Maverick Meshing Methods

Maverick Meshing Methods

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dennis René Bukenberger, M.Sc.

aus Stuttgart

Tübingen

2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	30.07.2021
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Hendrik P. A. Lensch
2. Berichterstatter:	Prof. Dr. Andreas Schilling

Abstract

The digital representation of real-world objects is a classical challenge in computer graphics, most commonly approached with either modeling or, where our focus lies, reconstructing the object from a scan. The information collected during a non-intrusive 3D scan of an object usually gets stored in the form of a point cloud. This loose set of three-dimensional positions only implies the geometry of the sampled surface. There is an established collection of different techniques and approaches to actually recover a closed hull from this kind of information. However, many of those methods still struggle under certain circumstances, produce artifacts, lack detail, or come with other drawbacks. Our goal is to explore this challenge in novel and unconventional ways rather than increment on existing research. This thesis is based on four main contributions, each presenting valuable additions to the state-of-the-art in their individual fields.

Our first surface meshing procedure is able to produce pure quad-meshes with adaptive resolution and feature-aligned structures. In contrast to other standard methods, ours does not require frame-fields, parameterization, or even normal information; a simple unoriented point cloud is already sufficient as input. A hierarchical space partitioning structure is utilized to cluster meaningful neighborhoods of the point cloud and interconnect them with quadrangular tiles, using a bottom-up algorithm.

Volume meshing procedures usually require at least a surface and some kind of auxiliary input like frame-fields, parameterized mappings, sometimes even handcrafted split planes, singularity graphs, or annotated feature edges. Our hex-meshing pipeline is able to skip the surface precomputation step and reconstruct hex-dominant volume meshes directly from point clouds, and naturally also from surfaces if given. Furthermore, our procedure introduces a novel generalized mesh class, called at-most-hexa meshes: It constrains the meshes to only feature hexahedral or smaller primitives but no general, larger, and arbitrarily shaped polyhedra as it is common in other hex-dominant meshes. A fundamental part of this volume meshing procedure is properly aligning hexahedral cells within the given volume, using a Lloyd relaxation under the L_∞ norm.

Our third contribution allows us to analytically formulate the L_∞ energy within a cell of the before mentioned relaxation procedure, using tetrahedral geometry extended with a density domain. Moreover, we propose various other application scenarios of tetrahedral meshes under linearly varying density; like analyzing, optimizing, and even 3D printing such geometry with altered mass properties.

The fourth contribution introduces another installment of a meshing technique in a rather classical sense, focused on accurate surface reconstructions with fine details, sharp features, and an overall high mesh uniformity. This is realized using a growing mesh complex: Starting from a small trivial initialization mesh or a rough approximation of the target, the mesh progressively assimilates towards the target hull, which can be given as a volumetric function, mesh or point cloud. Furthermore, we show how this method can be adapted to improve the reconstruction accuracy of other meshing pipelines, exemplarily demonstrated on our own quad- and hex-meshed results.

Due to the common origin and related fields of research, we first draw the links between the individually proposed contributions in introductory background sections. Each technique is described and discussed in a dedicated chapter but eventually concluded with a joint discussion on future improvements and achieved results.

Kurzfassung

Die digitale Abbildung von Objekten aus der realen Welt ist eine zentrale Herausforderung in der Computergrafik. Dabei wird häufig mit Modellierung gearbeitet, oder aber mit der Rekonstruktion von 3D-Abtastungen, worauf der Fokus dieser Arbeit liegt. Für gewöhnlich werden die Daten, die durch eine nicht-invasive 3D-Abtastung des Objekts erfasst werden, in Form einer Punktwolke gespeichert. Diese lose Menge an dreidimensionalen Positionen stellt die Geometrie der abgetasteten Oberfläche nur implizit dar. Es existiert bereits eine Vielzahl etablierter Verfahren und Methoden, um geschlossene Oberflächen aus dieser Art der Darstellung zu rekonstruieren. Allerdings haben viele dieser Verfahren auch Schwachstellen: Sie scheitern unter bestimmten Bedingungen, die Ergebnisse beinhalten Artefakte oder es mangelt an Details in der Rekonstruktion. Unser erklärtes Ziel ist es, diese Herausforderung auf noch unbetretenen Wegen anzugehen, ohne uns durch bereits existierende Forschung den Blick für bessere Lösungen trüben zu lassen. So basiert diese Dissertation auf vier Beiträgen, die grundlegenden Neuerungen zum aktuellsten Stand der jeweiligen Forschungsbereiche darstellen.

Unser erstes Verfahren zur Rekonstruktion einer soliden Objekthülle ist in der Lage, Oberflächennetze zu erstellen, die sich allein aus viereckigen Elementen zusammensetzen. Diese Netze haben eine adaptive Auflösung, welche auf der Abtastdichte der Punktwolke basiert, während sich der Netzfluss entsprechend an der Ausprägung bestimmter Objektdetails orientiert. Im Kontrast zu anderen gängigen Verfahren benötigt unseres für die Rekonstruktion keine Orientierungsfelder, Parametrisierungen oder Normalen-Information; eine einfache unorientierte Punktwolke genügt bereits als Eingabe. Dafür wird eine hierarchische Raum-Partitionierungsstruktur verwendet, welche die Punktwolke in kleine, zweckmäßige Regionen aufteilt und als solche bündelt. Jeweils vier davon werden zu einer viereckigen Kachel zusammengefasst. Diese Kacheln werden dann der Hierarchie entlang von unten nach oben miteinander verbunden, was letztendlich zu einer komplett geschlossenen Oberfläche führt.

Verfahren zur Erstellung von Netzstrukturen für Volumina setzen neben der standardmäßigen Hülle oft noch weitere Eingabedaten voraus, wie Orientierungsfelder, parametrisierte funktionale Abbildungen, manchmal sogar manuell erