLBVH traversal uses a short stack [VWB19] to reduce the memory requirement for the stack. As a result of the shallow hierarchy, a 32-bit integer is sufficient to encode the trail. Leaf nodes can be determined efficiently by comparison with the lower level offset of the last level. Since siblings are on the same level, it suffices to check the first child node. The short stack approach uses a queue Q to record the nodes in C_i (cf. eq. (7.2)) for which the predicate F_N evaluates as true.



Traversal step 1:
$k \leftarrow 0 = (0 \gg 3 \cdot 0) \& 0x7$
leafLevel $\leftarrow$ *false* $= 8 > 8$
$Q \leftarrow (1, 2, 5, 6)$
MANAGESHORTSTACK(. . .):
$i \leftarrow 1 = Q[k]$
$l_{cur} \leftarrow 1 = l_{cur} + 1$

Traversal step 2:
$k \leftarrow 0 = (0 \gg 3 \cdot 1) \& 0x7$
leafLevel $\leftarrow$ *true* $= 15 > 8$
$F_P(9), \ldots, F_P(15) =$ *false*
MANAGESHORTSTACK(. . .):
$i \leftarrow 2 = pop(S_{short})$
trail $\leftarrow 1 = 0 + (1 \ll (3 \cdot 0))$
$l_{cur} \leftarrow 1 = 0 + (1 \leq 2)$

Traversal step 3:
$k \leftarrow 0 = (1 \gg 3 \cdot 1) \& 0x7$
leafLevel $\leftarrow$ *true* $= 19 > 8$
$F_P(16) =$ *true*
result $\leftarrow 16$
exit $\leftarrow$ *true*
found $\leftarrow$ *true*

Figure 7.2.: Traversal proceeds on the compacted tree until a leaf node satisfies F_N. The blue arrows indicate the traversal path. Variable assignments and short stack states appear below the tree. Initially, traversal concludes that F_N holds for nodes 1, 2, 5 and 6. Thus, the next node is 1, while nodes 2, 5 and 6 remain on the short stack. In the second step, none of 1's children satisfy F_N. Thus, traversal pops 2 from the short stack determining the parent level to increment the trail and find the current level using eq. (7.6). Finally, F_N holds for 16 and traversal terminates.

Unlike Vaidyanathan et al.'s [VWB19] short stack implementation, the OLBVH traversal does not require to tag parent nodes, avoiding unnecessary branching. Additionally, the OLBVH traversal does not treat leaf nodes satisfying F_N throughout traversal. This is due to the fact that depending on l_α leaf nodes contain several primitives and a high primitive count may lead to significant thread divergence. After pushing intersecting child nodes to Q, the MANAGESHORTSTACK(. . .) procedure pushes tree nodes from Q to the short stack and identifies the tree node at which traversal continues. This procedure corresponds to lines 10-27 of Vaidyanathan et al.'s BVH$-N$ traversal method with $N = 8$. The only two adaptations are the bit-trail handling and comparison of $\texttt{NO}[parentLevel]$ to determine the level of a node on a short stack pop to avoid unnecessary branching (lines 16-20 in Stack Pop):

$$l_{cur} = parentLevel + (\texttt{NO}[parentLevel] \leq n) \tag{7.6}$$

Algorithm 10 OLBVH traversal using a short stack

```
procedure TRAVERSEOLBVH($F_N$, $F_P$)
    $trail \leftarrow 0\text{x}0$
    $l_{cur} \leftarrow 0$
    $S_{short} \leftarrow \emptyset$
    $i \leftarrow 0$                                              ▷ Initiate at root
    $result \leftarrow 0$                                         ▷ Alternatively, $N \leftarrow \emptyset$
    $found \leftarrow false$
    $nodeBound \leftarrow \texttt{NO}[l - 1]$
    $exit \leftarrow false$
    do
        $Q \leftarrow \emptyset$
        $k \leftarrow (trail \gg 3 \cdot l_{cur})$ & $0\text{x}7$
        $leafLevel \leftarrow \texttt{CO}[i + 1] > nodeBound$
        for $j \in C_i$ do
            if $F_N(j)$ then
                if $leafLevel$ then
                    $exit \leftarrow true$
                    $found \leftarrow true$
                    $result \leftarrow j$                         ▷ Alternatively, push $j$ onto $N$
                else
                    $Q \leftarrow Q \cup \{j\}$
                end if
            end if
        end for
        MANAGESHORTSTACK($S_{short}$, $trail$, $l_{cur}$, $i$, $Q$, $k$, $exit$)
    while $\neg exit$
    if $found$ then                      ▷ Alternatively, test intersection for every $j \in N$
        if $F_P(\mathbb{P}_{\mathcal{M}})$ for any $\mathrm{p}_{\mathcal{M}} \in P_{result}$ then      ▷ cf. eq. (7.4)
            treat incident
            return
        end if
    end if
end procedure
```

In many applications, it is sufficient to traverse the tree evaluating F_N until a leaf node is reached, since sibling cells never overlap. Primitives associated with the resulting leaf node are then checked until F_P evaluates to true. However, for some applications such as intersection detection it is required to keep track of intersecting leaf nodes and to test primitives for intersection. In such scenarios, a node set N is used to maintain leaf nodes satisfying F_N. This approach is inherently prone to overflow. Thus, additional treatment is required in order to prevent exceeding the size of N on node push. A possible solution is to evaluate F_P for the primitives of all nodes in N if a node push would result in an overflow. After traversal, one needs to evaluate F_P only for the primitives associated with the resulting leaf node or the set of leaf nodes N.

7.2. Direct Volume Rendering using the OLBVH

As one of the concerns of this thesis is DVR of unstructured tetrahedral meshes (see section 2.4) for visual analysis purposes, this section presents algorithms for rendering volumetric meshes using the OLBVH. A pre-process before ray marching determines relevant intervals along view rays to avoid unnecessary overheads due to sampling empty space. Section 7.2.1 presents this quick pre-process. After the pre-process, a ray marching algorithm uses the OLBVH and the ray intervals to sample the scalar field defined on the unstructured tetrahedral mesh. The ray marching procedure appears in section 7.2.2.

7.2.1. Empty Space Skipping

As DVR of unstructured meshes requires repeated point location along each view ray, a potential performance issue is that a large number of samples may not hit the geometry. For this reason, DVR benefits from the use of the boundary marking to efficiently compute the relevant ray interval in which the ray intersects the geometry. Prior to ray marching, a quick pre-process traverses only the boundary nodes of the OLBVH for each ray and determines the intervals for which each ray intersects leaf node AABBs. Ray marching then performs point sampling only for the interval of the ray from the first hit to the last hit of boundary leaf nodes. As a result, the number of samples for ray marching is significantly reduced. A visualization of the resulting space skipping appears in fig. 7.3. The method for the calculation of the ray intervals uses the short stack traversal described in section 7.1.4. While the following describes this method, algorithm 11 draws an outline.

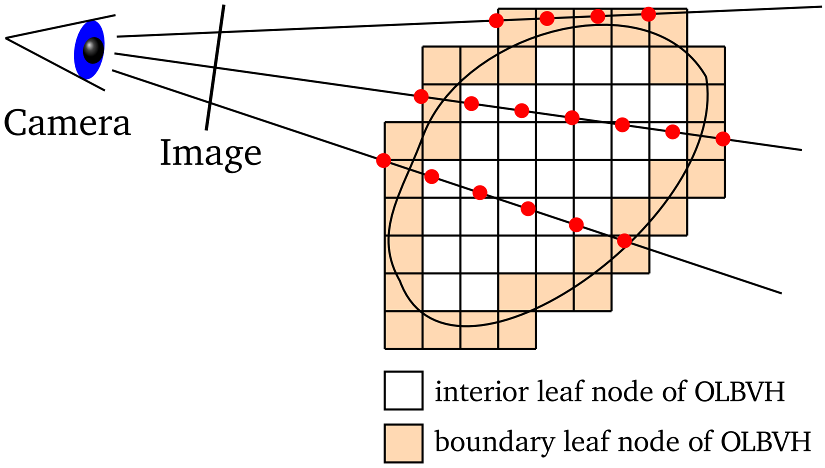


Figure 7.3.: Space skipping exploits the marking of boundary tree nodes computed in OLBVH construction.

Since the renderer emits one ray per pixel that is calculated using the method of Shirley et al. [SDH23], the computation of the interval of one ray is independent from the other rays. Thus, it is feasible to parallelize the interval computations over rays, in order to perform the pre-process at interactive rates, whenever the camera setup of the visualization changes. Therefore, the dimensions of the rendered image is a crucial factor for the workload for each frame.

The initial assumption of the empty space calculation is that the ray does not hit any part of the geometry. An invalid interval $[t^r_{\text{min}}, t^r_{\text{max}}] = [\infty, 0]$ with a value for the lower bound larger than the upper bound indicates empty space. This design choice should improve simplicity and performance of the pre-process, because no additional variables and branching are needed to express the case of empty space. In order to check the assumption of empty space, the pre-process tests if a ray intersects with the boundary nodes of the OLBVH. The key idea of the pre-process is motivated by the observation that a ray begins to intersect the object at a boundary face of the tetrahedral mesh and exits the object at another boundary face. These two boundary faces define the relevant interval and are by definition part of boundary tree nodes, which means that interior tree nodes never contain these boundary faces and can be skipped. While performing intersection tests for individual boundary faces is potentially too intrusive for a pre-process, a good estimation of the ray interval

Algorithm 11 Empty space skipping along view rays

```
procedure calculateRayInterval
    for each pixel in Image do                                          ▷ In parallel on the GPU
        $r \leftarrow$ getRayForPixel($pixel$)
        $[t^r_{\min}, t^r_{\max}] \leftarrow [\infty, 0]$                ▷ Initial assumption is empty space
        $F_N \leftarrow$ isBoundaryNode($N$) $\wedge$ intersectRayAABB($r, N$)    ▷ See [Wil+05]
        $F_P \leftarrow true$                                           ▷ No check on primitives necessary
        whenever an incident occurs during traverseOLBVH($F_N, F_P$) do  ▷ See line 29 in algorithm 10
            $[t^N_{\min}, t^N_{\max}] \leftarrow$ rayNodeInterval($r, N$)   ▷ Computed by [Wil+05]
            $[t^r_{\min}, t^r_{\max}] \leftarrow [min(t^r_{\min}, t^N_{\min}),\ max(t^r_{\max}, t^N_{\max})]$
        done
    end for
end procedure
```

without empty space is the first and last intersection with a boundary leaf node. The intersection test of a ray and an AABB can be efficiently and robustly implemented using the method of Williams et al. [Wil+05].

A short stack traversal as described in section 7.1.4 finds the intersecting boundary leaf nodes. Whenever the traversal reaches a boundary leaf node, the intersection test by Williams et al. [Wil+05] determines the interval $[t^N_{\min}, t^N_{\max}]$, along which the ray intersects the AABB of the leaf node. In order to compute the ray interval $[t^r_{\min}, t^r_{\max}]$ for the ray, the pre-process checks if the lower and upper bounds of $[t^N_{\min}, t^N_{\max}]$ identify a new minimum or maximum for the ray interval, respectively. The traversal terminates, when every intersecting boundary node is visited. The final result of the pre-process is a set of ray intervals, which bound the range for sampling along the view rays.

7.2.2. Sample the Simulation Results along View Rays

When the pre-process terminates, the ray marching performs point sampling along the view rays to compute the image. For the same reasons as described in section 7.2.1, the ray marching performs calculations in parallel over rays. As the CUDA model organizes GPU threads in grid blocks (cf. section 2.2.1), it is an interesting question how rays should be assigned to blocks to improve performance. As load balancing for ray tracing [HA98; CDE+13] and ray marching [MSE07] have been interesting research topics for decades, empirical testing evaluated the benefits of several static approaches to distribute the rays to grid blocks. These experiments statically assigned rays rowwise, columnwise and in tiles to the grid blocks. However, none of these methods consistently exhibited a significant advantage over the other. Therefore, the renderer applies the simple rowise pattern, whereas a dynamic approach for load balancing in ray calculation is an interesting topic for future work.

For ray marching, each GPU thread first retrieves the ray and the interval $[t^r_{\min}, t^r_{\max}] \subset \mathbb{R}$ computed by the pre-process. If the lower bound of the ray interval is larger than the upper bound, the ray does not hit the geometry and ray marching is not necessary. Otherwise, the ray marching methods prepares for sampling. In order to compute the number of samples for each ray, the bounds of the ray interval are quantized to integers. The quantization performs a division by the user specified sampling rate and a subsequent rounding. To obtain a conservative quantization the lower and upper bounds are rounded down and up, respectively. Subsequently, the ray marching begins to sample along the ray.

The ray sampling generates points along a view ray starting from the quantized interval's lower bound. The sampling then increments this initial position by the user-specified sampling rate until it reaches the interval's upper bound or the color for the pixel becomes almost opaque. For each sampling point, the ray marching traverses the OLBVH to find a tetrahedron containing the sampling point. As a result of the non-overlapping AABBs of the tree nodes, the traversal does not even need a stack or a short stack for this task.

If the primitive for traversal is a single spatial point, then it suffices to find one child node with an AABB containing the sampling point until traversal reaches a leaf node. The sampling point is either completely in the interior of the resulting node's AABB or lies on the boundary of up to eight leaf node AABBs. In both cases, it is sufficient to check the tetrahedra of only one leaf node, because every tree node includes the tetrahedra intersecting with its AABB by construction.

Whenever the traversal determined a leaf node containing the current sampling point, it checks all the tetrahedra associated with this leaf node for intersection with the sampling point. An intersection test can be implemented by calculating barycentric coordinates using Cramer's rule [Cra50]. If an intersection with a tetrahedron is found, the barycentric coordinates provide an interpolation of the scalar values associated with the vertices of the tetrahedron. The transfer function (cf. section 2.4) maps the interpolated scalar value to color. The mapped value is then composited over the current color value of the pixel. When ray marching terminates, each pixel in the rendering image is colored and the image is ready for visualization to the user.

Algorithm 12 Ray marching using OLBVH

```
procedure RAYMARCH(SamplingRate $\Delta t$)
    for each pixel in Image do                                    ▷ In parallel on the GPU
        Load the ray $r$ and $[t^r_{\text{min}}, t^r_{\text{max}}]$ calculated by algorithm 11
        if $t^r_{\text{max}} < t^r_{\text{min}}$ then
            continue
        end if
        $start \leftarrow floor(t^r_{\text{min}}/\Delta t)$
        $end \leftarrow ceil(t^r_{\text{max}}/\Delta t)$
        for $i \leftarrow start, \dots, end$ do
            $samplingPoint \leftarrow r.origin + i \cdot r.direction$
            $F_N \leftarrow$ POINTINAABB$(N, samplingPoint)$
            $F_P \leftarrow$ POINTINTETRAHEDRON$(P, samplingPoint)$
            obtain interpolated scalar value $s$ using TRAVERSEOLBVH$(F_N, F_P)$
            $col \leftarrow$ TRANSFERFUNCTION$(s)$                        ▷ See section 2.4
            $pixel.color \leftarrow$ COMPOSITOVER$(pixel.color, col)$     ▷ See section 2.4
            if $pixel.color.opacity > 99\%$ then
                continue
            end if
        end for
    end for
end procedure
```

7.3. Coarsening Meshes for Direct Volume Rendering

As coarsening meshes reduces memory requirements and rendering workloads, this section presents an algorithm using the collapsing method in chapter 5 that can reduce workloads for DVR with little loss of rendering quality. In order to achieve this goal, the customizable functions of the collapsing algorithm need to be specified for preserving rendering quality. In addition, this section describes an extension to the collapsing algorithm for maintaining the scalar data for rendering. For preserving the rendering quality, the coarsening method loads a scalar field Φ in addition to the tetrahedral mesh. Like in section 7.2.2, the scalar field Φ includes one value for each vertex. In order to prevent loss of important details, the used cost function measures the scalar field error incurred by an edge collapse operation:

$$\mathcal{C}(v_{\text{idx}_0}, v_{\text{idx}_1}, \Phi) = |\Phi(v_{\text{idx}_0}) - \Phi(v_{\text{idx}_1})|,$$

where v_{idx_0} and v_{idx_1} are the indices of the two edge vertices. In order to preserve features, the collapsing of edges should be limited to edges incurring only a low cost. Thus, collapsing is limited to edges with a cost value lower than

$$\varepsilon_{\Phi}(\chi) = \chi\left(\underset{v_{\text{idx}}=1,\ldots,N_{\text{V}}}{\arg\max}\;\Phi(v_{\text{idx}}) - \underset{v_{\text{idx}}=1,\ldots,N_{\text{V}}}{\arg\min}\;\Phi(v_{\text{idx}})\right),$$

where $\chi \in [0, 1]$ controls the loss of fidelity. The implementation consistently uses the middle point placement strategy, because the midpoint is an equidistant choice from both edge points resulting in a low scalar field error.

Collapsing edges relocates vertices to other spatial positions, invalidating the scalar field. Thus, the scalar values of relocated vertices need to be updated. Like Cignoni et al. [Cig+00], the coarsening for DVR interpolates the original mesh to approximate the scalar values of new vertices. As this essentially is a point lookup problem, the coarsening uses the acceleration structure applied for the DVR to interpolate scalar values of newly added vertices from the old scalar field. While any acceleration structure for efficient point lookup is applicable, this thesis uses the OLBVH. After each collapsing iteration, a parallel pass over the relocated mesh vertices performs a point lookup on the original mesh using the traversal described in section 7.2.2. For each of the relocated vertices, the lookup determines a tetrahedron including the vertex and calculates the barycentric coordinates for scalar field interpolation. The scalar value of the relocated vertex is then updated to the interpolated scalar value. In order to achieve robust barycentric interpolation, a safety tolerance of $\varepsilon_b = 1\mathrm{e}^{-3}$ is applied to the barycentric coordinates to avoid false results of the intersection test during the search for a tetrahedron containing the sampling point. This measurement has led to significantly better preservation of the scalar field compared to the results in the conference publication [SSF23] of the coarsening method.

7.4. Evaluation of Post-Processing Performance

In order to critically evaluate the massively parallel post-processing algorithms proposed in this chapter, a sequence of experiments demonstrates the performance of these algorithms and compares them with current methods. A discussion of the set of meshes for the evaluation appears in section 7.4.1. As a quick construction of a memory-efficient OLBVH is important for post-processing applications, section 7.4.2 evaluates and compares the performance of the OLBVH construction in terms of run time efficiency and memory consumption. In order to investigate the performance of ray marching for interactive DVR, section 7.4.3 investigates the throughput of samples in a comparative evaluation. For an in-depth analysis of ray marching performance, section 7.4.4 presents profiling results of DVR using OLBVH. Section 7.4.5 evaluates the performance of the massively parallel coarsening algorithm for DVR.

7.4.1. Evaluation Meshes

The evaluation of the proposed post-processing algorithms uses many different meshes, in order to include various mesh resolutions, tetrahedral element shapes, and surface geometries in the experiments. Figure 7.4 visualizes the different unstructured tetrahedral meshes used in the following evaluation. The meshes include simple geometries, e.g., cubic or cylindrical shapes, and shapes with complex surfaces, e.g., the Dragon or the Tardis geometries. The dimensions of the models also span a large variety, because the set of meshes includes models with even dimensions in each axis, e.g., the Die model, and thin models, e.g., the Wrench model. In order to cover different structures of primitives sizes, the test set includes models with different distribution of primitive sizes. Some meshes are rather regular such as the Jets model. Other meshes exhibit a uniform distribution of the primitive size, while the tetrahedral mesh has an irregular structure, e.g., the Part mesh. As many meshes for the FEM exhibit a fine-grained resolution at the boundary, while the interior structure is coarser, meshes of this type are also part of the data set, e.g., the Bunny or the Wrench models.

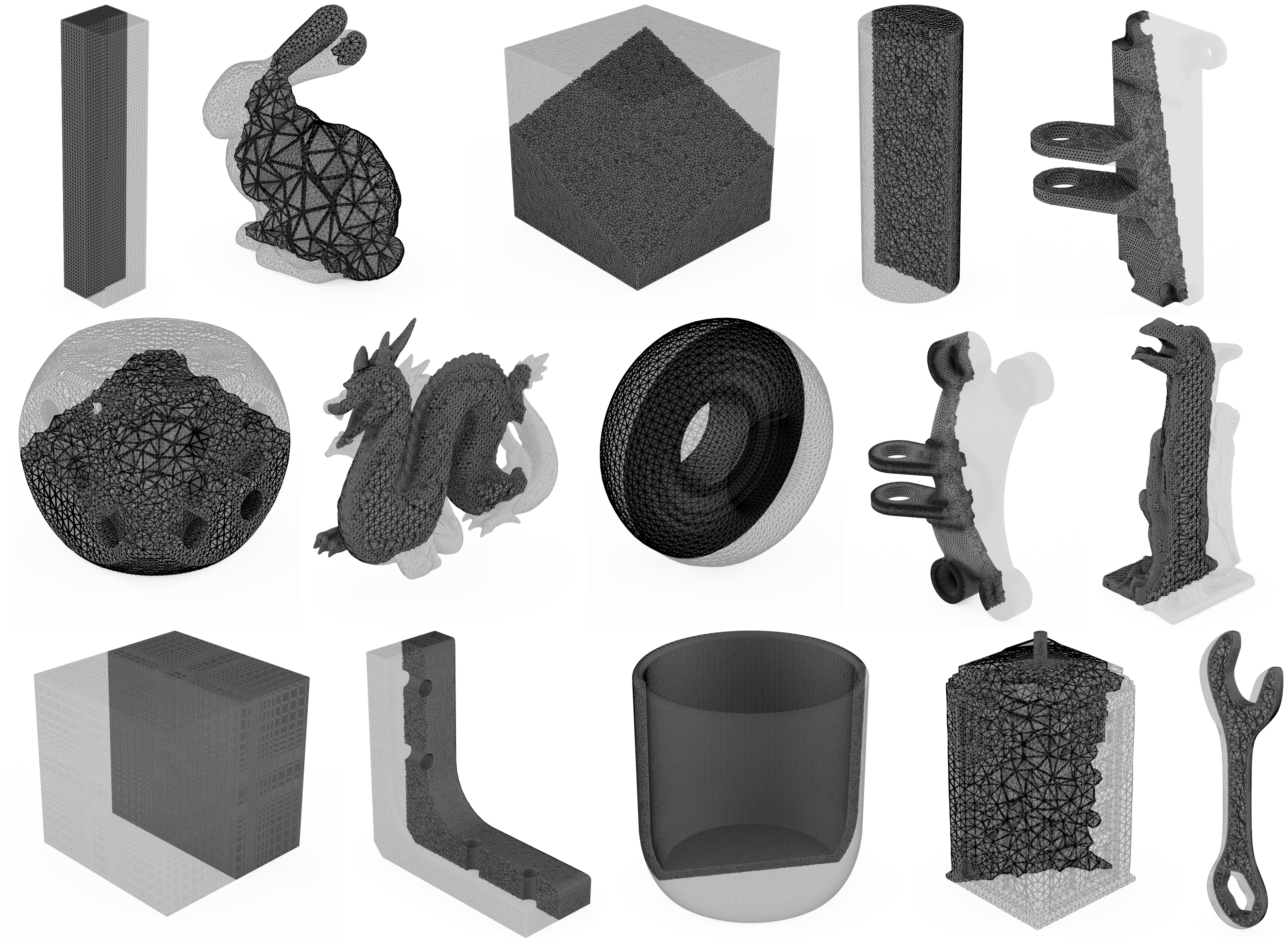


Figure 7.4.: Meshes used in the evaluation of post-processing. Surface overlaid over a cross section shown to visualize both exterior and interior resolution. From top left to bottom right: Bar, Bunny, Cube, Cylinder, SimJEB514, Die, Dragon, Fusion, SimJEB122, Gargoyle, Jets, Part, Pot, Tardis, and Wrench. The models include meshes with highly regular primitive size (e.g., Part) and meshes with large variance in primitive size (e.g., Tardis).

For an in-depth and critical view on the performance bottlenecks, section 7.4.4 presents profiling results of DVR using the OLBVH.

7.4.2. Performance of OLBVH Construction

This section evaluates the OLBVH construction in terms of run time and memory consumption. The evaluation compares OLBVH construction with Apetrei's [Ape14] improved LBVH construction and OptiX' [NVI24a] acceleration structure construction with subsequent compaction. The details of OptiX' acceleration structure are not documented and are considered a black box. The resulting construction times are shown in fig. 7.5. As tree depth is affected by the choice of l_α, we present construction times for $l_\alpha \in \{0, 1, 2\}$. For $l_\alpha = 0$ the OLBVH construction is slower than Apetrei's LBVH builder. For $l_\alpha \in \{1, 2\}$ the OLBVH construction outperforms or matches the construction times of the LBVH builder in many cases. If no hardware acceleration is used, OLBVH construction is faster than OptiX [NVI24a] on all but the smallest meshes (cf. table 7.1), even for $l_\alpha = 0$. When hardware acceleration is present, OLBVH construction is slightly slower than OptiX' BVH construction for $l_\alpha = 0$, but achieves significantly lower run times when $l_\alpha \in \{1, 2\}$.

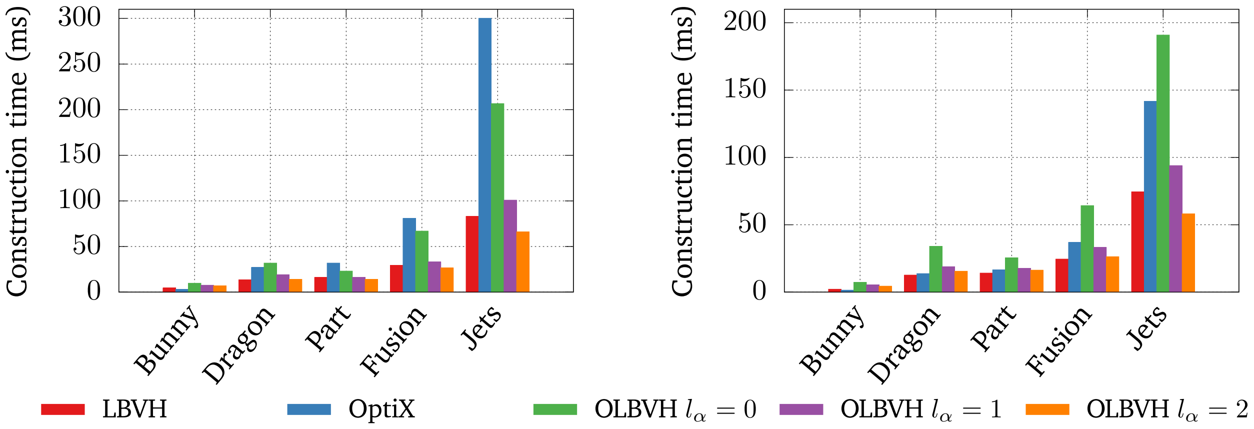


Figure 7.5.: Comparison of construction times between LBVH using Apetrei's [Ape14] fast agglomerative approach, OptiX, and OLBVH with $l_\alpha \in \{0, 1, 2\}$ on a Quadro GP 100 (left) and an RTX 2080 Ti (right).

Table 7.1.: Comparison of memory consumption (MiB) between LBVH, OptiX' compacted acceleration structure, and OLBVH for $l_\alpha \in \{0, 1, 2\}$. Even for $l_\alpha = 0$, OLBVH consumes less memory for most meshes due to the flatter octree hierarchy and compact offset-based encoding. The lowest results for allocated memory are provided in boldface.

Mesh	N_T	**LBVH**	**OptiX** [NVI24a]		**OLBVH** $l_\alpha =$		
			non-RTX	RTX	0	1	2
Bunny	32.0 k	2.07	10.4	1.94	1.83	0.57	**0.27**
Dragon	824.8 k	53.5	250	44.1	52.9	15.9	**7.41**
Part	1.1 M	71.3	313	54.9	28.4	12.3	**7.22**
Fusion	3.0 M	195	840	158	116	45.2	**24.6**
Jets	12.3 M	797	3427	399	433	164	**91.2**

Besides construction time, another beneficial aspect of the OLBVH is its memory consumption. Table 7.1 compares the memory consumption of OLBVH to LBVH and OptiX' acceleration structure. OLBVH consumes significantly less memory than LBVH or OptiX without specialized RTX hardware. For the Jets and Dragon meshes, OptiX [NVI24a] consumes slightly less memory than OLBVH with $l_\alpha = 0$ when RTX hardware is used. Increasing l_α results in a significant reduction of memory consumption due to generating shallow trees, i.e., fewer tree nodes. In a Bachelor thesis, González [Gon21] has shown that the memory consumption of OLBVH can be significantly reduced by creating subtrees of varying lenghts adaptive to the local mesh resolution.

7.4.3. Performance of Sampling Throughput for DVR

Since DVR relies on repeated point location along each view ray (cf. section 2.4), the run time performance of DVR highly depends on the throughput of the samples per time unit. Therefore, this section investigates the sampling rate Δt that represents the number of samples processed per second. For each mesh, the sampling rate Δt is set to the average edge length of the mesh. As a potential performance issue is that a large number of samples may not hit the geometry, the evaluation of DVR includes the space skipping presented in section 7.2.1. While many meshes were evaluated during this thesis, the evaluation discusses the performance of DVR on five meshes of varying shape and size.

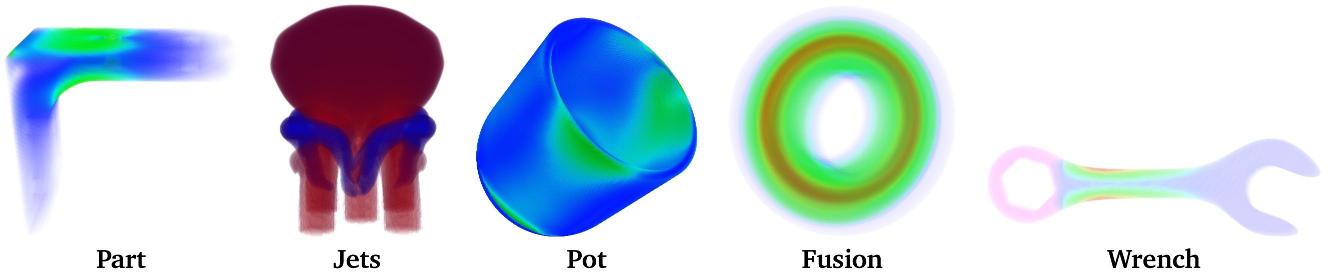


Figure 7.6.: Rendered images of test cases cropped to content.

The sizes of the evaluated meshes are given in table 7.2. The run times for the ray interval computation appear in table 7.2 as well. The evaluation compares the OLBVH DVR approach from section 7.2 with the LBVH as well as Wald et al. [Wal+19] approach with and without the RTX hardware accelerated spatial data structure (cf. section 3.8.3). For the comparisons, the evaluation investigates the other approaches using the same camera settings. In every measurement, a 1024^2 image was rendered. A visualization of the rendered images appears in fig. 7.6.

Table 7.2.: Sizes of DVR evaluation meshes and ray interval computation times (ms) on a Quadro GP 100.

Mesh	N_T	**Ray Interval** $l_\alpha =$ 0	1	2
Wrench	390.3 k	2.5	1.4	0.8
Part	1.1 M	3.1	1.6	0.8
Fusion	3.0 M	5.7	3.3	1.9
Pot	4.0 M	4.8	2.5	1.4
Jets	12.3 M	5.3	3.0	2.1

As the traversal for determining the relevant ray interval only considers boundary nodes and skips the majority of nodes, its run time imposes only a small overhead that is negligible compared to the actual sampling of view rays. Unlike Moriccal et al.'s [Mor+19] space skipping extension, the space skipping using OLBVH does not require a secondary acceleration structure. Therefore, the OLBVH space skipping does not add on the memory-consumption and construction overheads evaluated in section 7.4.2.

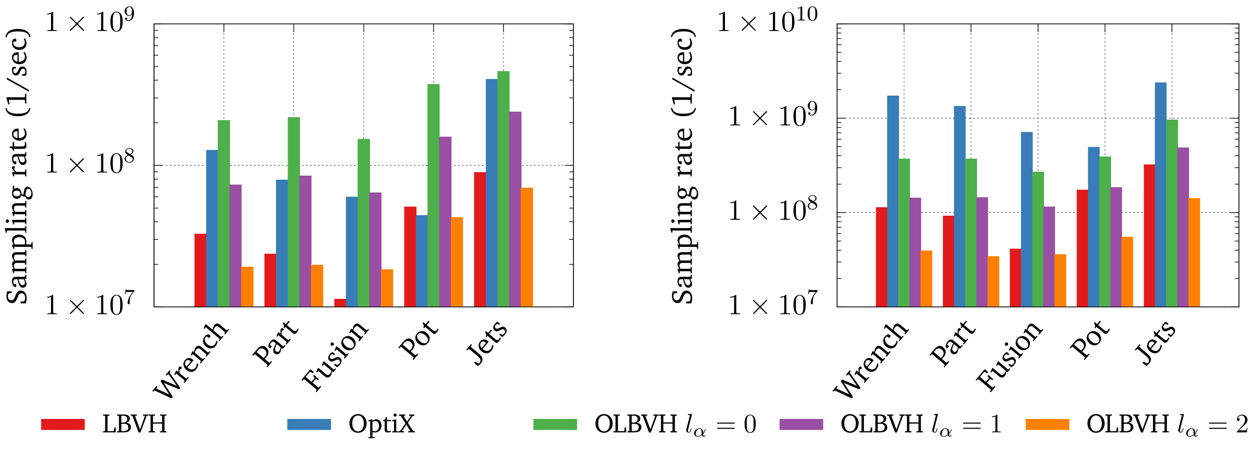
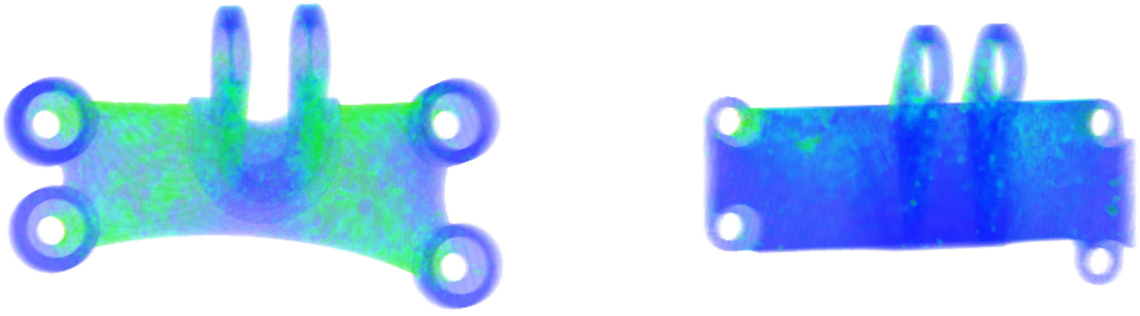


Figure 7.7.: Comparison of sampling rates per second (larger is better) using LBVH, Wald et al's method [Wal+19], and OLBVH with $l_\alpha \in \{0, 1, 2\}$ on a Quadro GP 100 (left, no RTX) and an RTX 2080 Ti (right, with RTX). The evaluation considers only samples inside the geometry.

Figure 7.7 shows the sampling rates per second for the different ray marching variants. If no specialized RTX hardware is used and $l_\alpha = 0$, the OLBVH approach outperforms the ray marching technique of Wald et al. [Wal+19] by up to 8.4× and the LBVH variant by 2×–13×. The Jets mesh benefits the least, which is due to the fact that it is geometrically a cube. Thus, ray marching cannot benefit from the prior space skipping using the OLBVH. Nonetheless, we observe a significant speedup. For the Fusion and Part meshes, it is even possible to choose $l_\alpha = 1$ and match the performance of Wald et al.'s method [Wal+19]. When the Pot mesh is rendered, the OLBVH approach achieves to match the sampling rate of Wald et al.'s method [Wal+19] with $l_\alpha = 2$. Therefore, an application could construct the OLBVH more quickly and use less memory while achieving the same performance. The use of the LBVH results in lower sampling rates than the other methods. Traversing the binary radix tree consumes significantly more time than using our method with $l_\alpha \in \{0, 1\}$. However, RTX hardware accelerated ray marching is faster than using the OLBVH approach. As only a minority of GPUs provides RTX hardware, the OLBVH approach is expected to be superior on the majority of GPUs.

7.4.4. Profiling DVR Performance

As the above sections evaluate the DVR only in terms of measured performance data, they do not give an in-depth insight into the actual bottlenecks of the method. Therefore, a joint project with Buelow et al. [Bue+24] aimed at the development of a visual profiling tool specifically for DVR. While the colleagues focused on the development of the profiling tool, the author of this PhD thesis prepared the OLBVH DVR implementation for using the profiling tool and interpreted the results. For preparation, each allocated array of interest receives a unique identifier $id \in \mathbb{N}$, which is handed over together with the memory address to a shared library that maintains data throughout the profiling session. This project has enabled interesting performance insights. The remainder of this section firstly gives an overview on the recorded performance data and then discusses the profiling results in detail.



(a) SimJEB122

(b) SimJEB514

Figure 7.8.: Render images of the used SimJEB [WBM21] models.

As the access to global GPU memory is slow (cf. section 2.2.1), the profiling tool counts the number N_{req} of requests to global memory. This counter is recorded for each pixel $N_{\text{Req}(x,y)}$ of the rendered image, i.e., for each view ray. In this way, the profiler illuminates, which rays and which parts of the geometry produce the peak of requests to global memory. Another important metric for performance is the branching b of a thread, as branching reduces the speedup obtained by parallel execution. For this reason, the profiler records the branching $b_{(x,y)}$ for each view ray. Furthermore, the caching of the arrays for traversal and access to primitives as well as the mesh can also illuminate bottlenecks due to incoherent access in memory. Therefore, the profiler records the cache hit rates $p(id)$ for a specified array of GPU memory with identifier $id \in \mathbb{N}$. Since the collection of performance data for each ray enables in-depth analysis, the cache hit rates $p(id)_{(x,y)}$ are also recorded for each pixel (x, y). As the recorded performance characteristics for one render image can be mapped to the interval $[0, 1]$, it is convenient to present the performance data in images, where each pixel

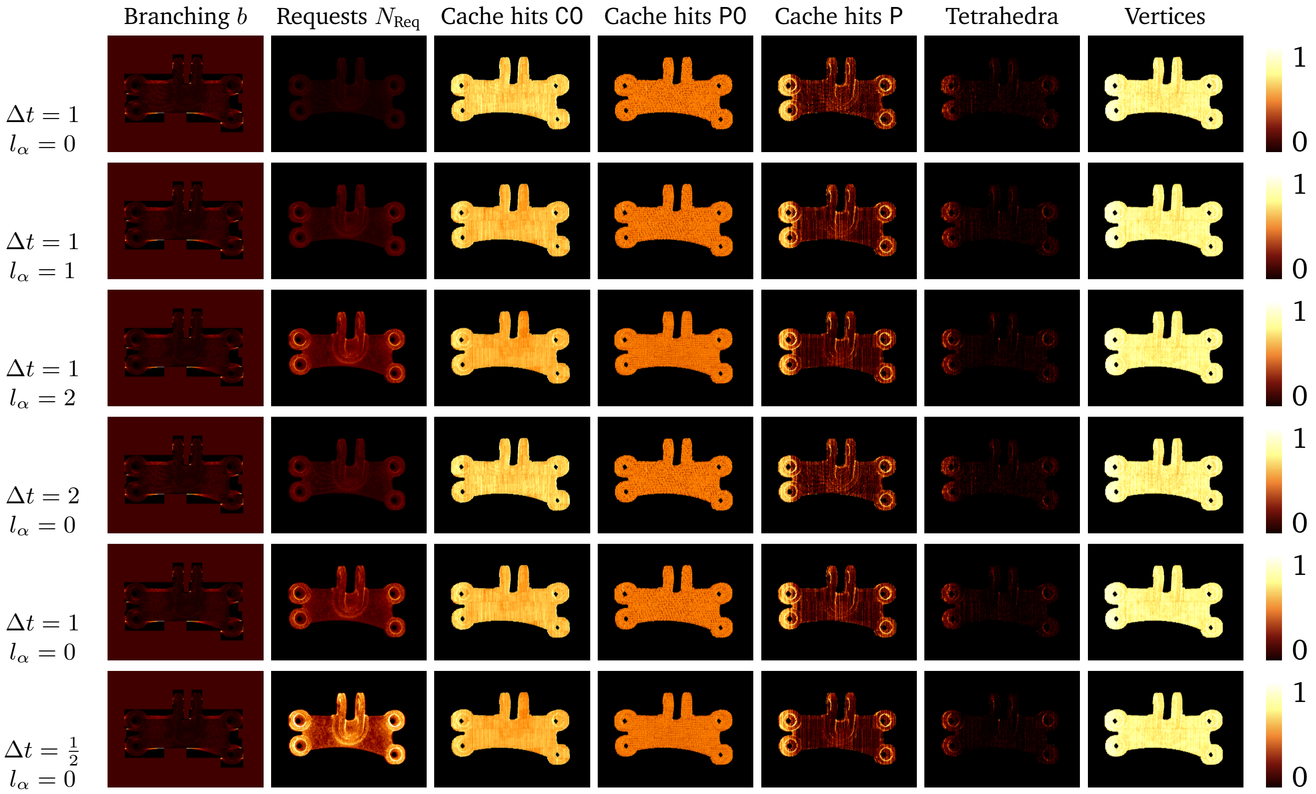


Figure 7.9.: Results for profiling the SimJeb122 model visualizing the branching rate $b_{(x,y)}$, numbers of requests $N_{\mathrm{Req}(x,y)}$ normalized by 2^{15}, and cache hit rates $p(id)_{(x,y)}$ for individual allocations id.

(x, y) represents the recorded data with a color value interpolated along a color scale. Following the example of flame graphs [Gre20], the color scale of use includes all the colors of fire to produce “flame images”.

In order to profile DVR performance for a real world scenario, the profiling evaluation used two meshes from the SimJeb dataset [WBM21] that includes optimized mechanical brackets from an engineering design competition. While we tested the profiling toolkit on many models, the evaluation showcases the results for the SimJEB514 with many sharp features and the SimJEB122 with many rounded features (see fig. 7.8). To determine the effects of different parameter setups, the evaluation covers rendering the models with different sampling rates and tree depths l_α. For the sampling rate, the average edge length of a mesh serves as the reference sampling rate that is factored by $\Delta t \in \{2, 1, \frac{1}{2}\}$. The experiments all use the regular sampling algorithm introduced in section 7.2.2. For the tree depth, the profiling covers the cases $l_\alpha \in \{0, 1, 2\}$. The profiling determines the cache hit rates for particular arrays related to traversal, in order to provide an in-depth analysis. For traversal, the most relevant arrays are the children offsets C0, primitive offsets P0 , and the primitive indices P. In addition, the profiling also evaluates the caching of the tetrahedra and vertices of the mesh to investigate which parts of the geometry receive the best cache hit rates. Figure 7.9 and fig. 7.10 showcase the profiling results for rendering the SimJEB122 and SimJEB514 models, respectively.

Most branching occurs for the rays intersecting only with the silhouette boundary of the model. Thus, these rays reduce the occupancy of massively parallel execution. Using shallower trees or finer sampling rates does not lead to significant differences in regards to branching behavior. On the contrary, the number of global accesses significantly increases when increasing l_α or reducing Δt. Global access numbers are especially large at the curved features of the models, where a tetrahedral meshing tool typically uses a finer resolution to correctly represent the features with sufficient element quality for simulation.

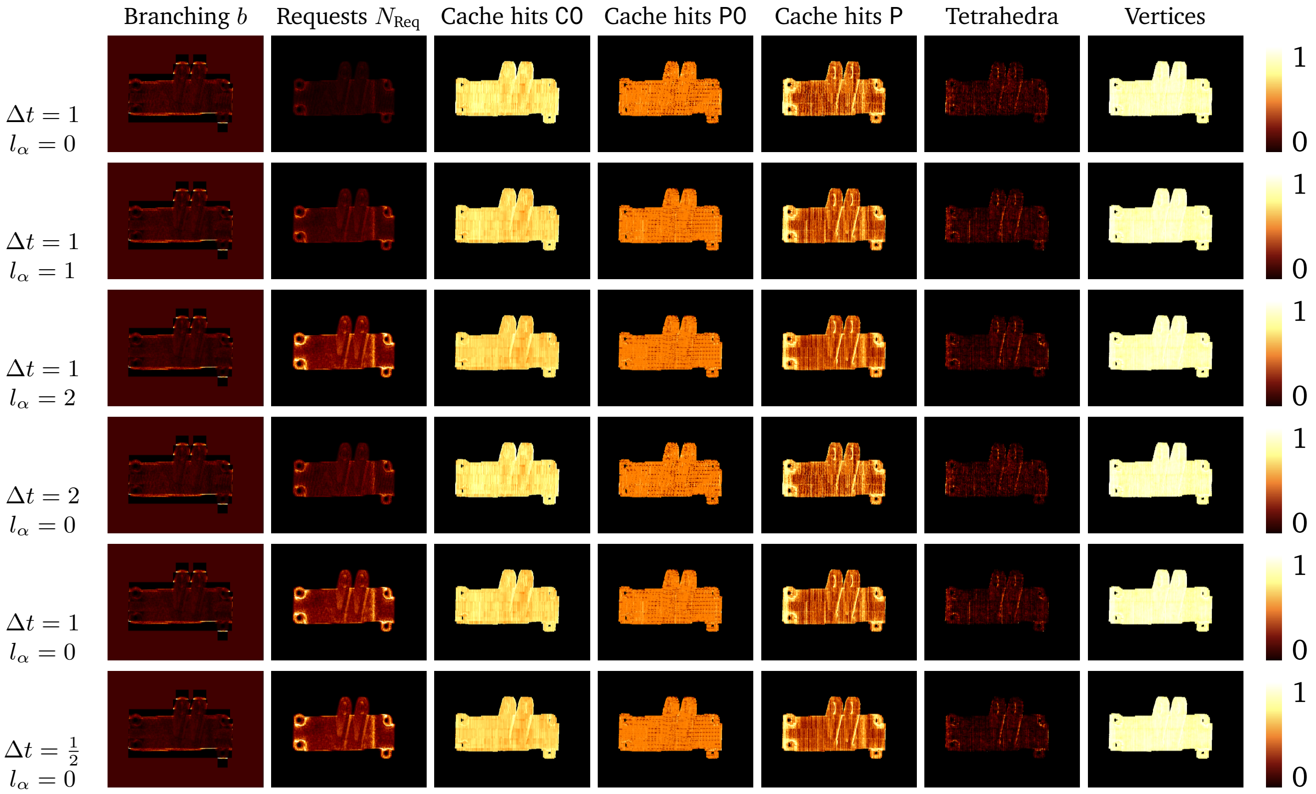


Figure 7.10.: Results for profiling the SimJeb514 model visualizing the branching rate $b_{(x,y)}$, numbers of requests $N_{\text{Req}(x,y)}$ normalized by 2^{15}, and cache hit rates $p(id)_{(x,y)}$ for individual allocations id.

Cache hit rates for the children offsets (CO) slightly increase for a shallower tree and a finer sampling rate, because traversal more often terminates in the same or memory-adjacent leaf nodes. However, hit rates for primitive offsets (PO) are equally distributed and do not significantly change for altering l_α or Δt. A more varying cache hit rate occurs for the primitive indices (P), while changing l_α or Δt does not exhibit significant influence. The cache hit rate for primitive indices is more governed by the scalar data and the structure of the model. As scalar values are low at the round holes, these regions are rendered with high transparency leading the DVR to spend more sampling points on the corresponding rays. Due to the round structure of the holes, the corresponding tree nodes contain more empty space and are associated with fewer tetrahedra, which leads to better cache hit rates.

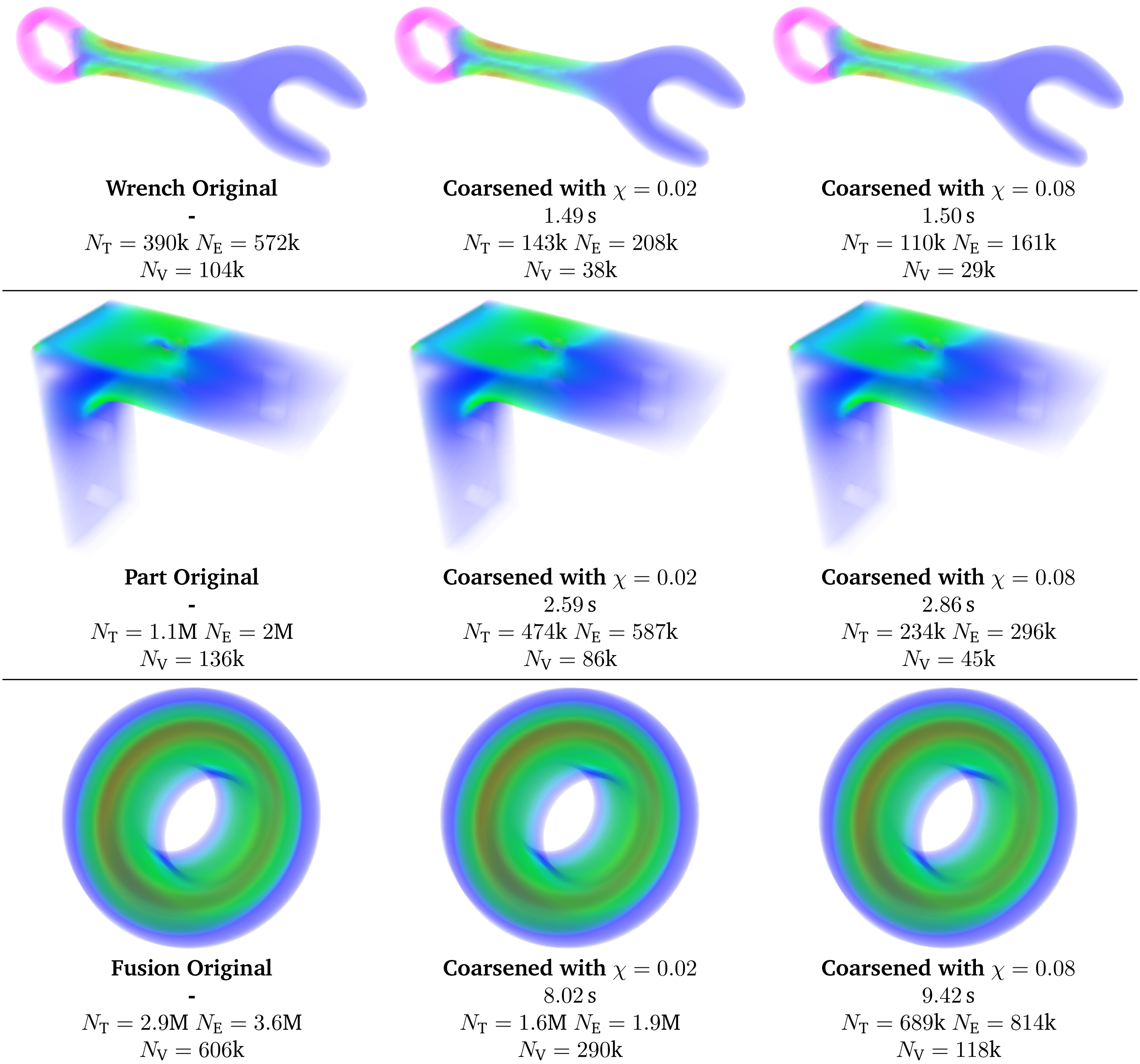
One vertical line in the right part of the images of the SimJEB514 model indicates a large number of accesses. The mesh resolution is finer along this vertical line leading to more memory accesses. Our visual representation shows that cache hit rates of the P array also increase at these positions. As more primitives are located along this line, the cache contains more primitive indices from P and the probability for a cache hit increases. Increasing l_α or Δt leads to a better cache hit rate at the thin structures near the boundary and the vertical line compared to the remainder of the model. The distribution of cache hit rates for vertices is uniform for all choices of l_α or Δt for rendering both models.

Reflecting on the profiling results, important insights about the performance of DVR have been gathered. As expected, the number of memory accesses increases for finer sampling of shallower trees. Especially, parts of the mesh with finer resolution receive more memory requests, because more primitives need to be checked after traversal. While these observations match common expectation, other insights are rather difficult to predict intuitively. One of these insights is that cache hit rates do not necessarily improve by using a finer sampling rate or the number of memory accesses in general. Even though more sample look ups take the same path during tree traversal when using finer sampling, the caching cannot compensate the more frequent memory requests. Another unexpected insight is that curved features lead to many memory requests, because the finer mesh resolution accounts for more candidates during the search for a containing tetrahedron. This is related to another insight, which reveals that peak branching occurs at the silhouette of the geometry. These insights suggest sampling adaptive to the local deviation of the scalar field and geometry features, e.g, mesh resolution or boundary structures. Avenues to improve DVR performance are considered as future work.

7.4.5. Performance of Coarsening for DVR

Since this thesis proposes a massively parallel algorithm for coarsening unstructured tetrahedral meshes for DVR in section 7.3, the evaluation also comprises this algorithm. The evaluation demonstrates the resulting DVR images on the Wrench, Part and Fusion meshes in fig. 7.11. In order to avoid many collapsing iterations with little effect (cf. section 5.4.4), the experiments run with a specified collapsing threshold of 200, 500, and 700 for the Wrench, Part, and Fusion, respectively. As different choices for χ impose different workloads, the evaluation sets $\chi = 0.02$ or $\chi = 0.08$ for the experiments. The median run times for the different meshes and choices for χ are measured to investigate the performance. The mesh sizes in terms of N_{T}, N_{E}, and N_{V} are determined in the evaluation to show the process of coarsening. The evaluation hardware setup is equipped with an NVIDIA RTX 3090 GPU and an Intel i9-11900K CPU.

The rendering images of the coarsened meshes in fig. 7.11 exhibit only little deviation from the rendered images of the original meshes. Due to favoring edge collapses with little cost, smooth transitions of color are well preserved. However, the proposed coarsening method can smooth sharp changes in the scalar field, e.g., see the sharp change from magenta to blue in the rendering image of the Wrench. While the rendering images are well preserved, the proposed coarsening method achieves to substantially reduce the size of the meshes. The evaluation shows that a small choice for χ such as $\chi = 0.02$ already leads to a substantial reduction of the mesh size. For $\chi = 0.02$ the proposed coarsening algorithm produces meshes of half the size and for $\chi = 0.02$ the algorithm produces meshes of even a quarter of the size. Therefore, many edges represent only small transitions in the scalar values, while only few edges represent the sharp and fine grained transitions in scalar values. The run time performance of the proposed methods exhibits run times of a few seconds for coarsening the meshes. Thus, run time performance of the proposed coarsening method for DVR does not enable coarsening at interactive rates and the coarsening of large unstructured tetrahedral meshes with millions of elements should be executed as a pre-process for rendering static models. It is notable that the coarsening for $\chi = 0.02$ can take several seconds, while the difference in run time between $\chi = 0.02$ and $\chi = 0.08$ is typically much smaller than several seconds. As the re-building of TCSR mesh is a dominant overhead (cf. section 5.4.3), this is an expected observation, because the mesh is becoming smaller throughout coarsening and smaller meshes require less time for determining the connectivity relationships. Throughout the evaluation of this thesis, experiments have investigated, if the coarsening of the meshes has an effect on DVR performance. The run time performance of DVR does not change significantly, as the sampling rate Δt is the dominant factor for the run time performance and depends on the scalar field. Therefore, one needs to specify the same sampling rate despite coarsening to obtain a detailed visualization of the simulation results. Nonetheless, the performance in terms of memory consumption improves due to coarsening, because the tree depth of the spatial data structure is governed by the resolution of the mesh. After coarsening the mesh, less tree nodes are required to achieve the point lookup with similar run time performance.



Wrench Original
-
$N_\mathrm{T} = 390\mathrm{k}$ $N_\mathrm{E} = 572\mathrm{k}$
$N_\mathrm{V} = 104\mathrm{k}$

Coarsened with $\chi = 0.02$
1.49 s
$N_\mathrm{T} = 143\mathrm{k}$ $N_\mathrm{E} = 208\mathrm{k}$
$N_\mathrm{V} = 38\mathrm{k}$

Coarsened with $\chi = 0.08$
1.50 s
$N_\mathrm{T} = 110\mathrm{k}$ $N_\mathrm{E} = 161\mathrm{k}$
$N_\mathrm{V} = 29\mathrm{k}$

Part Original
-
$N_\mathrm{T} = 1.1\mathrm{M}$ $N_\mathrm{E} = 2\mathrm{M}$
$N_\mathrm{V} = 136\mathrm{k}$

Coarsened with $\chi = 0.02$
2.59 s
$N_\mathrm{T} = 474\mathrm{k}$ $N_\mathrm{E} = 587\mathrm{k}$
$N_\mathrm{V} = 86\mathrm{k}$

Coarsened with $\chi = 0.08$
2.86 s
$N_\mathrm{T} = 234\mathrm{k}$ $N_\mathrm{E} = 296\mathrm{k}$
$N_\mathrm{V} = 45\mathrm{k}$

Fusion Original
-
$N_\mathrm{T} = 2.9\mathrm{M}$ $N_\mathrm{E} = 3.6\mathrm{M}$
$N_\mathrm{V} = 606\mathrm{k}$

Coarsened with $\chi = 0.02$
8.02 s
$N_\mathrm{T} = 1.6\mathrm{M}$ $N_\mathrm{E} = 1.9\mathrm{M}$
$N_\mathrm{V} = 290\mathrm{k}$

Coarsened with $\chi = 0.08$
9.42 s
$N_\mathrm{T} = 689\mathrm{k}$ $N_\mathrm{E} = 814\mathrm{k}$
$N_\mathrm{V} = 118\mathrm{k}$

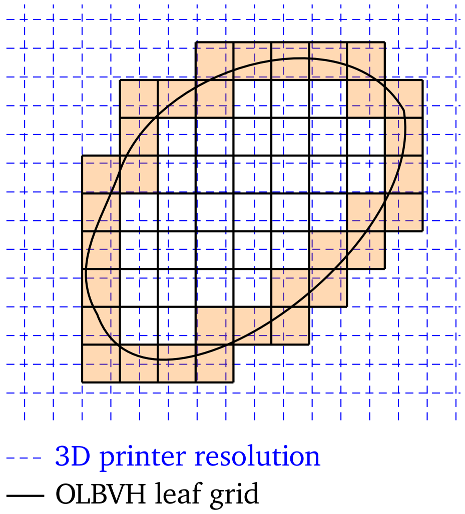
Figure 7.11.: Render images for coarsened versions of the Wrench, Part, and Fusion meshes. The collapse threshold ε_c was set to 200, 500, and 700 for the Wrench, Part, and Fusion, respectively.

7.5. Conservative Slicing

As an addition to memory-efficient DVR of unstructured tetrahedral meshes, this thesis presents a conservative slicing method to interpolate material/scalar parameters for a sampling point inside the input mesh using the containing tetrahedron. In case of a sampling point outside the mesh, the method extrapolates using the closest tetrahedron intersecting the conservative AABB. Section 7.5.1 presents an algorithm for conservative slicing and section 7.5.2 evaluates its run time performance.

7.5.1. Algorithm for Conservative Slicing using the OLBVH

Slicing is a technique in 3D printing, where the to-be-printed model is sampled in slices of 2D grids. A slicing algorithm checks whether a sampling point of a slice is inside or outside the input geometry. Typically, 3D printing relies on surface meshes. However, printing volumetric objects of multiple materials [Zas20] can benefit from using unstructured tetrahedral meshes. Especially, for high-resolution tetrahedral meshes, the slicing algorithm requires a spatial data structure for intersection tests and lookup of the tetrahedron containing a sampling point. Therefore it is interesting to explore, whether the OLBVH can be used for quick spatial search to achieve fast slicing of unstructured tetrahedral meshes.



A visualization of one slicing plane (blue) and the cross section of a model as well as the OLBVH appears in the inset. For each point in the blue grid, the model is sampled. If the point intersects with the to-be-printed model, the 3D printer fills this local area with material. In this way, a 3D prototype can be printed from the digital geometry connecting the sampled points. Thus, slicing of volumetric meshes with volumetrically varying materials, see e.g., Altenhofen et al. [Alt+18], is like DVR a point location problem.

As the resolution of the 3D printer limits the resolution of geometric details for printing, it can be challenging to produce prototypes of high-resolution geometric details. The idea of conservative slicing is to produce a prototype with a slightly larger shape so that the intended prototype can be carved out of the printed model. As post-processing of 3D printed models using conventional CAM or by polishing can only remove material, slicing typically has to be performed conservatively, i.e., material should be represented as a voxel of the printing volume even if only the AABB of the voxel intersects the geometry and not the voxel centroid itself. Therefore, if a tetrahedron intersects the AABB of the voxel, the sampling point is valid. Using the resulting conservative prototype, the development team is no longer limited to the resolution of the 3D printer at hand, because it can carve fine-grained geometric details out of the conservative prototype.

The following describes an efficient conservative slicing algorithm using OLBVH. This algorithm initially traverses the OLBVH for a sampling point, attempting to find a containing tetrahedron. If such tetrahedron is found, the slicing method computes the barycentric coordinates to interpolate the material properties. Due to tightly fitting AABBs, it is sufficient to find the single leaf node that contains the sampling point (cf. section 7.2.2). If no such tetrahedron can be found, the sampling point is potentially near the boundary. In this case, a second and final traversal phase constructs a conservative AABB around the sampling point. In order to switch between the two traversal phases, the slicing method utilizes the efficient re-initialization feature of the short stack [VWB19]. The second phase checks for intersections between the conservative AABB for the sampling point and the AABBs of tree nodes. This traversal pass only considers boundary nodes to decrease the number of relevant tree nodes. During traversal, the slicing method pushes each intersecting leaf node to a queue. For all the leaf nodes in the queue, the slicing method tests for intersection with the conservative AABB and the tetrahedra of the node using the intersection test by Ratschek and Rokne [RR97]. Whenever an intersection occurs, the slicing method calculates the barycentric coordinates of the grid point for the tetrahedron. As the barycentric coordinates indicate the closest vertex of the tetrahedron to the grid point, the slicing method calculates the distance of the point to the tetrahedron vertex. The slicing method approximately determines the closest tetrahedron and extrapolates the material properties using the barycentric coordinates.

7.5.2. Runtime Performance of Conservative Slicing

As the conservative slicing algorithm should achieve fast run time performance, the evaluation discusses run time measurements of the proposed algorithm using OLBVH and LBVH. The evaluation uses a 999^2 grid using planes and slice thicknesses normal to the slicing plane as they appear in table 7.3. For each slice, the origin of the plane is the mesh AABB's midpoint. The LBVH variant traverses the tree for the voxel AABB finding the closest point in the mesh to the sampling point. As Wald et al's [Wal+19] approach only applies to point sampling of unstructured tetrahedral meshes, it is not applicable to conservative slicing. Like the evaluations above, the evaluation of conservative slicing uses the Quadro GP 100 and RTX 2080 Ti GPUs. In order to investigate how the run times of conservative slicing scale, the evaluation includes varying mesh sizes. The use of geometries with different surface structures reveals the bottlenecks of the algorithm.

Table 7.3.: Mesh sizes and slicing setups for conservative slicing where the origin is the mesh AABB's midpoint. The slice thickness is the voxel size normal to the slicing plane.

Mesh	Tetrahedra	AABB of mesh	Slice	
			Plane	Thickness
Bunny	32.0 k	1×0.99×0.77	xy	0.05
Cylinder	172.4 k	0.52×1.4×0.52	xz	0.025
Cube	1.5 M	2×2×2	xy	0.05
Fusion	3.0 M	4.9×4.9×2.9	xy	0.1
Jets	12.3 M	127×127×127	xy	1

Figure 7.12 presents the median run times for conservative slicing using LBVH and OLBVH with $l_\alpha \in \{0, 1, 2\}$. As the number of samples is equal for every geometry, the resolution of the mesh is not the predominant factor that governs the run time. One important factor is the thickness of the slicing plane. For instance, the test case for the Cube scenario allows for 40 slicing planes, while the slicing of the Jets model allows for more than 100 slicing planes. Thus, the relative thickness is larger for slicing the Cube than for the slicing the Jets, which leads to slower run time performance for the Cube, although the Jets mesh consists of significantly more tetrahedra and the boundaries of both meshes are cubical. Since a thicker slicing plane typically leads to more potentially intersecting primitives, the traversal needs to check more tree nodes, which increases the run time. Another important factor for run time performance is the boundary structure of the mesh. The boundary of the Fusion model is a torus. Therefore, for many sampling points, the second traversal phase needs to determine barycentric coordinates to perform extrapolation. Since both spatial data structures rely on AABBs as hull volumes, a rounded surface structure is enclosed by several small AABBs that trigger more expensive narrow phase intersection tests of tetrahedron to box. This increases run times significantly.

For every experiment, the run time performance of conservative smoothing provides fast results. On both

GPUs, each slicing method exhibits fast performance for a single slice with run times of a few milliseconds. The approach using OLBVH outperforms LBVH for $l_\alpha \in \{0, 1\}$ with speedups between 3× and 25×. As a result, the OLBVH allows for substantial acceleration of conservative slicing, while 3D printing applications for producing exact prototypes can benefit from fast, memory-efficient construction on GPUs.

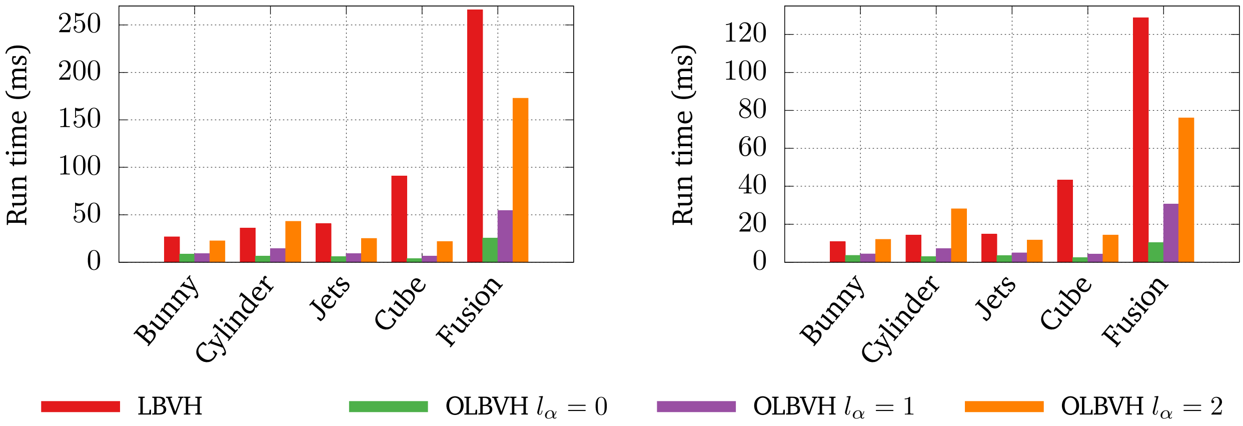


Figure 7.12.: Comparison of slicing times for a single slice using LBVH and our approach for $l_\alpha \in \{0, 1, 2\}$ on a Quadro GP 100 (left) and an RTX 2080 Ti (right).

7.6. Summary

In summary, this chapter has presented the memory-efficient OLBVH spatial data structure along with algorithms for memory-efficient DVR and conservative 3D printing. The OLBVH does not depend on the RTX platform (cf. section 3.8.3) and is specifically for volumetric meshes. The data layout of the OLBVH has enabled sparser memory usage compared to other spatial data structure that enable GPU-parallel construction. Further reduction of memory consumption has been achieved by reducing the tree depth at the cost of reduced run time performance due to larger sets of primitives in OLBVH leaf nodes. The evaluation results have revealed that the construction of the OLBVH terminates within a fraction of a second for meshes consisting of millions of tetrahedra.

The performance of the proposed DVR algorithm is comparable to current methods on GPUs without RTX hardware acceleration. This applies to the majority of GPUs. Due to neat exploitation of the OLBVH's boundary tree node marking, an efficient space skipping method is able to quickly shorten the relevant interval for each ray. In-depth profiling of the proposed DVR algorithm has revealed important bottlenecks: peaks in memory requests at locally more high-resolution mesh parts, high branching at the silhouette of the mesh, and many accesses to global memory for fine sampling rates.

For further reduction of memory consumption, this chapter has presented a massively parallel algorithm for coarsening unstructured tetrahedral meshes respecting the scalar field of simulation results. This algorithm relies on the conflict detection presented in chapter 5. The evaluation of the algorithm has shown that high-resolution unstructured tetrahedral meshes can be reduced to only a quarter of the initial size while well-preserving many details of the rendered images. However, the algorithm can smooth sharp transitions in the scalar field of simulation results. Despite the usage of the GPU for edge collapsing, the simplification of tetrahedral meshes until no more edges can be collapsed remains too expensive for real-time performance, because measured run times amount to several seconds. Only few iterations of edge collapsing, e.g., to remove ill-shaped tetrahedra, can be done within hundreds of milliseconds.

In order to extend the presented massively parallel algorithms for VP by a bridge to physical prototypes, this chapter has introduced a method for conservative slicing, which enables the 3D printing of a conservative physical prototype. By carving surplus material more high-resolution prototypes can be produced than the resolution of the 3D printer allows for. The usage of the OLBVH has shown significant acceleration for conservative slicing compared to LBVH.

Returning to RQ4[1], this chapter has shown that the memory efficiency of post-processing can be significantly improved for rendering unstructured tetrahedral meshes. This applies to a multitude of GPUs, because the presented algorithms do not depend on specific hardware acceleration such as the RTX platform. Compared to hardware accelerated post-processing, the primary costs of the improved memory efficiency is the overhead of prior coarsening, the use of a slightly more expensive construction for the spatial data structure, and reduced run time performance of rendering. Nonetheless, good run time performance of DVR could be achieved, which enables interactive rendering on large models with millions of tetrahedra. It is worth noting that the profiling has revealed potential for improving the performance of DVR with OLBVH by, e.g., implementing adaptive sampling or load balancing to reduce branching.

[1] RQ4: *How can the massively parallel post-processing of unstructured tetrahedral meshes be implemented with efficient memory usage?*

8. Conclusion and Future Work

As a key result, this thesis has facilitated the use of GPUs, which enables faster virtual prototyping (VP) for VP processes that involve unstructured tetrahedral meshes. The evaluation of this PhD-project has revealed that VP processes using high-resolution unstructured tetrahedral meshes include many computationally expensive tasks. Using the methods presented in this thesis, people can achieve substantial speedups for these tasks. Therefore, the research for this PhD thesis has enabled significant acceleration of many important geometry processing applications.

Since the research for this PhD thesis is primarily concerned with the RQs stated in section 1.3, the research results enable to formulate answers to the initial RQs in section 8.1. As the answers to the RQs impact many geometry processing applications, the conclusion in section 8.2 reflects on the key findings of this PhD thesis and describes their most relevant impacts on industry and science. In order to provide a critical view on this PhD thesis, section 8.3 describes important limitations of the proposed methods. Finally, section 8.4 provides avenues for future work based on this PhD thesis.

8.1. Answering the Research Questions

After investigating the RQs formulated in section 1.3, the research results of this PhD thesis provides answers to these RQs. To return to these RQs, this section revisits the initial RQs and provides a concise research answer (RA) to each of the RQs:

RQ1: *How can the use of the GPU accelerate mesh optimization and re-meshing tasks for unstructured tetrahedral meshes?*

→ ***RA1:*** *The efficient use of the GPU requires dense sets of independent sub-meshes. Methods for massively parallel conflict detection can quickly determine large sets of conflict-free sub-meshes of high-resolution tetrahedral meshes. For massively parallel edge/face flips, conflict detection can to find the locally most beneficial flip for a convex set of tetrahedra, which results in speedups of 133× - 254×. For massively parallel cavity-based re-meshing, conflict detection can use the mesh topology to find independent sub-meshes around edges in only two parallel passes. Using the resulting sub-meshes, enables significant acceleration of complex re-meshing tasks. For instance, detecting dense sub-meshes for coarsening unstructured tetrahedral meshes exhibits speedups of up to 34× compared to CPU-sequential re-meshing, of up to 7.4× compared to CPU-parallel re-meshing, and of up to 2.7× compared to state-of-the-art GPU-parallel re-meshing. A dense set of sub-meshes enables fast convergence of re-meshing, because many tetrahedra, i.e., cavities, can be processed in each parallel pass. The convergence of the overall optimization should require few iterations for convergence. Massively parallel optimization and re-meshing should rely primarily on the connectivity of the mesh avoiding auxiliary data structures. Due to following these principles, the massively parallel harmonic mesh optimization algorithm using vertex relocation and edge/face flips achieves speedups of 10× - 84× for precise boundary preservation and 3.77× - 60× for approximate boundary preservation.*

RQ2: *How can massively parallel optimization and re-meshing of unstructured tetrahedral meshes robustly produce meshes of sufficient quality for numerical simulations?*

→ ***RA2:*** *The conflict detection used for massively parallel processing needs to find non-overlapping sub-meshes so that the resulting meshes are valid. For optimization of vertex positions, the parallel relocation of vertices should perform a gradient descent of one vertex while the adjacent vertices are fixed, because this prevents inverted elements and conflicting gradient descent steps. Re-meshing operations should be prioritized by their improvement of the to-be-optimized functional for robust convergence. Each operation to change the geometry or connectivity of an unstructured tetrahedral mesh should check for inverted elements and degradation of the to-be-optimized functional. An optimization or re-meshing algorithm should prevent to perform operations that lead to inverted elements or worsen the to-be-optimized functional. The boundary treatment of the massively parallel optimization algorithm should carefully respect minimization of the to-be-optimized functional, which can be achieved with the use of directional derivatives along the boundary. In order to preserve the boundary of the geometry, a detection of the surface features is required. A robust and massively parallel detection of surface features can be achieved by using the topology of the mesh as well as the surface triangle normals. By following these considerations, the proposed optimization and re-meshing algorithms produce valid unstructured tetrahedral meshes on more than* 10 k *test meshes [Hu+20].*

RQ3: *How can massively parallel mesh processing be used for quick editing of unstructured tetrahedral meshes to shorten VP cycles?*

→ ***RA3:*** *Massively parallel mesh editing operations on the basis of face groups and cages for deformation control can be implemented to avoid returning to CAD, but these editing operations can degrade tetrahedral element quality. For a seamless transition from CAD, face groups can be assigned from tags that originate from CAD. If semantic face tags are not available or not suitable for the use case, face groups can also be determined with the proposed massively parallel algorithm, which immediately finds a set of face groups with a run time performance in a few milliseconds for meshes consisting of several ten thousands of triangles. In order to facilitate editing based on face groups, users can control the resulting set of face groups detected by the proposed algorithm with a user-defined angle threshold. Mesh editing algorithms based on face groups can achieve sufficient run time performance for immediate results below one second, which has been shown for two editing operations: hole closing and erosion. Hole closing fills drill holes with planar boundary loops, while erosion removes thin plate-like parts. The run time performance of tetrahedral mesh editing based on face groups significantly benefits from massively parallel boundary extraction, which results in a speedup of up to 12.1*× *compared to CPU-sequential processing. The erosion algorithm benefits from massively parallel propagation, optimization, and smoothing resulting in a speedup of up to 5.5*×*. Both editing operations achieved fast run time performance below one second for models of more than* 100 k *elements. For non-mechanical prototypes, cage-based deformation is a versatile method for interactive modeling of geometry. Cage-based deformation allows to express the geometry as linear affine sums of cage vertices, which is a suitable computation scheme for GPUs. The recent advances in cage-based deformation in the local deformation influence, support for quad cage faces, and shape/volume preservation provide means for more advanced use of cage-based deformation in VP applications. The local influence of cage vertices ensures that only the local nearby features of the geometry are deformed, while the global geometric shape is preserved. The support for quad layouts for cages enables symmetry preservation and compliance with industry standards. Due to advanced control over shape and volume preservation, cage-based deformation can be applied to prototypes with smooth organic-like surfaces.*

RQ4: *How can the massively parallel post-processing of unstructured tetrahedral meshes be implemented with efficient memory usage?*

→ ***RA4:*** *The OLBVH spatial data structure can be organized in a linear, compact, and offset-based encoding that allows for massively parallel construction and traversal. On graphics cards without RTX hardware, this results in up to 2.51× fewer memory consumption compared to state-of-the-art spatial data structures. On graphics cards with RTX hardware, the OLBVH consumes up to 1.93× fewer memory. The maximum tree depth can be controlled by a user-defined parameter l_α to control the trade off between memory consumption and run time performance. Through setting $l_\alpha = 2$, the memory consumption can be further reduced by up to 9.87× on graphics cards without RTX and by up to 7.6×, albeit at the cost of reduced runtime performance for the algorithms using OLBVH due to a more shallow hierarchy. Despite the more memory efficient layout, the DVR using OLBVH achieves a speedup of up to 8.4× on graphics cards without RTX hardware, which is the majority of GPUs. This speedup is due to efficient space skipping exploiting the boundary marking of OLBVH tree nodes. On graphics cards with RTX, the hardware-accelerated DVR is one order of magnitude faster than DVR using OLBVH. Besides using a more memory efficient spatial data structure, high-resolution unstructured meshes can be significantly coarsened. The coarsening can reduce the number of tetrahedral elements in a meshes by more than 75 % while preserving most of the smooth features. Sharp transitions in the scalar field might be smoothed by coarsening.*

8.2. Key Conclusions

Since this PhD thesis provides answers to the RQs 1-4, many geometry processing and rendering applications can be improved in terms of run time performance and memory consumption. The results of this PhD thesis predominantly impact VP processes that involve volumetric meshes, while their implications are not limited to the domain of VP.

The presented algorithms for low-level unstructured tetrahedral mesh editing (cf. section 3.1) provide robustness and speedup of important tasks of the virtual prototyping cycle by up to one or two orders of magnitude. The robustness of the presented methods has been validated on a large test set of 10 k unstructured tetrahedral meshes [Hu+20]. The core of the presented massively parallel re-meshing methods are conflict detection methods that prioritize the operations with the most desirable impact on the mesh and determine a dense set of sub-meshes. This thesis has introduced two types of conflict detection. One conflict detection method is specifically for the parallelization of edge/face flips, while the other finds conflict-free sub-meshes around edges. The proposed method for massively parallel edge/face flips enables a speedup of 133× - 254×. The run time performance evaluation of massively parallel harmonic mesh optimization has revealed significant speedups for mesh optimization using vertex relocation and edge/face flips in orchestration. With proper treatment of the mesh boundary using directional derivatives along the boundary elements of the mesh, the massively parallel optimization achieved speedups of 10× - 84× for full boundary preservation and 3.77× - 60× for approximate boundary preservation. The vertex position optimization performs line searches with safety measures that prevent element inversions. For coarsening unstructured tetrahedral meshes, the proposed method achieves a speedup of up to 34× compared to CPU-sequential processing, of up to 7.4× compared to CPU-parallel processing, and of up to 2.7× compared to state-of-the-art GPU-parallel processing.

These substantial speedups enable acceleration of well-established geometry processing applications. As mesh adaptation methods extensively apply edge/face flips, vertex relocation and cavity-based re-meshing [Iba+17], common mesh adaptation tools can use the methods presented in this thesis, in order to more quickly obtain volumetric meshes for numerical simulation. This enables acceleration of complex geometry processing tasks that rely on mesh adaptation. One example is topology optimization using locally adaptive mesh resolutions for finding better solutions [Li+21]. Since shape optimization methods also rely on remeshing to avoid low-quality elements [OSW23], the quick optimization methodology introduced in this PhD thesis can be used to improve mesh quality repeatedly during interactive design changes. Similar implications can be concluded for morphing methods, because these methods are also concerned with producing meshes of sufficient quality for numerical simulations [Sta+11].

For high-level editing (cf. section 3.1) of unstructured tetrahedral meshes, this thesis has presented user-interactive methods for modeling a prototype. The proposed methods offer two ways of user-interaction: face group selection and deformation by cage control. The face groups represent semantic features and can be extracted from CAD or detected with the use of a massively parallel algorithm introduced in this thesis. In order to achieve conceptional proof of the editing approach based on face groups, this thesis has presented the hole closing and erosion editing operations. Massively parallel boundary extraction enabled a speedup of up to 12.1× for hole closing and of up to 5.5× for erosion. Both editing operations have shown fast run time performance below one second for meshes with more than 100 k tetrahedra. Face group-based editing has successfully enabled the modeling of unstructured tetrahedral meshes so that numerical simulation using the method of Weber et al. [Web+13; Web+15] could immediately provide feedback on the design change without returning to CAD.

As face group-based mesh editing does not provide shape deformation, this thesis has comprehensibly reviewed the facilities of cage-based deformation for 3D modeling. This review of the state of the art has revealed that the recent advances in the generation of cages and the cage-based deformation bode well for versatile use in 3D modeling of geometries with smooth organic-like surfaces. The advances in interactive cage generation allow for user-defined cages that respect the semantic parts of the model. The local deformation control of cage-based deformation enable the modification of certain local features while protecting the global shape of the model. As some coordinate types support quad cage layouts, cage-based deformation is conformal with industry standards and preserves symmetric features well. In order to preserve the shape of the surface, some coordinate types provide shape and volume preservation. For the use in VP, the evaluation has shown that cage-based deformation typically provides preservation of element quality due to the smoothness of deformation. Many different coordinate types for cage-based deformation have been proposed, where each type has its own advantages and disadvantages. The most relevant coordinate types can be applied using the open source application CageModeler that was developed throughout this PhD project and can be retrieved at `https://github.com/DanStroeter/CageModeler`.

The customization of geometry is facilitated by using the mesh editing methods presented in this thesis. Users can apply the editing based on face groups for quick customization of models with immediate simulation feedback. Parametric modeling tools such as Fusion [Aut24] could employ the massively parallel face group detection method to enable users to quickly configure a set of face groups. Users can use CageModeler to deform unstructured surface and tetrahedral meshes. This enables the quick modification of prototypes with organic-like surfaces, which can be useful for 3D printing or shape optimization.

For memory-efficient post-processing of unstructured tetrahedral meshes, this thesis has presented an improved version of the OLBVH and algorithms that use this spatial data structure for DVR, coarsening and 3D printing. The memory efficiency of the OLBVH compares well to other spatial data structures that provide GPU-parallel construction. The compact offset-based data layout of the OLBVH organizes mesh elements linearly along a space filling curve, which is the key reason for the efficient usage of memory. As a result, the OLBVH achieves up to 2.51× fewer memory consumption on a GPU without RTX hardware and consumes up to 93 % fewer memory on a GPU with RTX hardware. In order to reduce memory consumption even further, setting a tuning parameter can lead to shallower trees reducing memory consumption by up to 9.87× with RTX and by up to 7.6× without RTX. In the evaluation, the OLBVH construction has exhibited quick run time performance creating an OLBVH within a fraction of a second for meshes consisting of millions of tetrahedra. The run time performance of the DVR algorithm using OLBVH compares well with state-of-the-art methods that do not use RTX hardware acceleration achieving a speedup of up to 8.4×. The largest speedups are achieved on meshes with shell-like boundaries, because the boundary marking allows for efficient empty space skipping for these models. RTX hardware-accelerated DVR is one order of magnitude faster than DVR using OLBVH. However, only a minority of the available GPUs includes RTX hardware. An in-depth profiling has revealed great performance improvement potential for sampling adaptive to the local mesh resolution, transfer function, and surface features.

Since the OLBVH provides controlled use of memory and fast rendering performance, it can be used on GPUs that provide low memory capacity. Due to its fast construction performance, the OLBVH provides acceleration of spatial lookups in time-critical environments such as simulating dynamically deforming bodies. This is interesting for collision detection applications, where several objects are dynamically moving and an intersection check needs to check for collision events. As the OLBVH is designed for volumetric meshes, it allows for collision detection in the interior of an object as it is required for volumetric simulation applications such as cut simulation [DKK22]. In addition, the post-processing of dynamically deforming geometry [Her+14], e.g., the application of a displacement field, can benefit from fast construction of OLBVH for deformed versions of the unstructured volumetric mesh. Due to the memory efficiency of the OLBVH, different spatial data structures can be computed for different deformed states of the geometry, provided that the deformation of the geometry is predictable.

Besides the memory-efficient OLBVH this thesis has presented a coarsening method to compress meshes for DVR, in order to reduce memory consumption. The evaluation has revealed that meshes can be significantly compressed by up to 75 % with good preservation of image fidelity. Smooth scalar data can be preserved well, while sharp transitions in the scalar field may be smoothed by simplification. For 3D printing, this thesis has presented an algorithm for slicing a high-resolution tetrahedral mesh consisting of millions of elements within a few milliseconds for each slice. This slicing algorithm provides a conservative bound of the model instead simple point sampling so that the printed model can be machined to a more fine-grained resolution.

The coarsening for DVR can be useful, whenever the mesh at hand exceeds the memory capacity of the machine. In addition, the coarsening can be used to compress high-resolution meshes for transmission over the web. This can be useful for DVR with access to or retrieval from a database server [Kri+06]. The conservative slicing can be applied in 3D printing of high-resolution geometries consisting of multiple materials [Zas20].

Besides presenting algorithms for massively parallel, memory-efficient processing of unstructured tetrahedral meshes, this thesis has extended the framework of harmonic triangulation [Ale19], which is worthwhile to reflect upon. The framework of harmonic triangulation has been extended by a gradient expression for a vertex of a simplex (cf. section 4.1). This facilitates the usage of Gauß-Seidel iteration order for optimizing vertex positions. Moreover, the new gradient expression consist only of geometric quantities, which enables the geometric interpretation of the harmonic gradient in future work. Another contribution about the harmonic triangulation consists in showing that the usage of edge collapse operations optimizing Dirichlet energy effectively eliminates sliver tetrahedra (see section 5.2).

In view of the implications described above, facilitating the use of the GPU for processing high-resolution unstructured tetrahedral meshes has opened the gate towards fast design and analysis of volumetric geometry.

8.3. Limitations

In order to reflect critically on the results of this thesis, this section describes relevant limitations that decrease the utility of the proposed algorithms for VP processes.

The presented element quality optimization and re-meshing algorithms converge to a local minimum and cannot guarantee to find a global minimum. Consequently, the proposed mesh improvement algorithms cannot guarantee to achieve a target mesh quality. Such a guarantee would require exhaustive optimization algorithms, which will typically exhibit slower run time performance, while the use of the GPU could also be beneficial for this class of optimization algorithms. Especially for complex geometries, terminating in a local optimum for element quality can lead to suboptimal mesh quality.

Similar to morphing methods [Sta+11], the element quality of tetrahedra can degrade throughout user-interactive editing. Therefore, an additional optimization step might be necessary, in order to obtain sufficient tetrahedral element quality for a numerical simulation. This can reduce the run time performance of editing operations so that they are not perceived as immediate anymore.

The run time performance of coarsening high-resolution, unstructured tetrahedral meshes for DVR has exhibited a run time performance of several seconds despite the usage of the GPU. Consequently, the coarsening does not achieve interactive rates for a collapsing of edges until convergence. Thus, it does not meet the demands for real-time applications such as DVR and can only be considered as a pre-process for such applications.

Since this thesis focuses on massively parallel processing, many of the proposed algorithms are designed with high-resolution meshes in mind. For instance, the approximate boundary preservation for optimizing quality (see-section 4.3.4) and the face group determination (see section 6.1.2) are probably not suitable for coarse meshes, while CPU-processing should be fast enough for low-resolution meshes.

As the OLBVH sorts Morton codes of 32-bit integers, the construction is limited to 10 hierarchical tree levels. Larger 64-bit integers could be used for storing Morton codes, which requires adjustment of the described construction. In addition, using larger integers can potentially reduce the run time performance of the construction significantly.

8.4. Future Work

Answering the initial research questions leads to new research questions, which can be addressed in future work. Since the improvements in run time performance are attractive for many applications, there are many avenues for future work based on this thesis. Section 8.4.1 suggests interesting avenues to further improve the capabilities of fast methods to generate unstructured tetrahedral meshes for numerical simulations. In order to better meet the demand for techniques to shorten the VP cycle, it might be worthwhile to further improve the user-interactive editing methods by following the avenues described in section 8.4.2. The capabilities of post-processing could be further improved by following the research avenues described in section 8.4.3. Some of he presented algorithms can be extended to unstructured hexahedral meshes following the research avenues described in section 8.4.4.

8.4.1. Avenues for Faster Generation and Optimization of Unstructured Tetrahedral Meshes

The following paragraphs describe avenues of future work based on the meshing techniques presented in this thesis:

Mesh Optimization along Continuously Curved Boundaries It is an interesting avenue of future work to explore if the boundary vertex optimization along directional derivatives (cf. section 4.3.4) can be extended to curved B-Reps. Since the most common types of CAD curves enable the computation of a derivative, it could be possible to efficiently evaluate the directional derivative for a given element quality function. This would provide a sub-curve along the continuous geometry. A line search with bracketing could potentially be combined with a sophisticated rule to handle the interfaces between different curves so that good mesh quality and convergence properties are achieved. When using super-linear basis functions and higher order elements, a massively parallel algorithm could probably be formulated and support for deforming objects could possibly be achieved. The resulting algorithm could be an important contribution to the recently very active fields of gradient-based mesh optimization [Rem+20; She+21; She+23] and shape optimization [CD20; WRC23; Cha23; HES23], as the available algorithms in these fields oftentimes use level sets and could benefit from increased efficiency. In addition, the proposed algorithm could find its use as an optimization step after mesh generation.

Massively Parallel Mesh Generation using the Advancing Front Method The advancing front (ADF) method generates an unstructured volumetric mesh by repeatedly adding elements to a closed surface, i.e., the front, until the front cannot evolve any further and the volumetric elements form a mesh representing Ω defined by the input closed surface. As the evolving of the surface can be massively parallelized, which was demonstrated by Zhou et al. [ZWY22], it may be possible to devise a massively parallel ADF method for generating a harmonic triangulation (cf. section 2.1.3). As harmonic flips provide good run time performance (cf. section 4.6.2), they can be used in place of Delaunay flips for ADF meshing. The resulting algorithm would allow for massively parallel mesh generation from an input PLC, which is a desirable goal, as most 3D massively parallel Delaunay triangulation algorithms [Cao+14; FLP14; Ray+18] do not ensure to conform to a PLC.

8.4.2. Avenues for Shorter Design Cycles in Virtual Prototyping

The following paragraphs describe avenues of future work based on the interactive editing operations of unstructured tetrahedral meshes presented in this thesis:

Sophisticated Editing of Unstructured Tetrahedral using Morphological Operations It might be possible to extend the proposed editing operations of unstructured tetrahedral meshing by more parametrized and robust morphological operations. Recent work in the field of configurable and robust morphological operations in surface meshes [Sel+20; SVM23] admit stable modeling using advanced morphological operations such as closing or shrink wrap. Therefore, these methods could be applied on selected face groups (cf. section 6.1) of the tetrahedral mesh boundary, in order to allow for more sophisticated editing. With the use of the presented massively parallel low-level mesh editing algorithms, it might be possible to accelerate many steps of these advanced morphological operations with the GPU so that fast run time performance for interactive editing is achieved. As a result, the user would be able to customize a virtual prototype for certain optimization goals, e.g., adding or removing material at selected parts of the model, while benefiting from immediate interactive simulation feedback.

Fast Reconstruction of CAD Geometries using Face Groups Since the face groups provide a link to the original geometry defined in CAD, it could be worthwhile to explore, whether the face groups can be used to reconstruct constructive solid geometry (CSG)-trees from the face groups. Academic research has evolved many methods to reverse-engineer CSG-trees from polygonal meshes [FF24]. Some of these methods could be apt to extract a CSG-tree from the boundary face groups of an edited unstructured tetrahedral mesh. One editing operation of the unstructured tetrahedral mesh typically invalidates the CAD geometry of the prototype. In order to allow for seamless prototyping with unstructured meshes, it would be beneficial to obtain CAD geometries after modeling the discrete mesh with immediate simulation feedback. As a result, users would be able to perform simple modeling operations on the discrete mesh with immediate simulation feedback and the modeling with high-precision demands could be carried out on the reconstructed CAD geometry.

Cage Generation of CAD Geometries for Editing Prototypes So far, cage generation has primarily been applied to discrete surface meshes. In VP, the discrete mesh is typically obtained from discretizing the B-Reps defined in CAD. As present open source CAD frameworks such as Open Cascade [Cap24] provide a multitude of algorithms to model geometry, it may be possible to robustly generate an enclosing cage of a CAD geometry that can be used for deform the discrete mesh. Another important advantage of cages on the basis of CAD geometries is that the generation can exploit the additional information stored by many CAD formats. As many CAD formats allow to tag certain semantic parts of the geometry, the cage generation could wrap the individual semantic parts with coarse cages combining these cages to one cage for the model. This would result in a similar approach to CSG-trees.

8.4.3. Avenues to Advance Rendering Performance

The following paragraphs describe avenues of future work based on the post-processing techniques presented in this thesis:

Improving the Performance of OLBVH As the evaluation revealed the current bottlenecks (see section 7.4.4), which allows to devise potential performance improvements, an interesting avenue of future research is improving the performance of DVR applications using OLBVH. One promising approach is the use of adaptive sampling, because it reduces the number of samples per view ray. Adaptive sampling methods have been successful in improving the performance of ray marching [Kra+07; Mor+19]. Since the OLBVH provides non-loose bounding volumes, each bounding volume covers a unique part of the scalar field. Therefore, statistical concepts could be applied to bounding volumes to govern the sampling rate for each bounding volume. A pre-process could apply a statistical scheme that uses the sampling rate or the local mesh resolution, in order to determine the sampling rate locally. The hierarchical structure of OLBVH could allow for combination of statistics and early approximation of the scalar values during traversal. Since the view rays that intersect the silhouette of the geometry impose the most branching, it could be beneficial to bundle these rays in thread blocks to reduce thread divergence (cf. section 2.2.1).

Fast Dynamic Update of Spatial Data Structures Since the construction of the OLBVH uses a quick bottom-up mapping scheme to assign primitives to unique bounding volumes, it might be beneficial to extend the construction to dynamically changing geometries. Currently, the majority of spatial data structures for quick GPU-parallel construction demands a reconstruction from scratch, whenever unpredictable and large deformations of the corresponding geometry occur. Fast update schemes for moving or otherwise changing geometry can potentially be derived by mapping new primitives and re-mapping changed primitives to existing tree nodes or quickly re-organizing the hierarchy. Probably, such schemes impose the cost of preallocation of larger buffers in memory for adding primitives. In numerical simulation environments, a fast update for the OLBVH could enable DVR of a prototype during its simulation so that the engineers can investigate the impact of loads during an actively running simulation that uses an unstructured volumetric mesh. Perhaps, research in quick updates of spatial data structures for volumetric meshes can spark innovations for maintaining spatial data structures for rendering surface meshes, because BLASes, e.g., for raytracing or pathtracing, need also to be reconstructed after large and unpredictable deformation [Sjo22]. In a thesis under the author's supervision Kelling [Kel24] has shown that culling the scene with non-overlapping AABBs can significantly accelerate pathtracing of dynamic scenes using state-of-the-art game engines. Thus, spatial data structures similar to the OLBVH might be useful for culling techniques as well.

8.4.4. Avenues for Massively Parallel Processing of Unstructured Hexahedral Meshes

Besides tetrahedral meshes, many VP applications use unstructured hexahedral meshes for robust numerical simulation [Pie+22]. However, the fast generation of high-resolution unstructured hexahedral meshes of high element shape quality remains as a challenge. Therefore, the following paragraphs describe avenues of future work for massively parallel processing of unstructured hexahedral meshes:

Massively Parallel Optimization and Re-meshing of Unstructured Hexahedral Meshes In order to accelerate the optimization of high-resolution unstructured hexahedral meshes, it could be worthwhile to extend the optimization and re-meshing algorithms proposed in this PhD thesis to hexahedral meshes. The global optimization of unstructured hexahedral meshes is also a compute-intensive problem, which is typically solved using Gauß-Seidel iterations to prevent inverted elements [Rui+15]. Therefore, a significant speedup of the optimization can be achieved by extending the proposed massively parallel vertex relocation algorithm to unstructured hexahedral meshes. The proposed vertex relocation prevents inversions by an intersection test between the ray in the direction of steepest descent and the halfspace spanned by opposing faces of an element. This can be extended to hexahedral elements to achieve a fast and robust optimization of vertex positions. In addition, the detection and preservation of the boundary features could be extended to hexahedral elements with the use of bilinear quads for interpolating surface normals. In this way, a boundary vertex could be relocated along the surface using directional derivatives.

As unstructured hexahedral meshes can also be optimized performing flips that do not change the boundary of their cavity [VPR19], extending the proposed conflict detection to hexahedral elements is a possible way to accelerate the compute-intensive connectivity optimization of hexahedral meshes. This extension requires computing the adjacent elements to a hexahedral element. Like the proposed algorithm, a conflict detection for hexahedral elements could find the most beneficial flip checking a flip with all of the adjacent elements. In another parallel pass, the locally most beneficial flip can be found. If two adjacent elements share the same most beneficial flip, a locally most beneficial flip is found. While performing the locally most beneficial flip, the other adjacent elements need to be fixed.

Improving Memory Efficiency of Post-processing Unstructured Hexahedral Meshes As the compact offset-based data layout of the OLBVH can be extended to unstructured hexahedral meshes, it is an interesting avenue for future work to use the OLBVH for DVR of hexahedral meshes. This would enable to use the GPU for visualizing simulation results using unstructured hexahedral meshes, which is a current challenge [Pie+22]. Many of the current approaches only use cutting planes to hide certain hexahedra [Bra+19]. Since this does not scale well to meshes with millions of hexahedra, DVR of hexahedral meshes using OLBVH could improve the state of the art in post-processing. For GPU-parallel construction of an OLBVH for unstructured hexahedral meshes, one needs to implement a way to generate Morton codes for a hexahedron so that one Morton code is generated for each cell of the Morton grid intersecting with the given hexahedron. In addition, it would be useful to quickly determine the boundary hexahedra to employ boundary marking of tree nodes for efficient empty space skipping. For convex hexahedra, the proposed massively parallel DVR algorithm using OLBVH could perform the analogue steps to visualize a scalar field of simulation results using an unstructured hexahedral mesh, while for non-convex hexahedra the DVR involves additional challenges for efficient intersection detection and interpolation.

References

[AAN12] G. Anderson, M. Aftosmis, M. Nemec, “Parametric deformation of discrete geometry for aerodynamic shape design”. In: *Proceedings of the 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. Nashville, January 2012, Article 965, 18 pages. DOI: 10.2514/6.2012-965.

[Ado24] Adobe. *Mixamo*. 2024. URL: https://www.mixamo.com (visited on May 7, 2024).

[Al +16] H. Al Akhras, T. Elguedj, A. Gravouil, M. Rochette, “Isogeometric analysis-suitable trivariate NURBS models from standard B-Rep models”. In: *Computer Methods in Applied Mechanics and Engineering* 307 (2016), pp. 256–274. DOI: 10.1016/j.cma.2016.04.028.

[Ala+06] F. Alauzet, X. Li, E. S. Seol, M. S. Shephard, “Parallel anisotropic 3D mesh adaptation by mesh modification”. In: *Engineering with Computers* 21.3 (January 2006), pp. 247–258. DOI: 10.1007/s00366-005-0009-3.

[Ale+20] M. Alexa, P. Herholz, M. Kohlbrenner, O. Sorkine-Hornung, “Properties of Laplace Operators for Tetrahedral Meshes”. In: *Computer Graphics Forum* 39.5 (2020), pp. 55–68. DOI: 10.1111/cgf.14068.

[Ale19] M. Alexa. “Harmonic triangulations”. In: *ACM Transactions on Graphics* 38.4 (2019), pp. 1–14. DOI: 10.1145/3306346.3322986.

[All+05] P. Alliez, D. Cohen-Steiner, M. Yvinec, M. Desbrun, “Variational Tetrahedral Meshing”. In: *ACM Transactions on Graphics* 24.3 (2005), p. 617. DOI: 10.1145/1073204.1073238.

[Alt+18] C. Altenhofen, T. H. Luu, T. Grasser, M. Dennstädt, J. S. Mueller-Roemer, D. Weber, A. Stork, “Continuous Property Gradation for Multi-material 3D-printed Objects”. In: *Proceedings of the 29th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*. SFF ’18. 2018, pp. 1675–1685.

[Alt21] C. Altenhofen. “Volumetric Subdivision for Efficient Integrated Modeling and Simulation”. PhD thesis. Darmstadt, Germany: Technical University of Darmstadt, 2021. DOI: 10.26083/tuprints-00014617.

[AO06] M. Arroyo, M. Ortiz, “Local *maximum-entropy* approximation schemes: A seamless bridge between finite elements and meshfree methods”. In: *International Journal for Numerical Methods in Engineering* 65.13 (March 2006), pp. 2167–2202. DOI: 10.1002/nme.1534.

[Ape14] C. Apetrei. “Fast and Simple Agglomerative LBVH Construction”. In: *Computer Graphics and Visual Computing (CGVC)*. 2014. DOI: 10.2312/cgvc.20141206.

[APH17] D. Anisimov, D. Panozzo, K. Hormann, “Blended barycentric coordinates”. In: *Computer Aided Geometric Design* 52–53 (March 2017), pp. 205–216. DOI: 10.1016/j.cagd.2017.02.007.

[Arp+22] L. Arpaia, H. Beaugendre, L. Cirrottola, A. Froehly, M. Lorini, L. Nouveau, M. Ricchiuto, “H- and r-adaptation on simplicial meshes using MMG tools”. In: *Mesh Generation and Adaptation: Cutting-Edge Techniques*. Springer, 2022, pp. 183–208.

[Asa+21] A. Asaduzzaman, A. Trent, S. Osborne, C. Aldershof, F. N. Sibai, “Impact of CUDA and OpenCL on Parallel and Distributed Computing”. In: *2021 8th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, April 2021. DOI: 10.1109/iceee52452.2021.9415927.

[Att+08] M. Attene, M. Mortara, M. Spagnuolo, B. Falcidieno, “Hierarchical Convex Approximation of 3D Shapes for Fast Region Selection”. In: *Computer Graphics Forum* 27.5 (July 2008), pp. 1323–1332. DOI: 10.1111/j.1467-8659.2008.01271.x.

[Aut24] Autodesk. *Create a face group from faces on a mesh body*. 2024. URL: https://help.autodesk.com/view/fusion360/ENU/?guid=MESH-CREATE-FACE-GROUP (visited on July 10, 2024).

[Aya15] U. Ayachit. *The paraview guide: a parallel visualization application*. Kitware, Inc., 2015.

[Bad+16] C. Bader, W. G. Patrick, D. Kolb, S. G. Hays, S. Keating, S. Sharma, D. Dikovsky, B. Belocon, J. C. Weaver, P. A. Silver, N. Oxman, “Grown, Printed, and Biologically Augmented: An Additively Manufactured Microfluidic Wearable, Functionally Templated for Synthetic Microbes”. In: *3D Printing and Additive Manufacturing* 3.2 (2016), pp. 79–89. DOI: 10.1089/3dp.2016.0027.

[Bak+17] D. Bakunas-Milanowski, V. Rego, J. Sang, Y. Chansu, “Efficient Algorithms for Stream Compaction on GPUs”. In: *International Journal of Networking and Computing* 7.2 (2017), pp. 208–226. DOI: 10.15803/ijnc.7.2_208.

[Bel06] A. Belyaev. “On transfinite barycentric coordinates”. In: *Proceedings of the 4th Symposium on Geometry Processing*. SGP ’06. Cagliari, June 2006, pp. 89–99. DOI: 10.2312/SGP/SGP06/089-099.

[Ben+18] D. Benítez, E. Rodríguez, J. M. Escobar, R. Montenegro Armas, “Parallel optimization of tetrahedral meshes”. In: *Proceedings of the 6th European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th European Conference on Computational Fluid Dynamics*. 2018, pp. 4403–4412.

[BHU10] D. Burkhart, B. Hamann, G. Umlauf, “Iso-geometric Finite Element Analysis Based on Catmull-Clark Subdivision Solids”. In: *Computer Graphics Forum* 29.5 (September 2010), pp. 1575–1584. DOI: 10.1111/j.1467-8659.2010.01766.x.

[Bis17] J. E. Bishop. “Applications of polyhedral finite elements in solid mechanics”. In: *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics*. CRC Press, 2017, pp. 179–196.

[BK04] M. Botsch, L. Kobbelt, “An intuitive framework for real-time freeform modeling”. In: *ACM Transactions on Graphics* 23.3 (August 2004), pp. 630–634. DOI: 10.1145/1015706.1015772.

[BK16] N. Binder, A. Keller, “Efficient Stackless Hierarchy Traversal on GPUs with Backtracking in Constant Time”. In: *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by Ulf Assarsson and Warren Hunt. The Eurographics Association, 2016. DOI: 10.2312/hpg.20161191.

[Ble23] Blender Foundation. *Blender 4.0*. 2023. URL: https://www.blender.org/ (visited on November 27, 2023).

[Ble90] G. E. Blelloch. *Prefix sums and their applications*. Tech. rep. School of Computer Science, Carnegie Mellon University Pittsburgh, PA, USA, November 1990.

[Boy10] S. Boyd. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine Learning* 3.1 (July 2010), pp. 1–122. DOI: 10.1561/2200000016.

[Bra+19] M. Bracci, M. Tarini, N. Pietroni, M. Livesu, P. Cignoni, "HexaLab.net: An online viewer for hexahedral meshes". In: *Computer-Aided Design* 110 (2019), pp. 24–36. DOI: 10.1016/j.cad.2018.12.003.

[BS07] A. I. Bobenko, B. A. Springborn, "A Discrete Laplace–Beltrami Operator for Simplicial Surfaces". In: *Discrete & Computational Geometry* 38.4 (September 2007), pp. 740–756. DOI: 10.1007/s00454-007-9006-1.

[BSB07] A. de Boer, M. S. van der Schoot, H. Bijl, "Mesh deformation based on radial basis function interpolation". In: *Computers & Structures* 85.11–14 (June 2007), pp. 784–795. DOI: 10.1016/j.compstruc.2007.01.013.

[Bud+16] M. Budninskiy, B. Liu, Y. Tong, M. Desbrun, "*Power coordinates*: A geometric construction of barycentric coordinates on convex polytopes". In: *ACM Transactions on Graphics* 35.6 (November 2016), Article 241, 11 pages. DOI: 10.1145/2980179.2982441.

[Bue+24] M. v. Buelow, **D. Ströter**, A. Rak, D. W. Fellner, "A Visual Profiling System for Direct Volume Rendering". In: *Eurographics 2024 - Short Papers*. Ed. by Ruizhen Hu and Panayiotis Charalambous. The Eurographics Association, 2024. DOI: 10.2312/egs.20241030.

[BWG09] M. Ben-Chen, O. Weber, C. Gotsman, "Spatial deformation transfer". In: *Proceedings of the Symposium on Computer Animation*. SCA '09. New Orleans, August 2009, pp. 67–74. DOI: 10.1145/1599470.1599479.

[Cao+14] T.-T. Cao, A. Nanjappa, M. Gao, T.-S. Tan, "A GPU accelerated algorithm for 3D Delaunay triangulation". In: *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 47–54. DOI: 10.1145/2556700.2556710.

[Cap24] Capgemini. *Open Cascade*. 2024. URL: https://www.opencascade.com/ (visited on June 7, 2024).

[Cas+19] S. Casti, M. Livesu, N. Mellado, N. Abu Rumman, R. Scateni, L. Barthe, E. Puppo, "Skeleton based cage generation guided by harmonic fields". In: *Computers & Graphics* 81 (June 2019), pp. 140–151. DOI: 10.1016/j.cag.2019.04.004.

[CB17] S. Calderon, T. Boubekeur, "Bounding proxies for shape approximation". In: *ACM Transactions on Graphics* 36.4 (July 2017), Article 57, 13 pages. DOI: 10.1145/3072959.3073714.

[CC12] J. Cohen, P. Castonguay, "Efficient graph matching and coloring on the GPU". In: *GPU Technology Conference*. 2012, pp. 1–10.

[CD20] B. Chaudet-Dumas, J. Deteix, "Shape Derivatives for the Penalty Formulation of Elastic Contact Problems with Tresca Friction". In: *SIAM Journal on Control and Optimization* 58.6 (2020), pp. 3237–3261. DOI: 10.1137/19M125813X.

[CDD23] J. Chen, F. De Goes, M. Desbrun, "Somigliana coordinates: An elasticity-derived approach for cage deformation". In: *SIGGRAPH 2023 Conference Proceedings*. Los Angeles, July 2023, Article 52, 8 pages. DOI: 10.1145/3588432.3591519.

[CDE+13] B. Cosenza, C. Dachsbacher, U. Erra, “GPU Cost Estimation for Load Balancing in Parallel Ray Tracing”. In: *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*. SciTePress - Science, 2013. DOI: 10.5220/0004283401390151.

[CDH23] Q. Chang, C. Deng, K. Hormann, “Maximum likelihood coordinates”. In: *Computer Graphics Forum* 42.5 (August 2023), Article e14908, 13 pages. DOI: 10.1111/cgf.14908.

[CDM04] B. Cutler, J. Dorsey, L. McMillan, “Simplification and Improvement of Tetrahedral Models for Simulation”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. Association for Computing Machinery (ACM), July 2004. DOI: 10.1145/1057432.1057445.

[CF14] X. Chen, J. Feng, “Adaptive skeleton-driven cages for mesh sequences”. In: *Computer Animation & Virtual Worlds* 25.3–4 (May 2014), pp. 445–453. DOI: 10.1002/cav.1577.

[Cha23] B. Chaudet-Dumas. “A shape optimization algorithm based on directional derivatives for three-dimensional contact problems”. In: *International Journal for Numerical Methods in Engineering* 124.13 (2023), pp. 2935–2964. DOI: 10.1002/nme.7235.

[CHB09] J. A. Cottrell, T. J. Hughes, Y. Bazilevs, *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009. Chap. 1, pp. 1–7.

[Che+13a] S.-W. Cheng, T. K. Dey, J. Shewchuk, S. Sahni, *Delaunay mesh generation*. CRC Press Boca Raton, 2013, pp. 86–88.

[Che+13b] S.-W. Cheng, T. K. Dey, J. Shewchuk, S. Sahni, *Delaunay mesh generation*. CRC Press Boca Raton, 2013, pp. 91–95.

[Che+13c] S.-W. Cheng, T. K. Dey, J. Shewchuk, S. Sahni, *Delaunay mesh generation*. CRC Press Boca Raton, 2013, pp. 96–98.

[Che+13d] S.-W. Cheng, T. K. Dey, J. Shewchuk, S. Sahni, *Delaunay mesh generation*. CRC Press Boca Raton, 2013, pp. 98–102.

[CHP89] J. E. Chadwick, D. R. Haumann, R. E. Parent, “Layered construction for deformable animated characters”. In: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’89. Boston, July 1989, pp. 243–252. DOI: 10.1145/74333.74358.

[Cig+00] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, R. Scopigno, “Simplification of Tetrahedral Meshes with Accurate Error Evaluation”. In: *Proceedings Visualization 2000*. IEEE, 2000. DOI: 10.1109/visual.2000.885680.

[Cig+08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, “Meshlab: an open-source mesh processing tool.” In: *Eurographics Italian chapter conference*. Vol. 2008. Salerno, Italy. 2008, pp. 129–136.

[CM02] P. Chopra, J. Meyer, “TetFusion: An Algorithm for Rapid Tetrahedral Mesh Simplification”. In: *IEEE Visualization, 2002. VIS 2002.* IEEE, 2002. DOI: 10.1109/visual.2002.1183767.

[Com+09] G. Compère, J.-F. Remacle, J. Jansson, J. Hoffman, “A Mesh Adaptation Framework for Dealing with Large Deforming Meshes”. In: *International Journal for Numerical Methods in Engineering* 82.7 (November 2009), pp. 843–867. DOI: 10.1002/nme.2788.

[Cor+12] S. Coros, S. Martin, B. Thomaszewski, C. Schumacher, R. Sumner, M. Gross, “Deformable objects alive!” In: *ACM Transactions on Graphics* 31.4 (August 2012), Article 69, 9 pages. DOI: 10.1145/2185520.2185565.

[Cor+20] F. Corda, J.-M. Thiery, M. Livesu, E. Puppo, T. Boubekeur, R. Scateni, “Real-time deformation with coupled cages and skeletons”. In: *Computer Graphics Forum* 39.6 (January 2020), pp. 19–32. DOI: 10.1111/cgf.13900.

[Cra50] G. Cramer. *Introduction à l’analyse des lignes courbes algébriques*. chez les frères Cramer et C. Philibert, 1750.

[CTO20] Z. Chen, T.-S. Tan, H.-Y. Ong, *On Designing GPU Algorithms with Applications to Mesh Refinement*. July 2020. arXiv: 2007.00324 [cs.GR].

[CV01] T. F. Chan, L. A. Vese, “Active contours without edges”. In: *IEEE Transactions on Image Processing* 10.2 (February 2001), pp. 266–277. DOI: 10.1109/83.902291.

[Das+18] F. Dassi, L. Kamenski, P. Farrell, H. Si, “Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction”. In: *Computer-Aided Design* 103 (2018), pp. 2–13. DOI: 10.1016/j.cad.2017.11.010.

[DCH20] C. Deng, Q. Chang, K. Hormann, “Iterative coordinates”. In: *Computer Aided Geometric Design* 79 (May 2020), Article 101861, 13 pages. DOI: 10.1016/j.cagd.2020.101861.

[DCH88] R. A. Drebin, L. Carpenter, P. Hanrahan, “Volume rendering”. In: *ACM SIGGRAPH Computer Graphics* 22.4 (July 1988), pp. 65–74. DOI: 10.1145/378456.378484.

[Dev+16] M. Deveci, E. G. Boman, K. D. Devine, S. Rajamanickam, “Parallel Graph Coloring for Manycore Architectures”. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2016. DOI: 10.1109/ipdps.2016.54.

[Dey+99] T. Dey, H. Edelsbrunner, S. Guha, D. Nekhayev, “Topology Preserving Edge Contraction”. In: *Publications de l’Institut Mathématique* 66 (1999).

[Dia+23] L. Diazzi, D. Panozzo, A. Vaxman, M. Attene, “Constrained Delaunay Tetrahedrization: A Robust and Practical Approach”. In: *ACM Transactions on Graphics* 42.6 (December 2023). DOI: 10.1145/3618352.

[Dim+06] D. Dimitrov, C. Knauer, K. Kriegel, G. Rote, “On the bounding boxes obtained by principal component analysis”. In: *Proceedings of the 22nd European Workshop on Computational Geometry*. EWCG ’06. Delphi, March 2006, pp. 193–196.

[DJS16] M. Davia-Aracil, A. Jimeno-Morenilla, F. Salas, “A new methodological approach for shoe sole design and validation”. In: *The International Journal of Advanced Manufacturing Technology* 86.9–12 (October 2016), pp. 3495–3516. DOI: 10.1007/s00170-016-8427-5.

[DKK22] H. Das, S. Kumar, S. Kumar, “Precise Parallel FEM-based Interactive Cutting Simulation of Deformable Bodies”. In: *IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 2022, pp. 198–203. DOI: 10.1109/HiPC56025.2022.00036.

[DLM11] Z.-J. Deng, X.-N. Luo, X.-P. Miao, “Automatic cage building with quadric error metrics”. In: *Journal of Computer Science and Technology* 26.3 (May 2011), pp. 538–547. DOI: 10.1007/s11390-011-1153-4.

[DMA02] M. Desbrun, M. Meyer, P. Alliez, “Intrinsic parameterizations of surface meshes”. In: *Computer Graphics Forum* 21.3 (September 2002), pp. 209–218. DOI: 10.1111/1467-8659.00580.

[DS99] H. De Cougny, M. S. Shephard, “Parallel Refinement and Coarsening of Tetrahedral Meshes”. In: *International Journal for Numerical Methods in Engineering* 46.7 (1999), pp. 1101–1125. DOI: 10/b84p49.

[DT07] C. DeCoro, N. Tatarchuk, “Real-time Mesh Simplification using the GPU”. In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. Association for Computing Machinery (ACM), April 2007. DOI: 10.1145/1230100.1230128.

[DTC19] F. Drakopoulos, C. Tsolakis, N. P. Chrisochoides, “Fine-Grained Speculative Topological Transformation Scheme for Local Reconnection Methods”. In: *American Institute of Aeronautics and Astronautics Journal* 57.9 (2019), pp. 4007–4018. DOI: 10.2514/1.j057657.

[DTM17] M. J. Doyle, C. Tuohy, M. Manzke, “Evaluation of a BVH Construction Accelerator Architecture for High-Quality Visualization”. In: *IEEE Transactions on Multi-Scale Computing Systems* 4.1 (2017), pp. 83–94. DOI: 10.1109/tmscs.2017.2695338.

[DV13] J. D’Amato, M. Vénere, “A CPU–GPU framework for optimizing the quality of large meshes”. In: *Journal of Parallel and Distributed Computing* 73.8 (August 2013), pp. 1127–1134. DOI: 10.1016/j.jpdc.2013.03.007.

[Eck+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle, “Multiresolution analysis of arbitrary meshes”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. Los Angeles, September 1995, pp. 173–182. DOI: 10.1145/218380.218440.

[EG15] L. C. Evans, R. F. Gariepy, *Measure Theory and Fine Properties of Functions*. New York: Chapman and Hall/CRC, 2015. DOI: 10.1201/b18333.

[Els+24] Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, A. Zolotas, “A Comprehensive Survey on Delaunay Triangulation: Applications, Algorithms, and Implementations Over CPUs, GPUs, and FPGAs”. In: *IEEE Access* 12 (January 2024), pp. 12562–12585. DOI: 10.1109/access.2024.3354709.

[Eng+04] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, D. Weiskopf, “Real-time volume graphics”. In: *ACM Siggraph 2004 Course Notes*. 2004, 29–es. DOI: 10.1145/1103900.1103929.

[ETA02] M. Elad, A. Tal, S. Ar, “Content based retrieval of VRML objects — An iterative and interactive approach”. In: *Multimedia 2001*. Ed. by Joaquim Jorge, Nuno Correia, Huw Jones, and Meera Blattner Kamegai. Vienna: Springer, 2002, pp. 107–118. DOI: 10.1007/978-3-7091-6103-6_12.

[Far+09] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, D. Lischinski, “Coordinates for instant image cloning”. In: *ACM Transactions on Graphics* 28.3 (August 2009), Article 67, 9 pages. DOI: 10.1145/1576246.1531373.

[Far02] G. E. Farin. *Curves and surfaces for CAGD: a practical guide*. 5. Morgan Kaufmann, 2002, pp. 227–228.

[FF24] P.-A. Fayolle, M. Friedrich, “A Survey of Methods for Converting Unstructured Data to CSG Models”. In: *Computer-Aided Design* 168 (March 2024), p. 103655. DOI: 10.1016/j.cad.2023.103655.

[FJP99] L. Freitag, M. Jones, P. Plassmann, “A Parallel Algorithm for Mesh Smoothing”. In: *SIAM Journal on Scientific Computing* 20.6 (1999), pp. 2023–2040. DOI: 10.1137/s1064827597323208.

[FK98] G. Fairweather, A. Karageorghis, “The method of fundamental solutions for elliptic boundary value problems”. In: *Advances in Computational Mathematics* 9.1–2 (September 1998), pp. 69–95. DOI: 10.1023/A:1018981221740.

[FKR05] M. S. Floater, G. Kós, M. Reimers, "Mean value coordinates in 3D". In: *Computer Aided Geometric Design* 22.7 (October 2005), pp. 623–631. DOI: 10.1016/j.cagd.2005.06.004.

[FLG15] X.-M. Fu, Y. Liu, B. Guo, "Computing locally injective mappings by advanced MIPS". In: *ACM Transactions on Graphics* 34.4 (July 2015), pp. 1–12. DOI: 10.1145/2766938.

[Flo03] M. S. Floater. "Mean value coordinates". In: *Computer Aided Geometric Design* 20.1 (March 2003), pp. 19–27. DOI: 10.1016/S0167-8396(03)00002-5.

[Flo15] M. S. Floater. "Generalized barycentric coordinates and applications". In: *Acta Numerica* 24 (April 2015), pp. 161–214. DOI: 10.1017/s0962492914000129.

[Flo97] M. S. Floater. "Parameterization and smooth approximation of surface triangulations". In: *Computer Aided Geometric Design* 14.3 (April 1997), pp. 231–250. DOI: 10.1016/S0167-8396(96)00031-3.

[FLP14] V. Fuetterling, C. Lojewski, F.-J. Pfreundt, "High-Performance Delaunay Triangulation for Many-Core Computers". In: *Prooceedings of the Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2014, pp. 97–104. DOI: 10.2312/HPG.20141098.

[Fly72] M. J. Flynn. "Some Computer Organizations and Their Effectiveness". In: *IEEE Transactions on Computers* C–21.9 (September 1972), pp. 948–960. DOI: 10.1109/tc.1972.5009071.

[FO97] L. A. Freitag, C. Ollivier-Gooch, "Tetrahedral mesh improvement using swapping and smoothing". In: *International Journal for Numerical Methods in Engineering* 40.21 (November 1997), pp. 3979–4002. DOI: 10.1002/(sici)1097-0207(19971115)40:21<3979::aid-nme251>3.0.co;2-9.

[Fra06] T. Frank. "Advanced visualization and modeling of tetrahedral meshes". PhD thesis. Institut National Polytechnique de Lorraine, 2006.

[FT15] E. Fogel, M. Teillaud, "The computational geometry algorithms library CGAL". In: *ACM Communications in Computer Algebra* 49.1 (2015), pp. 10–12.

[Gal+09] R. Gal, O. Sorkine, N. J. Mitra, D. Cohen-Or, "iWIRES: an analyze-and-edit approach to shape manipulation". In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. New Orleans, Louisiana: Association for Computing Machinery (ACM), 2009. DOI: 10.1145/1576246.1531339.

[Gar+14] A. García, S. Murguia, U. Olivares, F. F. Ramos, "Fast parallel construction of stack-less complete LBVH trees with efficient bit-trail traversal for ray tracing". In: *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM. 2014, pp. 151–158. DOI: 10.1145/2670473.2670488.

[GH97] M. Garland, P. S. Heckbert, "Surface Simplification using Quadric Error Metrics". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97*. ACM Press, 1997. DOI: 10.1145/258734.258849.

[Gha+20] A. Ghazanfarpour, N. Mellado, C. E. Himeur, L. Barthe, J.-P. Jessel, "Proximity-aware multiple meshes decimation using quadric error metric". In: *Graphical Models* 109 (2020), p. 101062. DOI: 10.1016/j.gmod.2020.101062.

[Gie22] A. Giebel. *Harmonic Mesh Optimization on the GPU*. Visual Computing Lab. Technical University of Darmstadt, Department of Computer Science, March 2022.

[GJG18] F. Gu, J. Jendersie, T. Grosch, "Fast and Dynamic Construction of Bounding Volume Hierarchies Based on Loose Octrees". In: *Vision, Modeling and Visualization*. 2018. DOI: 10.2312/vmv.20181257.

[GKN23] P. Gautron, C. Kubisch, NVIDIA Corporation, *Interactive GPU-based Remeshing of Large Meshes*. March 2023. URL: https://register.nvidia.com/flow/nvidia/gtcspring2023/attendeeportal/page/sessioncatalog/session/1666622202853001BIHK (visited on December 6, 2023). NVIDIA GTC Developer Conference.

[GL22] M. González, N. Licheva, *Boundary Face Groups for Mesh Interaction*. Visual Computing Lab. Technical University of Darmstadt, Department of Computer Science, March 2022.

[Gon21] M. González Nothnagel. "Adaptive bounding volume hierarchy for volumetric meshes". Bachelor's Thesis. Technische Universität Darmstadt, 2021.

[GP89] J. Griessmair, W. Purgathofer, "Deformation of solids with trivariate B-splines". In: *Eurographics '89 Conference Proceedings*. Hamburg, September 1989, pp. 137–148. DOI: 10.2312/egtp.19891010.

[GPM11] K. Garanzha, J. Pantaleoni, D. McAllister, "Simpler and faster HLBVH with work queues". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. 2011, pp. 59–64. DOI: 10.1145/2018323.2018333.

[GR09] C. Geuzaine, J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (May 2009), pp. 1309–1331. DOI: 10.1002/nme.2579.

[GR23] C. Geuzaine, J.-F. Remacle, *MSH file format*. 2023. URL: https://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format (visited on September 16, 2023).

[Gre20] B. Gregg. *Flame Graphs*. October 2020. URL: https://www.brendangregg.com/flamegraphs.html (visited on April 5, 2024).

[GW74] W. J. Gordon, J. A. Wixom, "Pseudo-harmonic interpolation on convex domains". In: *SIAM Journal on Numerical Analysis* 11.5 (1974), pp. 909–933. DOI: 10.1137/0711072.

[HA98] A. Heirich, J. Arvo, "A Competitive Analysis of Load Balancing Strategies for Parallel Ray Tracing". In: *The Journal of Supercomputing* 12.1/2 (January 1998), pp. 57–68. DOI: 10.1023/a:1007977326603.

[Has20] D. Hastings. *Atlas Shaper Crank S7-100*. GRABCAD. July 2020. URL: https://grabcad.com/library/atlas-shaper-crank-s7-100-1.

[HB15] T. Hoefler, R. Belli, "Scientific Benchmarking of Parallel Computing Systems". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Association for Computing Machinery ACM, November 2015. DOI: 10.1145/2807591.2807644.

[HCB05] T. Hughes, J. Cottrell, Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement". In: *Computer Methods in Applied Mechanics and Engineering* 194.39-41 (October 2005), pp. 4135–4195. DOI: 10.1016/j.cma.2004.10.008.

[Her+14] I. Herrera, C. Buchart, I. Aguinaga, D. Borro, "Study of a Ray Casting Technique for the Visualization of Deformable Volumes". In: *IEEE Transactions on Visualization and Computer Graphics* 20.11 (2014), pp. 1555–1565. DOI: 10.1109/TVCG.2014.2337332.

[HES23] P. J. Herbert, J. A. P. Escobar, M. Siebenborn, *Shape optimization in $W^{1,\infty}$ with geometric constraints: a study in distributed-memory systems*. 2023. arXiv: 2309.15607 [math.OC].

[HF06] K. Hormann, M. S. Floater, "Mean value coordinates for arbitrary planar polygons". In: *ACM Transactions on Graphics* 25.4 (October 2006), pp. 1424–1441. DOI: 10.1145/1183287.1183295.

[HG00] K. Hormann, G. Greiner, “MIPS: An efficient global parametrization method”. In: *Curve and Surface Design: Saint-Malo 1999* (2000), pp. 153–162.

[Hop96] H. Hoppe. “Progressive meshes”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New Orleans, August 1996, pp. 99–108. DOI: 10.1145/237170.237216.

[Hor05] D. Horn. “Stream reduction operations for GPGPU applications”. In: *GPU Gems 2*. Ed. by Matt Pharr and Randima Fernando. Vol. 2. Addison-Wesley Professional, April 2005. Chap. 36, pp. 573–589.

[HS08] K. Hormann, N. Sukumar, “Maximum entropy coordinates for arbitrary polytopes”. In: *Computer Graphics Forum* 27.5 (July 2008), pp. 1513–1520. DOI: 10.1111/j.1467-8659.2008.01292.x.

[HS17] K. Hormann, N. Sukumar, *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics*. Boca Raton: Taylor & Francis/CRC, 2017. DOI: 10.1201/9781315153452.

[HS86] W. D. Hillis, G. L. Steele, “Data parallel algorithms”. In: *Communications of the ACM* 29.12 (December 1986), pp. 1170–1183. DOI: 10.1145/7902.7903.

[HSO07] M. Harris, S. Sengupta, J. D. Owens, “Parallel prefix sum (scan) with CUDA”. In: *GPU Gems 3*. Ed. by Cyril Zeller, Evan Hart, Ignacio Castaño, Kevin Bjorke, Kevin Myers, and Nolan Goodnight. Addison-Wesley Professional, August 2007. Chap. 39, pp. 851–876.

[HT04] K. Hormann, M. Tarini, “A quadrilateral rendering primitive”. In: *Proceedings of Graphics Hardware*. GH ’04. Grenoble, August 2004, pp. 7–14. DOI: 10.2312/EGGH/EGGH04/007-014.

[Hu+18] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, D. Panozzo, “Tetrahedral meshing in the wild”. In: *ACM Transactions on Graphics* 37.4 (August 2018), Article 60, 14 pages. DOI: 10.1145/3197517.3201353.

[Hu+20] Y. Hu, T. Schneider, B. Wang, D. Zorin, D. Panozzo, “Fast tetrahedral meshing in the wild”. In: *ACM Transactions on Graphics* 39.4 (August 2020), Article 117, 18 pages. DOI: 10.1145/3386569.3392385.

[Hug+13] D. M. Hughes, I. S. Lim, M. W. Jones, A. Knoll, B. Spencer, “InK-Compact: In-Kernel Stream Compaction and Its Application to Multi-Kernel Data Visualization on General-Purpose GPUs”. In: *Computer Graphics Forum* 32.6 (April 2013), pp. 178–188. DOI: 10.1111/cgf.12083.

[Iba+17] D. Ibanez, N. Barral, J. Krakos, A. Loseille, T. Michal, M. Park, “First Benchmark of the Unstructured Grid Adaptation Working Group”. In: *Procedia Engineering* 203 (2017), pp. 154–166. DOI: 10.1016/j.proeng.2017.09.800.

[Iba22] D. Ibanez. *Omega_h*. 2022. URL: https://github.com/sandialabs/omega_h (visited on July 5, 2024).

[Inr24] Inria. *Graphite*. 2024. URL: https://github.com/BrunoLevy/GraphiteThree (visited on June 11, 2024).

[IS16] D. Ibanez, M. Shephard, *Mesh adaptation for moving objects on shared memory hardware*. Tech. rep. 2016-24. Rensselaer Polytechnic Institute, 2016. URL: https://scorec.rpi.edu/REPORTS/2016-24.pdf.

[Jac+11] A. Jacobson, I. Baran, J. Popović, O. Sorkine, “Bounded biharmonic weights for real-time deformation”. In: *ACM Transactions on Graphics* 30.4 (July 2011), Article 78, 8 pages. DOI: 10.1145/2010324.1964973.

[Jac14] A. Jacobson. “Automatic skinning via constrained energy optimization”. In: *Skinning: Real-Time Shape Deformation*. SIGGRAPH 2014 Course Notes. July 2014. Chap. 2. DOI: 10.1145/2614028.2615427.

[Jia+21] Z. Jiang, Z. Zhang, Y. Hu, T. Schneider, D. Zorin, D. Panozzo, “Bijective and coarse high-order tetrahedral meshes”. In: *ACM Transactions on Graphics* 40.4 (July 2021). DOI: 10.1145/3450626.3459840.

[Jia+22] Z. Jiang, J. Dai, Y. Hu, Y. Zhou, J. Dumas, Q. Zhou, G. S. Bajwa, D. Zorin, D. Panozzo, T. Schneider, “Declarative Specification for Unstructured Mesh Editing Algorithms”. In: *ACM Transactions on Graphics* 41.6 (November 2022), pp. 1–14. DOI: 10.1145/3550454.3555513.

[Jia06] X. Jiao. “Volume and Feature Preservation in Surface Mesh Optimization”. In: *Proceedings of the 15th International Meshing Roundtable*. Springer Berlin Heidelberg, 2006, pp. 359–373. DOI: 10.1007/978-3-540-34958-7_21.

[JLW07] T. Ju, P. Liepa, J. Warren, “A general geometric construction of coordinates in a convex simplicial polytope”. In: *Computer Aided Geometric Design* 24.3 (April 2007), pp. 161–178. DOI: 10.1016/j.cagd.2006.12.001.

[Jos+07] P. Joshi, M. Meyer, T. DeRose, B. Green, T. Sanocki, “Harmonic coordinates for character articulation”. In: *ACM Transactions on Graphics* 26.3 (July 2007), Article 71, 9 pages. DOI: 10.1145/1276377.1276466.

[JSW05] T. Ju, S. Schaefer, J. Warren, “Mean value coordinates for closed triangular meshes”. In: *ACM Transactions on Graphics* 24.3 (July 2005), pp. 561–566. DOI: 10.1145/1073204.1073229.

[Ju+05] T. Ju, S. Schaefer, J. Warren, M. Desbrun, “A geometric construction of coordinates for convex polyhedra using polar duals”. In: *Proceedings of the 3rd Symposium on Geometry Processing*. SGP ’05. Vienna, July 2005, pp. 181–186. DOI: 10.2312/SGP/SGP05/181-186.

[Ju+08] T. Ju, Q.-Y. Zhou, M. van de Panne, D. Cohen-Or, U. Neumann, “Reusable skinning templates using cage-based deformations”. In: *ACM Transactions on Graphics* 27.5 (December 2008), Article 122, 10 pages. DOI: 10.1145/1409060.1409075.

[Kar12] T. Karras. “Maximizing parallelism in the construction of BVHs, octrees, and k-d trees”. In: *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. HPG ’12. 2012, pp. 33–37. DOI: 10.2312/EGGH/HPG12/033-037.

[Kav+08] L. Kavan, S. Collins, J. Žára, C. O’Sullivan, “Geometric skinning with approximate dual quaternion blending”. In: *ACM Transactions on Graphics* 27.4 (October 2008), Article 105, 23 pages. DOI: 10.1145/1409625.1409627.

[KBK13] M. Kremer, D. Bommes, L. Kobbelt, “OpenVolumeMesh – A Versatile Index-Based Data Structure for 3D Polytopal Complexes”. In: *Proceedings of the 22nd International Meshing Roundtable*. Springer Berlin Heidelberg, 2013, pp. 531–548. DOI: 10.1007/978-3-642-33573-0_31.

[KE03] M. Kraus, T. Ertl, “Simplification of Nonconvex Tetrahedral Meshes”. In: *Hierarchical and Geometrical Methods in Scientific Visualization*. Springer. 2003, pp. 185–195.

[Kel+83] D. W. Kelly, J. P. De S. R. Gago, O. C. Zienkiewicz, I. Babuska, “A posteriori error analysis and adaptive processes in the finite element method: Part I—error analysis”. In: *International Journal for Numerical Methods in Engineering* 19.11 (November 1983), pp. 1593–1619. DOI: 10.1002/nme.1620191103.

[Kel24] J. Kelling. “Memory-efficient Real-time Path Tracing for Computer Games”. Master’s Thesis. Technische Universität Darmstadt, 2024.

[KG03] Y. Kho, M. Garland, "User-guided simplification". In: *Proceedings of the 2003 Symposium on Interactive 3D Graphics*. I3D '03. Monterey, California: Association for Computing Machinery (ACM), 2003, pp. 123–126. DOI: 10.1145/641480.641504.

[Khr23] Khronos Group. *OpenCL*. 2023. URL: https://www.khronos.org/opencl/ (visited on December 4, 2023).

[Kim+14] J. Kim, Y. Seol, T. Kwon, J. Lee, "Interactive manipulation of large-scale crowd animation". In: *ACM Transactions on Graphics* 33.4 (July 2014), Article 83, 10 pages. DOI: 10.1145/2601097.2601170.

[Knu00] P. M. Knupp. "Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II - A framework for volume mesh optimization and the condition number of the Jacobian matrix". In: *International Journal for Numerical Methods in Engineering* 48.8 (July 2000), pp. 1165–1185. DOI: 10.1002/(sici)1097-0207(20000720)48:8<1165::aid-nme940>3.0.co;2-y.

[Kra+07] M. Kraus, M. Strengert, T. Klein, T. Ertl, "Adaptive sampling in three dimensions for volume rendering on GPUs". In: *6th International Asia-Pacific Symposium on Visualization*. 2007, pp. 113–120. DOI: 10.1109/APVIS.2007.329285.

[Kri+06] K. Krishnan, M. Marcellin, A. Bilgin, M. Nadar, "Efficient transmission of compressed data for remote volume visualization". In: *IEEE Transactions on Medical Imaging* 25.9 (2006), pp. 1189–1199. DOI: 10.1109/TMI.2006.879956.

[KS07] B. M. Klingner, J. R. Shewchuk, "Aggressive Tetrahedral Mesh Improvement". In: *Proceedings of the 16th International Meshing Roundtable*. 2007, pp. 3–23. DOI: 10.1007/978-3-540-75103-8_1.

[KS08] B. M. Klingner, J. R. Shewchuk, *STELLAR A Tetrahedral Mesh Improvement Program*. 2008. URL: https://people.eecs.berkeley.edu/~jrs/stellar/ (visited on July 5, 2024).

[KSE04] T. Klein, S. Stegmaier, T. Ertl, "Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids". In: *12th Pacific Conference on Computer Graphics and Applications, PG 2004. Proceedings.* 2004, pp. 186–195. DOI: 10.1109/PCCGA.2004.1348349.

[Kur11] N. Kurachi. *The magic of computer graphics*. CRC Press, 2011. Chap. 6, pp. 55–58.

[KW03] J. Kruger, R. Westermann, "Acceleration techniques for GPU-based volume rendering". In: *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*. VISUAL-03. IEEE, December 2003. DOI: 10.1109/visual.2003.1250384.

[KW19] M. J. Kochenderfer, T. A. Wheeler, *Algorithms for Optimization*. The MIT Press, 2019. Chap. 3, pp. 43–44.

[Lar+15] M. Larsen, S. Labasan, P. Navrátil, J. Meredith, H. Childs, "Volume Rendering Via Data-Parallel Primitives". In: *Eurographics Symposium on Parallel Graphics and Visualization*. Ed. by C. Dachsbacher and P. Navrátil. The Eurographics Association, 2015. DOI: 10.2312/pgv.20151155.

[LAR20] J. López, C. Anitescu, T. Rabczuk, "CAD-compatible structural shape optimization with a movable Bézier tetrahedral mesh". In: *Computer Methods in Applied Mechanics and Engineering* 367 (2020), p. 113066. DOI: 10.1016/j.cma.2020.113066.

[Lau+09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha, "Fast BVH Construction on GPUs". In: *Computer Graphics Forum* 28.2 (2009), pp. 375–384. DOI: 10.1111/j.1467-8659.2009.01377.x.

[LB16] P. Laug, H. Borouchaki, "Discrete CAD Model for Visualization and Meshing". In: *Procedia Engineering* 163 (2016), pp. 149–161. DOI: 10.1016/j.proeng.2016.11.039.

[LBS06] T. Langer, A. Belyaev, H.-P. Seidel, "Spherical barycentric coordinates". In: *Proceedings of the 4th Symposium on Geometry Processing*. SGP '06. Cagliari, June 2006, pp. 81–88. DOI: 10.2312/SGP/SGP06/081-088.

[LD17] B. H. Le, Z. Deng, "Interactive cage generation for mesh deformation". In: *Proceedings of the 21st Symposium on Interactive 3D Graphics and Games*. I3D '17. San Francisco, February 2017, Article 3, 9 pages. DOI: 10.1145/3023368.3023369.

[LD89] C. T. Loop, T. D. DeRose, "A multisided generalization of Bézier surfaces". In: *ACM Transactions on Graphics* 8.3 (July 1989), pp. 204–234. DOI: 10.1145/77055.77059.

[LH13] X.-Y. Li, S.-M. Hu, "Poisson coordinates". In: *IEEE Transactions on Visualization and Computer Graphics* 19.2 (February 2013), pp. 344–352. DOI: 10.1109/TVCG.2012.109.

[Li+21] H. Li, T. Yamada, P. Jolivet, K. Furuta, T. Kondoh, K. Izui, S. Nishiwaki, "Full-scale 3D structural topology optimization using adaptive mesh refinement based on the level-set method". In: *Finite Elements in Analysis and Design* 194 (2021), p. 103561. DOI: 10.1016/j.finel.2021.103561.

[Lip+07] Y. Lipman, J. Kopf, D. Cohen-Or, D. Levin, "GPU-assisted positive mean value coordinates for mesh deformations". In: *Proceedings of the 5th Symposium on Geometry Processing*. SGP '07. Barcelona, July 2007, pp. 117–123. DOI: 10.2312/SGP/SGP07/117-123.

[LJH13] X.-Y. Li, T. Ju, S.-M. Hu, "Cubic mean value coordinates". In: *ACM Transactions on Graphics* 32.4 (July 2013), Article 126, 10 pages. DOI: 10.1145/2461912.2461917.

[Lju+16] P. Ljung, J. Krüger, E. Groller, M. Hadwiger, C. D. Hansen, A. Ynnerman, "State of the Art in Transfer Functions for Direct Volume Rendering". In: *Computer Graphics Forum* 35.3 (2016), pp. 669–691. DOI: 10.1111/cgf.12934.

[LLC08] Y. Lipman, D. Levin, D. Cohen-Or, "Green coordinates". In: *ACM Transactions on Graphics* 27.3 (August 2008), Article 78, 10 pages. DOI: 10.1145/1360612.1360677.

[LLH22] K. Y. Lam, L.-H. Lee, P. Hui, "3DeformR: Freehand 3D model editing in virtual environments considering head movements on mobile headsets". In: *Proceedings of the 13th Multimedia Systems Conference*. MMSys '22. Athlone, June 2022, pp. 52–61. DOI: 10.1145/3524273.3528180.

[LM14] A. Loseille, V. Menier, "Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive". In: *Prooceedings of the 22nd International Meshing Roundtable*. Springer International Publishing, 2014, pp. 541–558. DOI: 10.1007/978-3-319-02335-9_30.

[LMA15] A. Loseille, V. Menier, F. Alauzet, "Parallel Generation of Large-size Adapted Meshes". In: *Procedia Engineering* 124 (2015), pp. 57–69. DOI: 10.1016/j.proeng.2015.10.122.

[Lo14a] D. S. H. Lo. *Finite Element Mesh Generation*. CRC Press, 2014, pp. 374–377. DOI: 10.1201/b17713.

[Lo14b] D. S. H. Lo. *Finite Element Mesh Generation*. CRC Press, 2014, pp. 557–559. DOI: 10.1201/b17713.

[Lo15] D. S. Lo. *Finite element mesh generation*. CRC Press, 2015. Chap. 1, pp. 7–8.

[LS07] T. Langer, H.-P. Seidel, "Mean value Bézier surfaces". In: *Mathematics of Surfaces XII*. Ed. by Ralph Martin, Malcolm Sabin, and Joab Winkler. Vol. 4647. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 263–274. DOI: 10.1007/978-3-540-73843-5_16.

[LS08] T. Langer, H.-P. Seidel, “Higher order barycentric coordinates”. In: *Computer Graphics Forum* 27.2 (April 2008), pp. 459–466. DOI: 10.1111/j.1467-8659.2008.01143.x.

[LU16] P. Laube, G. Umlauf, “A short survey on recent methods for cage computation”. In: *Proceedings of the 3rd BW-CAR Symposium on Information and Communication Systems*. SInCom ’16. Karlsruhe, December 2016, pp. 37–42.

[LW07] J. Li, Y. Wang, “Automatically constructing skeletons and parametric structures for polygonal human bodies”. In: *Proceedings of the 25th Computer Graphics International Conference*. CGI ’07. Pétropolis, May 2007.

[LZ08] Y. Li, Q. Zhu, “A New Mesh Simplification Algorithm Based on Quadric Error Metrics”. In: *2008 International Conference on Advanced Computer Theory and Engineering*. 2008, pp. 528–532. DOI: 10.1109/ICACTE.2008.92.

[Mar+08] S. Martin, P. Kaufmann, M. Botsch, M. Wicke, M. Gross, “Polyhedral finite elements using harmonic basis functions”. In: *Computer Graphics Forum* 27.5 (July 2008), pp. 1521–1529. DOI: 10.1111/j.1467-8659.2008.01293.x.

[MAS17] J. S. Mueller-Roemer, C. Altenhofen, A. Stork, “Ternary Sparse Matrix Representation for Volumetric Mesh Subdivision and Processing on GPUs”. In: *Computer Graphics Forum* 36.5 (2017), pp. 59–69. DOI: 10.1111/cgf.13245.

[Max+08] A. Maximo, S. Ribeiro, C. Bentes, A. Oliveira, R. Farias, “Memory Efficient GPU-Based Ray Casting for Unstructured Volume Rendering”. In: *IEEE/ EG Symposium on Volume and Point-Based Graphics*. Ed. by Hans-Christian Hege, David Laidlaw, Renato Pajarola, and Oliver Staadt. The Eurographics Association, 2008. DOI: 10.2312/VG/VG-PBG08/155-162.

[MCA15] M. Mandad, D. Cohen-Steiner, P. Alliez, “Isotopic approximation within a tolerance volume”. In: *ACM Transactions on Graphics* 34.4 (July 2015), Article 64, 12 pages. DOI: 10.1145/2766950.

[Mei+21] D. Meister, S. Ogaki, C. Benthin, M. J. Doyle, M. Guthe, J. Bittner, “A Survey on Bounding Volume Hierarchies for Ray Tracing”. In: *Computer Graphics Forum* 40.2 (May 2021), pp. 683–712. DOI: 10.1111/cgf.142662.

[Men+09] W. Meng, B. Sheng, S. Wang, H. Sun, E. Wu, “Interactive image deformation using cage coordinates on GPU”. In: *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*. VRCAI ’09. Yokohama, December 2009, pp. 119–126. DOI: 10.1145/1670252.1670279.

[Mey+02] M. Meyer, H. Lee, A. Barr, M. Desbrun, “Generalized barycentric coordinates on irregular polygons”. In: *Journal of Graphics Tools* 7.1 (2002), pp. 13–22. DOI: 10.1080/10867651.2002.10487551.

[Mis+09] M. K. Misztal, J. A. Bærentzen, F. Anton, K. Erleben, “Tetrahedral Mesh Improvement using Multi-face Retriangulation”. In: *Proceedings of the 18th International Meshing Roundtable*. Springer Berlin Heidelberg, 2009, pp. 539–555. DOI: 10.1007/978-3-642-04319-2_31.

[MJ96] R. MacCracken, K. I. Joy, “Free-form deformations with lattices of arbitrary topology”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New Orleans, August 1996, pp. 181–188. DOI: 10.1145/237170.237247.

[MLD05] E. A. Malsch, J. J. Lin, G. Dasgupta, “Smooth two dimensional interpolants: A recipe for all polygons”. In: *Journal of Graphics Tools* 10.2 (2005), pp. 27–39. DOI: 10.1080/2151237X.2005.10129192.

[MLS11] J. Manson, K. Li, S. Schaefer, “Positive Gordon–Wixom coordinates”. In: *Computer-Aided Design* 43.11 (November 2011), pp. 1422–1426. DOI: 10.1016/j.cad.2011.08.019.

[MLT88] N. Magnenat-Thalmann, R. Laperrière, D. Thalmann, “Joint-dependent local deformations for hand animation and object grasping”. In: *Proceedings of Graphics Interface*. GI ’88. Edmonton, June 1988, pp. 26–33. DOI: 10.20380/GI1988.04.

[MMG06] B. Merry, P. Marais, J. Gain, “Animation space: A truly linear framework for character animation”. In: *ACM Transactions on Graphics* 25.4 (October 2006), pp. 1400–1423. DOI: 10.1145/1183287.1183294.

[Möb27] A. F. Möbius. *Der barycentrische Calcul*. Leipzig: Johann Ambrosius Barth Verlag, 1827.

[Mor+19] N. Morrical, W. Usher, I. Wald, V. Pascucci, “Efficient Space Skipping and Adaptive Sampling of Unstructured Volumes Using Hardware Accelerated Ray Tracing”. In: *2019 IEEE Visualization Conference (VIS)*. IEEE, October 2019. DOI: 10.1109/visual.2019.8933539.

[Mor+22] N. Morrical, I. Wald, W. Usher, V. Pascucci, “Accelerating unstructured mesh point location with RT cores”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.8 (August 2022), pp. 2852–2866. DOI: 10.1109/tvcg.2020.3042930.

[Mor+23] N. Morrical, A. Sahistan, U. Güdükbay, I. Wald, V. Pascucci, “Quick Clusters: A GPU-Parallel Partitioning for Efficient Path Tracing of Unstructured Volumetric Grids”. In: *IEEE Transactions on Visualization and Computer Graphics* 29.1 (2023), pp. 537–547. DOI: 10.1109/TVCG.2022.3209418.

[Mor66] G. M. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. IBM Corporation. 1966. URL: https://dominoweb.draco.res.ibm.com/reports/Morton1966.pdf.

[MP07] P. Milbradt, T. Pick, “Polytope finite elements”. In: *International Journal for Numerical Methods in Engineering* 73.12 (July 2007), pp. 1811–1835. DOI: 10.1002/nme.2149.

[MS10] J. Manson, S. Schaefer, “Moving least squares coordinates”. In: *Computer Graphics Forum* 29.5 (July 2010), pp. 1517–1524. DOI: 10.1111/j.1467-8659.2010.01760.x.

[MS18] J. S. Mueller-Roemer, A. Stork, “GPU-based Polynomial Finite Element Matrix Assembly for Simplex Meshes”. In: *Computer Graphics Forum* 37.7 (2018), pp. 443–454. DOI: 10.1111/cgf.13581.

[MSE07] C. Müller, M. Strengert, T. Ertl, “Adaptive load balancing for raycasting of non-uniformly bricked volumes”. In: *Parallel Computing* 33.6 (June 2007), pp. 406–419. DOI: 10.1016/j.parco.2006.12.002.

[MT23] É. Michel, J.-M. Thiery, “Polynomial 2D Green coordinates for polygonal cages”. In: *SIGGRAPH 2023 Conference Proceedings*. Los Angeles, July 2023, Article 23, 9 pages. DOI: 10.1145/3588432.3591499.

[Mue20] J. S. Mueller-Roemer. “GPU data structures and code generation for modeling, simulation, and visualization”. PhD thesis. Darmstadt, Germany: Technical University of Darmstadt, 2020. DOI: 10.25534/tuprints-00011291.

[Mui+11] P. Muigg, M. Hadwiger, H. Doleisch, E. Groller, “Interactive Volume Visualization of General Polyhedral Grids”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2115–2124. DOI: 10.1109/TVCG.2011.216.

[Mur+13] S. Murguia, F. Avila, L. Reyes, A. Garcia, “Bit-trail traversal for stackless LBVH on DirectCompute”. In: *GPU Pro 4: Advanced Rendering Techniques*. Ed. by Wolfgang Engel. 1st ed. CRC Press, April 2013, pp. 319–336. DOI: 10.1201/b14077-29.

[MV19] S. Mittal, S. Vaishay, “A survey of techniques for optimizing deep learning on GPUs”. In: *Journal of Systems Architecture* 99 (October 2019), p. 101635. DOI: 10.1016/j.sysarc.2019.101635.

[NCC15] M. Naumov, P. Castonguay, J. M. Cohen, “Parallel Graph Coloring with Applications to the Incomplete-LU Factorization on the GPU”. In: *NVIDIA White Papers*. 2015. URL: https://api.semanticscholar.org/CorpusID:15964368.

[ND10] J. Nickolls, W. J. Dally, “The GPU Computing Era”. In: *IEEE Micro* 30.2 (March 2010), pp. 56–69. DOI: 10.1109/mm.2010.41.

[NE04] V. Natarajan, H. Edelsbrunner, “Simplification of Three-dimensional Density Maps”. In: *IEEE Transactions on Visualization and Computer Graphics* 10.5 (September 2004), pp. 587–597. DOI: 10.1109/tvcg.2004.32.

[Nea+06] A. Nealen, T. Igarashi, O. Sorkine, M. Alexa, “Laplacian mesh optimization”. In: *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*. New York, NY, USA: Association for Computing Machinery, 2006, pp. 381–389. DOI: 10.1145/1174429.1174494.

[Nes+09] M. Nesme, P. G. Kry, L. Jeřábková, F. Faure, “Preserving topology and elasticity for embedded deformable models”. In: *ACM Transactions on Graphics* 28.3 (August 2009), Article 52, 9 pages. DOI: 10.1145/1531326.1531358.

[New94] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1994. Chap. 8.

[NF+06] M. Nesme, F. Faure, “Animating shapes at arbitrary resolution with non-uniform stiffness”. In: *Proceedings of the 3rd Workshop on Virtual Reality Interactions and Physical Simulations*. VRIPHYS ’06. Madrid, November 2006, pp. 17–24. DOI: 10.2312/PE/vriphys/vriphys06/017-024.

[NVI22a] NVIDIA. *CUB*. 2022. URL: https://nvlabs.github.io/cub/ (visited on December 11, 2023).

[NVI22b] NVIDIA. *CUDA 11.8 C++ Programming Guide*. September 2022. URL: https://docs.nvidia.com/cuda/archive/11.8.0/cuda-c-programming-guide/index.html (visited on December 5, 2023).

[NVI22c] C. NVIDIA Corporation. *Cusparse library*. 2022. URL: https://developer.nvidia.com/cuda-downloads (visited on July 9, 2014).

[NVI23a] NVIDIA. *CUDA 11.8*. 2023. URL: https://developer.nvidia.com/cuda-11-8-0-download-archive (visited on December 4, 2023).

[NVI23b] NVIDIA. *Displacement Micro-Map Toolkit*. April 2023. URL: https://github.com/NVIDIAGameWorks/Displacement-MicroMap-Toolkit (visited on December 6, 2023).

[NVI23c] NVIDIA. *Thrust*. 2023. URL: https://developer.nvidia.com/thrust (visited on December 11, 2023).

[NVI24a] NVIDIA. *NVIDIA OptiX Ray Tracing Engine*. 2024. URL: https://developer.nvidia.com/optix (visited on July 15, 2024).

[NVI24b] NVIDIA. *NVIDIA RTX Technology*. 2024. URL: https://www.nvidia.com/en-us/design-visualization/technologies/rtx/ (visited on January 25, 2024).

[OG14] E. Okuyan, U. Güdükbay, “Direct volume rendering of unstructured tetrahedral meshes using CUDA and OpenMP”. In: *The Journal of Supercomputing* 67 (2014), pp. 324–344. DOI: 10.1007/s11227-013-1004-x.

[OSW23] S. Onyshkevych, M. Siebenborn, W. Wollner, “Preserving mesh quality in shape optimization”. In: *PAMM* 24.1 (December 2023). DOI: 10.1002/pamm.202300146.

[Pan22] D. Panozzo. “Robust Geometry Processing for Physical Simulation”. In: *Symposium on Geometry Processing Keynotes*. SGP ’22. 2022. URL: https://www.youtube.com/watch?v=rDo3P9NGC28 (visited on May 8, 2024).

[Par22] M. Park. *Refine*. 2022. URL: https://github.com/nasa/refine (visited on July 5, 2024).

[Ped23] J. Peddie. *The History of the GPU - New Developments*. Springer Nature, January 2023. Chap. 5. DOI: 10.1007/978-3-031-14047-1.

[Pen+22] Y. Peng, Y. Yan, S. Liu, Y. Cheng, S. Guan, B. Pan, G. Zhai, X. Yang, “CageNeRF: Cage-based neural radiance field for generalized 3D deformation and animation”. In: *Proceedings of the Conference on Neural Information Processing Systems*. NeurIPS ’22. New Orleans, November 2022, pp. 31402–31415.

[Pie+22] N. Pietroni, M. Campen, A. Sheffer, G. Cherchi, D. Bommes, X. Gao, R. Scateni, F. Ledoux, J. Remacle, M. Livesu, “Hex-Mesh Generation and Processing: A Survey”. In: *ACM Transactions on Graphics* 42.2 (October 2022). DOI: 10.1145/3554920.

[PL10] J. Pantaleoni, D. Luebke, “HLBVH: Hierarchical LBVH Construction for Real-time Ray Tracing of Dynamic Geometry”. In: *Proceedings of the Conference on High Performance Graphics*. HPG ’10. 2010, pp. 87–95. DOI: 10.2312/EGGH/HPG10/087-095.

[Pla23] Plastic Software LLC. *Plasticity*. 2023. URL: https://www.plasticity.xyz/ (visited on November 27, 2023).

[Pop+20] D. Popov, E. Maltsev, O. Fryazinov, A. Pasko, I. Akhatov, “Efficient contouring of functionally represented objects for additive manufacturing”. In: *Computer-Aided Design* 129 (2020), p. 102917. DOI: 10.1016/j.cad.2020.102917.

[Por+21] S. Porziani, C. Groth, W. Waldman, M. E. Biancolini, “Automatic shape optimisation of structural parts driven by BGM and RBF mesh morphing”. In: *International Journal of Mechanical Sciences* 189 (January 2021), Article 105976, 11 pages. DOI: 10.1016/j.ijmecsci.2020.105976.

[PP14] A. Papageorgiou, N. Platis, “Triangular Mesh Simplification on the GPU”. In: *The Visual Computer* 31.2 (November 2014), pp. 235–244. DOI: 10.1007/s00371-014-1039-x.

[PP93] U. Pinkall, K. Polthier, “Computing discrete minimal surfaces and their conjugates”. In: *Experimental Mathematics* 2.1 (1993), pp. 15–36. DOI: 10.1080/10586458.1993.10504266.

[Pre+02] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*. 2nd ed. Cambridge University Pr., 2002, pp. 359–362.

[Pre07] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. Chap. 2, pp. 65–75.

[PTA20] R. Picelli, S. Townsend, H. Alicia Kim, “Microstructural Stress Shape Optimization Using the Level Set Method”. In: *Journal of Mechanical Design* 142.11 (June 2020), p. 111705. DOI: 10.1115/1.4047152.

[Qia10] X. Qian. “Full analytical sensitivities in NURBS based isogeometric shape optimization”. In: *Computer Methods in Applied Mechanics and Engineering* 199.29 (2010), pp. 2059–2071. DOI: 10.1016/j.cma.2010.03.005.

[Rab+17] M. Rabinovich, R. Poranne, D. Panozzo, O. Sorkine-Hornung, “Scalable locally injective mappings”. In: *ACM Transactions on Graphics* 36.2 (April 2017), Article 16, 16 pages. DOI: 10.1145/2983621.

[Ram03] J. Rambau. “On a generalization of Schönhardt’s polyhedron”. In: *Combinatorial and computational geometry* 52 (2003), pp. 501–516.

[Ray+18] N. Ray, D. Sokolov, S. Lefebvre, B. Lévy, “Meshless voronoi on the GPU”. In: *ACM Transactions on Graphics* 37.6 (December 2018). DOI: 10.1145/3272127.3275092.

[Rem+20] E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, P. Fua, “MeshSDF: Differentiable Iso-Surface Extraction”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 22468–22478. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/fe40fb944ee700392ed51bfe84dd4e3d-Paper.pdf.

[RGD22] D. Reed, D. Gannon, J. Dongarra, *Reinventing High Performance Computing: Challenges and Opportunities*. 2022. arXiv: 2203.02544 [cs.DC].

[Rip90] S. Rippa. “Minimal roughness property of the Delaunay triangulation”. In: *Computer Aided Geometric Design* 7.6 (November 1990), pp. 489–497. DOI: 10.1016/0167-8396(90)90011-f.

[RR97] H. Ratschek, J. Rokne, “Test for intersection between box and tetrahedron”. In: *International Journal of Computer Mathematics* 65.3-4 (1997), pp. 191–204. DOI: 10.1080/00207169708804610.

[Rui+15] E. Ruiz-Gironés, X. Roca, J. Sarrate, R. Montenegro, J. Escobar, “Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework”. In: *Advances in Engineering Software* 80 (2015). Civil-Comp, pp. 12–24. DOI: 10.1016/j.advengsoft.2014.09.021.

[Rus07] R. M. Rustamov. *Boundary Element Formulation of Harmonic Coordinates*. Tech. rep. Department of Mathematics, Purdue University, November 2007.

[Şah+21] A. Şahistan, S. Demirci, N. Morrical, S. Zellmann, A. Aman, I. Wald, U. Güdükbay, “Ray-traced Shell Traversal of Tetrahedral Meshes for Direct Volume Visualization”. In: *2021 IEEE Visualization Conference (VIS)*. 2021, pp. 91–95. DOI: 10.1109/VIS49827.2021.9623298.

[San+00] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, J. Snyder, “Silhouette clipping”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. New Orleans, July 2000, pp. 327–334. DOI: 10.1145/344779.344935.

[Sar+23] J. Sarton, S. Zellmann, S. Demirci, U. Güdükbay, W. Alexandre-Barff, L. Lucas, J. M. Dischler, S. Wesner, I. Wald, “State-of-the-art in Large-Scale Volume Visualization Beyond Structured Data”. In: *Computer Graphics Forum* 42.3 (2023), pp. 491–515. DOI: 10.1111/cgf.14857.

[SATS07] C. Stoll, E. de Aguiar, C. Theobalt, H.-P. Seidel, *A volumetric approach to interactive shape editing*. Tech. rep. MPI-I-2007-4-004. Max-Planck-Institut für Informatik, 2007.

[Sca+20] A. Scalas, Y. Zhu, F. Giannini, R. Lou, K. Lupinetti, M. Monti, M. Mortara, M. Spagnuolo, “A first step towards cage-based deformation in virtual reality”. In: *Proceedings of the Italian Chapter Conference on Smart Tools and Apps for Graphics*. STAG ’20. Online, November 2020, pp. 119–130. DOI: 10.2312/STAG.20201246.

[Sch+18] T. Schneider, Y. Hu, J. Dumas, X. Gao, D. Panozzo, D. Zorin, “Decoupling simulation accuracy from mesh quality”. In: *ACM Transactions on Graphics* 37.6 (December 2018), Article 280, 14 pages. DOI: 10.1145/3272127.3275067.

[Sch28] E. Schönhardt. “Über die Zerlegung von Dreieckspolyedern in Tetraeder”. In: *Mathematische Annalen* 98.1 (1928), pp. 309–312.

[SCV14] J. Solomon, K. Crane, E. Vouga, “Laplace-Beltrami: The Swiss army knife of geometry processing”. In: *Symposium on Geometry Processing Graduate School (Cardiff, UK, 2014)*. Vol. 2. 2014.

[SDH23] P. Shirley, T. David Black, S. Hollasch, *Ray tracing in one weekend*. Vol. 4. Github, 2023. Chap. 4. URL: https://github.com/RayTracing/raytracing.github.io.

[Sel+10] S. Sellamani, R. Muthuganapathy, Y. Kalyanaraman, S. Murugappan, M. Goyal, K. Ramani, C. M. Hoffman, “PCS: Prominent cross-sections for mesh models”. In: *Computer-Aided Design and Applications* 7.4 (2010), pp. 601–620. DOI: 10.3722/cadaps.2010.601-620.

[Sel+20] S. Sellán, J. Kesten, A. Y. Sheng, A. Jacobson, “Opening and closing surfaces”. In: *ACM Transactions on Graphics* 39.6 (November 2020). DOI: 10.1145/3414685.3417778.

[SF11] Y. Savoye, J.-S. Franco, “Cage-based tracking for performance animation”. In: *Computer Vision – ACCV 2010*. Ed. by Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto. Vol. 6494. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 599–612. DOI: 10.1007/978-3-642-19318-7_47.

[SF19] A. Schollmeyer, B. Froehlich, “Efficient and Anti-Aliased Trimming for Rendering Large NURBS Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.3 (2019), pp. 1489–1498. DOI: 10.1109/TVCG.2018.2814987.

[SFG20] V. V. Sanzharov, V. A. Frolov, V. A. Galaktionov, “Survey of Nvidia RTX Technology”. In: *Programming and Computer Software* 46.4 (July 2020), pp. 297–304. DOI: 10.1134/s0361768820030068.

[SG98] O. Staadt, M. Gross, “Progressive Tetrahedralizations”. In: *Proceedings Visualization ’98*. IEEE, 1998. DOI: 10.1109/visual.1998.745329.

[Sha+16] M. Shang, C. Zhu, J. Chen, Z. Xiao, Y. Zheng, “A Parallel Local Reconnection Approach for Tetrahedral Mesh Improvement”. In: *Procedia Engineering* 163 (2016), pp. 289–301. DOI: 10.1016/j.proeng.2016.11.062.

[She+08] J. Shen, Z. Chen, Z. Ding, S. Zhang, “Heuristic region growing mesh reconstruction algorithm”. In: *Journal of Zhejiang University (Engineering Science* 12 (2008), p. 007.

[She+21] T. Shen, J. Gao, K. Yin, M.-Y. Liu, S. Fidler, “Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 6087–6101. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/30a237d18c50f563cba4531f1db44acf-Paper.pdf.

[She+23] T. Shen, J. Munkberg, J. Hasselgren, K. Yin, Z. Wang, W. Chen, Z. Gojcic, S. Fidler, N. Sharp, J. Gao, “Flexible Isosurface Extraction for Gradient-Based Mesh Optimization”. In: *ACM Transactions on Graphics* 42.4 (July 2023). DOI: 10.1145/3592430.

[She02a] J. R. Shewchuk. “Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery.” In: *Prooceedings of the 11th International Meshing Roundtable*. Citeseer. 2002, pp. 193–204.

[She02b] J. R. Shewchuk. *What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures*. Preprint. University of California at Berkeley, 2002. URL: https://people.eecs.berkeley.edu/~jrs/papers/elemj.pdf.

[She96] J. R. Shewchuk. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator". In: *Applied Computational Geometry Towards Geometric Engineering*. 1996, pp. 203–222. DOI: 10.1007/bfb0014497.

[SHF13] T. Schneider, K. Hormann, M. S. Floater, "Bijective composite mean value mappings". In: *Computer Graphics Forum* 32.5 (August 2013), pp. 137–146. DOI: 10.1111/cgf.12180.

[Si20] H. Si. *TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator (Version 1.6 - User's Manual)*. Tech. rep. Berlin: Weierstraß-Institut für Angewandte Analysis und Stochastik, August 2020.

[Sil+05] C. T. Silva, J. L. D. Comba, S. P. Callahan, F. F. Bernardon, "A survey of GPU-based volume rendering of unstructured grids". In: *Revista de informática teórica e aplicada. Porto Alegre, RS. Vol. 12, n. 2 (out. 2005), p. 9-29* (2005).

[Sjo22] J. Sjoholm. *Best Practices for Using NVIDIA RTX Ray Tracing (Updated)*. July 2022. URL: https://developer.nvidia.com/blog/best-practices-for-using-nvidia-rtx-ray-tracing-updated/ (visited on July 9, 2024).

[SM06] N. Sukumar, E. A. Malsch, "Recent advances in the construction of polygonal finite element interpolants". In: *Archives of Computational Methods in Engineering* 13.1 (March 2006), pp. 129–163. DOI: 10.1007/BF02905933.

[SMB13] D. Sieger, S. Menzel, M. Botsch, "High Quality Mesh Morphing Using Triharmonic Radial Basis Functions". In: *Proceedings of the 21st International Meshing Roundtable*. Ed. by Xiangmin Jiao and Jean-Christophe Weill. Springer Berlin Heidelberg, 2013, pp. 1–15. DOI: 10.1007/978-3-642-33573-0_1.

[Som85] C. Somigliana. "Sopra l'equilibrio di un corpo elastico isotropo". In: *Il Nuovo Cimento* 17 (December 1885), pp. 140–148. DOI: 10.1007/BF02817783.

[Sor+23] T. Sorgente, S. Biasotti, G. Manzini, M. Spagnuolo, "A Survey of Indicators for Mesh Quality Assessment". In: *Computer Graphics Forum* 42.2 (2023), pp. 461–483. DOI: 10.1111/cgf.14779.

[SOS04] C. Shen, J. F. O'Brien, J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup". In: *ACM Transactions on Graphics* 23.3 (August 2004), pp. 896–904. DOI: 10.1145/1015706.1015816.

[SP86] T. W. Sederberg, S. R. Parry, "Free-form deformation of solid geometric models". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. Dallas, August 1986, pp. 151–160. DOI: 10.1145/15922.15903.

[SS10a] R. Schmidt, K. Singh, "Meshmixer: An Interface for Rapid Mesh Composition". In: *ACM SIGGRAPH 2010 Talks*. SIGGRAPH '10. Los Angeles, California: Association for Computing Machinery, 2010. DOI: 10.1145/1837026.1837034.

[SS10b] M. Schwarz, H.-P. Seidel, "Fast parallel surface and solid voxelization on GPUs". In: *ACM Transactions on Graphics* 29.6 (December 2010), Article 179, 10 pages. DOI: 10.1145/1882261.1866201.

[SS15] J. Smith, S. Schaefer, "Bijective parameterization with free boundaries". In: *ACM Transactions on Graphics* 34.4 (July 2015), pp. 1–9. DOI: 10.1145/2766947.

[SSF10] S. P. Serna, A. Stork, D. W. Fellner, "Tetrahedral Mesh-Based Embodiment Design". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. 2010. DOI: 10.1115/detc2010-28971.

[SSF23] **D. Ströter**, A. Stork, D. W. Fellner, “Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes”. In: *High-Performance Graphics - Symposium Papers*. Ed. by Jacco Bikker and Christiaan Gribble. Presented at High-Performance Graphics 2023. The Eurographics Association, 2023. DOI: 10.2312/hpg.20231139.

[Sta+11] M. L. Staten, S. J. Owen, S. M. Shontz, A. G. Salinger, T. S. Coffey, “A Comparison of Mesh Morphing Methods for 3D Shape Optimization”. In: *Proceedings of the 20th International Meshing Roundtable*. Springer Berlin Heidelberg, 2011, pp. 293–311. DOI: 10.1007/978-3-642-24734-7_16.

[Ste+18] O. Stein, E. Grinspun, M. Wardetzky, A. Jacobson, “Natural Boundary Conditions for Smoothing in Geometry Processing”. In: *ACM Transactions on Graphics* 37.2 (2018), pp. 1–13. DOI: 10.1145/3186564.

[Ste24] M. Stegemann. “Tetraedernetz-Adaption mittels Fehlerabschätzung und harmonischer Optimierung”. Master’s Thesis. Technische Universität Darmstadt, 2024.

[Sti18] M. Stich. *Introduction to NVIDIA RTX and DirectX Ray Tracing*. 2018. URL: https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing/ (visited on July 5, 2024).

[Sto15] A. Stork. “Visual computing challenges of advanced manufacturing and industrie 4.0 [guest editors’ introduction]”. In: *IEEE Computer Graphics and Applications* 35.2 (2015), pp. 21–25. DOI: 10.1109/MCG.2015.46.

[Str+20] **D. Ströter**, J. S. Mueller-Roemer, A. Stork, D. W. Fellner, “OLBVH: octree linear bounding volume hierarchy for volumetric meshes”. In: *The Visual Computer* 36.10-12 (July 2020). **Honorable mention from Fraunhofer IGD for best papers in the category “Impact on Science”**, Presented at Computer Graphics International 2020, pp. 2327–2340. DOI: 10.1007/s00371-020-01886-6.

[Str+21] **D. Ströter**, U. Krispel, J. Mueller-Roemer, D. Fellner, “TEdit: A Distributed Tetrahedral Mesh Editor with Immediate Simulation Feedback”. In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Presented at SIMULTECH 2021. SciTePress 2013. SCITEPRESS - Science and Technology Publications, 2021, pp. 271–277. DOI: 10.5220/0010544402710277.

[Str+22] **D. Ströter**, J. Mueller-Roemer, D. Weber, D. W. Fellner, “Fast harmonic tetrahedral mesh optimization”. In: *The Visual Computer* (June 2022). **Visual Computer Best Paper Award at Computer Graphics International 2022**. DOI: 10.1007/s00371-022-02547-6.

[Str+23] **D. Ströter**, A. Halm, U. Krispel, J. S. Mueller-Roemer, D. W. Fellner, “Integrating GPU-Accelerated Tetrahedral Mesh Editing and Simulation”. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Gerd Wagner, Frank Werner, Tuncer Oren, and Floriano De Rango. Springer International Publishing, 2023, pp. 24–42. DOI: 10.1007/978-3-031-23149-0_2.

[Str+24] **D. Ströter**, J. M. Thiery, K. Hormann, J. Chen, Q. Chang, S. Besler, J. S. Mueller-Roemer, T. Boubekeur, A. Stork, D. W. Fellner, “A Survey on Cage-based Deformation of 3D Models”. In: *Computer Graphics Forum* 43.2 (May 2024). Presented at EUROGRAPHICS 2024. DOI: 10.1111/cgf.15060.

[Str19] **D. Ströter**. “Tetrahedral Mesh Processing and Data Structures for Adaptive Volumetric Mesh Booleans on GPUs”. Master’s Thesis. Technische Universität Darmstadt, 2019.

[Suk04] N. Sukumar. "Construction of polygonal interpolants: A maximum entropy approach". In: *International Journal for Numerical Methods in Engineering* 61.12 (November 2004), pp. 2159–2181. DOI: 10.1002/nme.1193.

[SV03] S. M. Shontz, S. A. Vavasis, "A Mesh Warping Algorithm Based on Weighted Laplacian Smoothing". In: *Prooceedings of the 12th International Meshing Roundtable*. Citeseer. 2003, pp. 147–158.

[SV18] P. Salvi, T. Várady, "Multi-sided Bézier surfaces over concave polygonal domains". In: *Computers & Graphics* 74 (August 2018), pp. 56–65. DOI: 10.1016/j.cag.2018.05.006.

[SVH20] S. M. Shontz, M. A. L. Varilla, W. Huang, "A Parallel Variational Mesh Quality Improvement for Tetrahedral Meshes". In: *Proceedings of the 28th International Meshing Roundtable*. February 2020, pp. 37–49. DOI: 10.5281/zenodo.3653361.

[SVJ15] L. Sacht, E. Vouga, A. Jacobson, "Nested cages". In: *ACM Transactions on Graphics* 34.6 (November 2015), Article 170, 14 pages. DOI: 10.1145/2816795.2818093.

[SVM23] V. K. Suriyababu, C. Vuik, M. Möller, "Towards a High Quality Shrink Wrap Mesh Generation Algorithm Using Mathematical Morphology". In: *Computer-Aided Design* 164 (2023), p. 103608. DOI: 10.1016/j.cad.2023.103608.

[Tak+10] K. Takayama, O. Sorkine, A. Nealen, T. Igarashi, "Volumetric modeling with diffusion surfaces". In: *ACM Transactions on Graphics* 29.6 (December 2010), Article 180, 8 pages. DOI: 10.1145/1882261.1866202.

[TB22] J.-M. Thiery, T. Boubekeur, "Green coordinates for triquad cages in 3D". In: *SIGGRAPH Asia 2022 Conference Proceedings*. Daegu, December 2022, Article 38, 8 pages. DOI: 10.1145/3550469.3555400.

[TDZ19] J. Tao, B. Deng, J. Zhang, "A fast numerical solver for local barycentric coordinates". In: *Computer Aided Geometric Design* 70 (March 2019), pp. 46–58. DOI: 10.1016/j.cagd.2019.04.006.

[Tho48] W. Thomson. "Note on the integration of the equations of equilibrium of an elastic solid". In: *Cambridge and Dublin Mathematical Journal* 3 (1848), pp. 87–89.

[TMB18] J.-M. Thiery, P. Memari, T. Boubekeur, "Mean value coordinates for quad cages in 3D". In: *ACM Transactions on Graphics* 37.6 (December 2018), Article 229, 14 pages. DOI: 10.1145/3272127.3275063.

[TS08] A. Tabarraei, N. Sukumar, "Extended finite element method on polygonal and quadtree meshes". In: *Computer Methods in Applied Mechanics and Engineering* 197.5 (January 2008), pp. 425–438. DOI: 10.1016/j.cma.2007.08.013.

[TTB12] J.-M. Thiery, J. Tierny, T. Boubekeur, "CageR: Cage-based reverse engineering of animated 3D shapes". In: *Computer Graphics Forum* 31.8 (October 2012), pp. 2303–2316. DOI: 10.1111/j.1467-8659.2012.03159.x.

[Uga22] J. Ugalde. *T-Slot 3030 Corner Bracket 60x30*. GRABCAD. September 2022. URL: https://grabcad.com/library/t-slot-3030-corner-bracket-60x30-1.

[Ura00] M. Urago. "Analytical integrals of fundamental solution of three-dimensional Laplace equation and their gradients". In: *Transactions of the Japan Society of Mechanical Engineers Series A* 66.642 (February 2000), pp. 254–261. DOI: 10.1299/kikaia.66.254.

[Var+03] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, D. Manocha, “Efficient max-norm distance computation and reliable voxelization”. In: *Proceedings of the 1st Symposium on Geometry Processing*. SGP ’03. Aachen, June 2003, pp. 116–126. DOI: 10.2312/SGP/SGP03/116-126.

[VBH17] M. Vinkler, J. Bittner, V. Havran, “Extended Morton codes for high performance bounding volume hierarchy construction”. In: *Proceedings of High Performance Graphics*. HPG ’17. 2017, 9:1–9:8. DOI: 10.1145/3105762.3105782.

[VF09] A. Vasilakis, I. Fudos, “Skeleton-based rigid skinning for character animation”. In: *Proceedings of the 4th International Conference on Computer Graphics Theory and Applications*. GRAPP ’09. Lisboa, February 2009, pp. 302–308. DOI: 10.5220/0001799803020308.

[Vii+17] T. Viitanen, M. Koskela, P. Jääskeläinen, H. Kultala, J. Takala, “MergeTree: A Fast Hardware HLBVH Constructor for Animated Ray Tracing”. In: *ACM Transactions on Graphics* 36.5 (2017), p. 169. DOI: 10.1145/3132702.

[Vii+18] T. Viitanen, M. Koskela, P. Jääskeläinen, A. Tervo, J. Takala, “PLOCTree”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.2 (2018), 35:1–35:19. DOI: 10.1145/3233309.

[VPR19] K. Verhetsel, J. Pellerin, J.-F. Remacle, “A 44-element mesh of Schneiders’ pyramid: Bounding the difficulty of hex-meshing problems”. In: *Computer-Aided Design* 116 (2019), p. 102735. DOI: 10.1016/j.cad.2019.102735.

[VWB19] K. Vaidyanathan, S. Woop, C. Benthin, “Wide BVH Traversal with a Short Stack”. In: *Proceedings of the Conference on High-Performance Graphics*. 2019. DOI: 10.2312/hpg.20191190.

[Wac75] E. L. Wachspress. *A Rational Finite Element Basis*. Vol. 114. Mathematics in Science and Engineering. New York: Academic Press, 1975.

[Wal+07] I. Wald, H. Friedrich, A. Knoll, C. D. Hansen, “Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1727–1734. DOI: 10.1109/tvcg.2007.70566.

[Wal+19] I. Wald, W. Usher, N. Morrical, L. Lediaev, V. Pascucci, “RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location”. In: *Proceedings of the Conference on High-Performance Graphics*. HPG ’19. Strasbourg, France: Eurographics Association, 2019, pp. 7–13. DOI: 10.2312/hpg.20191189.

[Wal+21] I. Wald, S. Zellmann, W. Usher, N. Morrical, U. Lang, V. Pascucci, “Ray Tracing Structured AMR Data Using ExaBricks”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 625–634. DOI: 10.1109/TVCG.2020.3030470.

[Wal21] I. Wald. *GPGPU-Parallel Re-indexing of Triangle Meshes with Duplicate-Vertex and Unused-Vertex Removal*. September 2021. DOI: 10.48550/arXiv.2109.09812. arXiv: 2109.09812 [cs.DC].

[Wan+15] Y. Wang, A. Jacobson, J. Barbič, L. Kavan, “Linear subspace design for real-time shape deformation”. In: *ACM Transactions on Graphics* 34.4 (August 2015), Article 57, 11 pages. DOI: 10.1145/2766952.

[Wan+19] Z. Wang, Y. Li, W. Ma, C. Deng, “Positive and smooth Gordon–Wixom coordinates”. In: *Computer Aided Geometric Design* 74 (October 2019), Article 101774, 9 pages. DOI: 10.1016/j.cagd.2019.101774.

[Wan02] G. G. Wang. “Definition and Review of Virtual Prototyping”. In: *Journal of Computing and Information Science in Engineering* 2.3 (September 2002), pp. 232–236. DOI: 10.1115/1.1526508.

[War+07] J. Warren, S. Schaefer, A. N. Hirani, M. Desbrun, “Barycentric coordinates for convex sets”. In: *Advances in Computational Mathematics* 27.3 (October 2007), pp. 319–338. DOI: 10.1007/s10444-005-9008-6.

[War96] J. Warren. “Barycentric coordinates for convex polytopes”. In: *Advances in Computational Mathematics* 6.1 (December 1996), pp. 97–108. DOI: 10.1007/BF02127699.

[WBG07] M. Wicke, M. Botsch, M. Gross, “A finite element method on convex polyhedra”. In: *Computer Graphics Forum* 26.3 (September 2007), pp. 355–364. DOI: 10.1111/j.1467-8659.2007.01058.x.

[WBM21] E. Whalen, A. Beyene, C. Mueller, “SimJEB: Simulated Jet Engine Bracket Dataset”. In: *Computer Graphics Forum* 40.5 (August 2021), pp. 9–17. DOI: 10.1111/cgf.14353.

[Web+12] D. Weber, J. Bender, M. Schnoes, A. Stork, D. Fellner, “Efficient GPU Data Structures and Methods to Solve Sparse Linear Systems in Dynamics Applications”. In: *Computer Graphics Forum* 32.1 (October 2012), pp. 16–26. DOI: 10.1111/j.1467-8659.2012.03227.x.

[Web+13] D. Weber, J. Bender, M. Schnoes, A. Stork, D. Fellner, “Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications”. In: *Computer Graphics Forum*. Vol. 32. 1. Wiley Online Library. 2013, pp. 16–26.

[Web+15] D. Weber, J. Mueller-Roemer, C. Altenhofen, A. Stork, D. Fellner, “Deformation Simulation using Cubic Finite Elements and Efficient p-Multigrid Methods”. In: *Computers & Graphics* 53 (2015), pp. 185–195. DOI: 10.1016/j.cag.2015.06.010.

[WG10] O. Weber, C. Gotsman, “Controllable conformal maps for shape deformation and interpolation”. In: *ACM Transactions on Graphics* 29.4 (July 2010), Article 78, 11 pages. DOI: 10.1145/1778765.1778815.

[Wic+10] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, J. F. O’Brien, “Dynamic local remeshing for elastoplastic simulation”. In: *ACM Transactions on Graphics* 29.4 (July 2010), pp. 1–11. DOI: 10.1145/1778765.1778786.

[Wil+05] A. Williams, S. Barrus, R. K. Morley, P. Shirley, “An efficient and robust ray-box intersection algorithm”. In: *ACM SIGGRAPH 2005 Courses on - SIGGRAPH ’05*. SIGGRAPH ’05. ACM Press, 2005. DOI: 10.1145/1198555.1198748.

[WMZ21] I. Wald, N. Morrical, S. Zellmann, “A Memory Efficient Encoding for Ray Tracing Large Unstructured Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2021), pp. 583–592. DOI: 10.1109/TVCG.2021.3114869.

[WRC23] Z. J. Wegert, A. P. Roberts, V. J. Challis, “A Hilbertian projection method for constrained level set-based topology optimisation”. In: *Structural and Multidisciplinary Optimization* 66.9 (September 2023). DOI: 10.1007/s00158-023-03663-0.

[WS21] Y. Wang, J. Solomon, “Fast quasi-harmonic weights for geometric data interpolation”. In: *ACM Transactions on Graphics* 40.4 (August 2021), Article 73, 15 pages. DOI: 10.1145/3450626.3459801.

[Wyl+02] B. Wylie, K. Moreland, L. Fisk, P. Crossno, “Tetrahedral projection using vertex shaders”. In: *Symposium on Volume Visualization and Graphics. Proceedings. IEEE / ACM SIGGRAPH*. 2002, pp. 7–12. DOI: 10.1109/SWG.2002.1226504.

[XGZ11] C. Xian, S. Gao, T. Zhang, “Tetrahedral Mesh Editing with Local Feature Manipulations”. In: *2011 12th International Conference on Computer-Aided Design and Computer Graphics*. 2011, pp. 130–137. DOI: 10.1109/CAD/Graphics.2011.58.

[XH22] T. Xu, T. Harada, “Deforming radiance fields with cages”. In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni M. Farinella, and Tal Hassner. Vol. 13693. Lecture Notes in Computer Science. Cham: Springer, 2022, pp. 159–175. DOI: 10.1007/978-3-031-19827-4_10.

[Xi+21] N. Xi, Y. Sun, L. Xiao, G. Mei, “Designing Parallel Adaptive Laplacian Smoothing for Improving Tetrahedral Mesh Quality on the GPU”. In: *Applied Sciences* 11.12 (June 2021), p. 5543. DOI: 10.3390/app11125543.

[Xia+19] L. Xiao, G. Yang, K. Zhao, G. Mei, “Efficient Parallel Algorithms for 3D Laplacian Smoothing on the GPU”. In: *Applied Sciences* 9.24 (December 2019), p. 5437. DOI: 10.3390/app9245437.

[XLG09] C. Xian, H. Lin, S. Gao, “Automatic generation of coarse bounding cages from dense meshes”. In: *Proceedings of the International Conference on Shape Modeling and Applications*. SMI ’09. Beijing, June 2009, pp. 21–27. DOI: 10.1109/smi.2009.5170159.

[XLG11] C. Xian, H. Lin, S. Gao, “Automatic cage generation by improved OBBs for mesh deformation”. In: *The Visual Computer* 28.1 (April 2011), pp. 21–33. DOI: 10.1007/s00371-011-0595-6.

[XLX15] C. Xian, G. Li, Y. Xiong, “Efficient and effective cage generation by region decomposition”. In: *Computer Animation & Virtual Worlds* 26.2 (March 2015), pp. 173–184. DOI: 10.1002/cav.1571.

[Xu+09] K. Xu, Z.-Q. Cheng, Y. Wang, Y. Xiong, H. Zhang, “Quality encoding for tetrahedral mesh optimization”. In: *Computers & Graphics* 33.3 (June 2009), pp. 250–261. DOI: 10.1016/j.cag.2009.03.020.

[XZG13] C. Xian, T. Zhang, S. Gao, “Semantic Cage Generation for FE Mesh Editing”. In: *2013 International Conference on Computer-Aided Design and Computer Graphics*. 2013, pp. 220–227. DOI: 10.1109/CADGraphics.2013.36.

[Yan+12] X. Yang, J. Chang, R. Southern, J. J. Zhang, “Automatic cage construction for retargeted muscle fitting”. In: *The Visual Computer* 29.5 (June 2012), pp. 369–380. DOI: 10.1007/s00371-012-0739-3.

[YHC13] M. Yoon, S.-H. Ha, S. Cho, “Isogeometric shape design optimization of heat conduction problems”. In: *International Journal of Heat and Mass Transfer* 62 (2013), pp. 272–285. DOI: 10.1016/j.ijheatmasstransfer.2013.02.077.

[Yif+20] W. Yifan, N. Aigerman, V. G. Kim, S. Chaudhuri, O. Sorkine-Hornung, “Neural cages for detail-preserving 3D deformations”. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. CVPR ’20. Seattle, June 2020, pp. 72–80. DOI: 10.1109/CVPR42600.2020.00015.

[YS19] Z. Yan, S. Schaefer, “A family of barycentric coordinates for co-dimension 1 manifolds with simplicial facets”. In: *Computer Graphics Forum* 38.5 (August 2019), pp. 75–83. DOI: 10.1111/cgf.13790.

[YT08] J. Yin, C. Teodosiu, “Constrained mesh optimization on boundary”. In: *Engineering with Computers* 24.3 (May 2008), pp. 231–240. DOI: 10.1007/s00366-008-0090-5.

[Zas20] M. Zastrow. “The new 3D printing”. In: *Nature* 578.7793 (2020), pp. 20–23.

[Zel+23] S. Zellmann, Q. Wu, K.-L. Ma, I. Wald, “Memory-Efficient GPU Volume Path Tracing of AMR Data Using the Dual Mesh”. In: *Computer Graphics Forum* 42.3 (June 2023), pp. 51–62. DOI: 10.1111/cgf.14811.

[ZG19] D. Zint, R. Grosso, “Discrete Mesh Optimization on GPU”. In: *Lecture Notes in Computational Science and Engineering*. July 2019, pp. 445–460. DOI: 10.1007/978-3-030-13992-6_24.

[Zha+14] J. Zhang, B. Deng, Z. Liu, G. Patanè, S. Bouaziz, K. Hormann, L. Liu, “Local barycentric coordinates”. In: *ACM Transactions on Graphics* 33.6 (November 2014), Article 188, 12 pages. DOI: 10.1145/2661229.2661255.

[ZHL19] S. Zellmann, M. Hellmann, U. Lang, “A Linear Time BVH Construction Algorithm for Sparse Volumes”. In: *2019 IEEE Pacific Visualization Symposium (PacificVis)*. 2019, pp. 222–226. DOI: 10.1109/pacificvis.2019.00033.

[Zho+11] K. Zhou, M. Gong, X. Huang, B. Guo, “Data-Parallel Octrees for Surface Reconstruction”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.5 (2011), pp. 669–681. DOI: 10.1109/TVCG.2010.75.

[ZJ16] Q. Zhou, A. Jacobson, *Thingi10K: A Dataset of 10,000 3D-Printing Models*. 2016. DOI: 10.48550/ARXIV.1605.04797.

[ZWY22] Q. Zhou, Q. Wang, Z. Yu, “SAFT: Shotgun advancing front technique for massively parallel mesh generation on graphics processing unit”. In: *International Journal for Numerical Methods in Engineering* 123.18 (2022), pp. 4391–4406. DOI: 10.1002/nme.7038.

[ZZ87] O. C. Zienkiewicz, J. Z. Zhu, “A simple error estimator and adaptive procedure for practical engineerng analysis”. In: *International Journal for Numerical Methods in Engineering* 24.2 (February 1987), pp. 337–357. DOI: 10.1002/nme.1620240206.

Appendices

A. Glossary

AABB axis aligned bounding box. 126, 127, 128, 129, 130, 131, 133, 134, 135, 145, 146, 156

ADF advancing front. 155

AMIPS advanced most isometric parameterizations. 29, 95, 111, 120

API application programming interface. 9, 15, 56

BBW bounded biharmonic weights. 45, 117, 118, 119, 121, 122, 123

BLAS bottom level acceleration structure. 56, 156

B-Reps boundary representations. 1, 2, 27, 99, 154, 155

CAD computer-aided design. 1, 3, 4, 20, 37, 53, 54, 99, 100, 101, 110, 111, 115, 123, 124, 150, 152, 154, 155, 156

CAM computer aided manufacturing. 145

CC Catmull-Clark. 35, 54

CGAL computational geometry algorithms library. 75

CPU central processing unit. 3, 16, 31, 32, 33, 55, 74, 75, 89, 90, 91, 92, 93, 94, 105, 110, 111, 113, 114, 119, 130, 143, 149, 150, 151, 154

CSG constructive solid geometry. 155, 156

DVR direct volume rendering. 3, 5, 7, 8, 9, 24, 25, 27, 56, 57, 126, 132, 135, 136, 137, 138, 139, 140, 141, 142, 143, 145, 147, 148, 151, 152, 153, 154, 156, 157

EMC energy minimization-based coordinates. 44, 46, 49, 117, 118, 119, 120, 121, 122, 123, 124

FEA finite element analysis. 2, 4, 5, 33, 53, 100, 123, 124

FEM finite element method. 2, 10, 24, 27, 29, 33, 34, 45, 53, 88, 100, 124, 136

GBC generalized barycentric coordinates. 21, 22, 23, 40, 41, 42, 43, 44, 46, 47, 48, 49, 117, 118, 119, 123

GC Green coordinates. 49, 50, 51, 52, 118, 119, 120, 121, 123

GPGPU general purpose graphics processing unit. 15

GPU graphics processing units. 1, 2, 3, 4, 5, 8, 9, 15, 16, 19, 20, 25, 31, 32, 34, 35, 40, 42, 53, 54, 55, 56, 57, 62, 63, 64, 74, 75, 79, 80, 81, 83, 87, 89, 90, 92, 94, 97, 100, 105, 106, 107, 110, 111, 113, 114, 115, 119, 123, 125, 128, 131, 134, 135, 140, 143, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157

HC harmonic coordinates. 44, 45, 117, 118, 119, 121, 122, 123

HLBVH hierarchical linear bounding volume hierarchy. 54, 55

HPC high-performance computing. 3, 32

IGA isogeometric analysis. 53, 54

LBC local barycentric coordinates. 45, 46, 117, 118, 119, 121, 122, 123, 124

LBS linear blend skinning. 23, 35, 36, 45

LBVH linear bounding volume hierarchy. 54, 55, 126, 130, 137, 138, 139, 140, 146, 147, 148

LOBVH loose octree bounding volume hierarchy. 55

MEC maximum entropy coordinates. 47, 123

MIPS most isometric parameterizations. 29

MLC maximum likelihood coordinates. 47, 48, 49, 117, 118, 119, 120, 122, 123

MVC mean value coordinates. 41, 42, 43, 48, 117, 118, 119, 123, 124

NURBS non-uniform rational B-splines. 1, 53, 54

OBB oriented bounding box. 38, 39

OLBVH octree linear bounding volume hierarchy. 5, 7, 41, 56, 125, 126, 127, 128, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 145, 146, 147, 148, 151, 152, 153, 154, 156, 157

PCA principal component analysis. 38, 39

PDE partial differential equation. 2, 3, 44, 45, 46, 49, 53

PLC piecewise linear complex. 12, 13, 105, 111, 112, 155

PMVC positive mean value coordinates. 42, 43, 123

QFS quadratic fit search. 87

QGC Green coordinates for tri-quad cages. 50, 52, 118, 120, 121, 122, 123, 124

QMVC mean value coordinates for tri-quad cages. 43, 44, 50, 52, 118, 120, 123

quad quadrilateral. 21, 43, 44, 45, 53, 117, 120, 121, 122, 123, 150, 152, 157

RBF radial basis function. 36

RGBA red, green, blue and alpha. 24

SAH surface area heuristic. 54, 55

SBC spherical barycentric coordinates. 43, 123

SC Somigliana coordinates. 52, 53, 119, 120, 121, 122, 123

SIMD single instruction multiple data. 15, 16, 17

SIMT single instruction multiple threads. 15, 16

TCSR ternary compressed sparse row. 19, 62, 65, 72, 83, 89, 92, 97, 102, 106, 109, 113, 128, 143

TLAS top level acceleration structure. 56

VP virtual prototyping. 1, 3, 4, 5, 6, 27, 34, 36, 37, 38, 53, 54, 59, 80, 81, 97, 99, 100, 102, 110, 114, 115, 116, 117, 118, 120, 122, 123, 124, 126, 147, 149, 150, 151, 152, 153, 154, 155, 156

B. Supervisory Activities

B.1. Lectures

1. Teaching assistant for the *Graphische Datenverarbeitung I* course (TU Darmstadt TUCaN ID 20-00-0040-iv)
2. Teaching assistant for the *Graphische Datenverarbeitung II* course (TU Darmstadt TUCaN ID 20-00-0041-iv)

B.2. Practicals

1. Supervision of several programming practicals in the *(Advanced) Visual Computing Lab* course (TU Darmstadt TUCaN IDs 20-00-0537-pr and 20-00-0418-pr).
2. Supervision of several teaching practicals for the *Praktikum in der Lehre - Graphische Datenverarbeitung I* course (TU Darmstadt TUCaN ID 20-00-1101-pl).

B.3. Seminars

1. Supervision of several seminal works in the *Applied Topics of Computer Graphics* seminar (TU Darmstadt TUCaN ID 20-00-0724-se).

B.4. Bachelor's Theses

[Gon21] M. González Nothnagel. "Adaptive bounding volume hierarchy for volumetric meshes". Bachelor's Thesis. Technische Universität Darmstadt, 2021

B.5. Master's Theses

[Kel24] J. Kelling. "Memory-efficient Real-time Path Tracing for Computer Games". Master's Thesis. Technische Universität Darmstadt, 2024

[Ste24] M. Stegemann. "Tetraedernetz-Adaption mittels Fehlerabschätzung und harmonischer Optimierung". Master's Thesis. Technische Universität Darmstadt, 2024

C. Conference Presentations of Published Papers

During my PhD-project, I have presented my publications at several conferences. The following list shows my conference presentations. If a conference presentation was given online due to the COVID-19 pandemic, the presentation is marked as (online presentation):

- Computer Graphics International (CGI) October, 2020. Presentation of the paper: **"OLBVH: octree linear bounding volume hierarchy for volumetric meshes"**. Geneva, Switzerland (online presentation). URL: https://www.youtube.com/watch?v=0Uzvu1v1SkA
- International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH) July, 2021. Presentation of the paper: **"TEdit: A Distributed Tetrahedral Mesh Editor with Immediate Simulation Feedback"**. Lisbon, Portugal. (online presentation)
- Computer Graphics International (CGI) September, 2022. Presentation of the paper: **"Fast harmonic tetrahedral mesh optimization"**. Geneva, Switzerland (online presentation). URL: https://www.youtube.com/watch?v=K2gBBpKf6bc
- High-Performance Graphics (HPG) June, 2023. Presentation of the paper: **"Massively Parallel Adaptive Collapsing of Edges for Unstructured Tetrahedral Meshes"**. Delft, Netherlands. Conference is sponsored by European Association for Computer Graphics (EUROGRAPHICS) and ACM Siggraph. URL: https://www.youtube.com/watch?v=xY_W_No9Jxk
- Conference of the European Association for Computer Graphics (EUROGRAPHICS) April, 2024. Presentation of the paper: **"A Survey on Cage-based Deformation of 3D Models"**. Limassol, Cyprus.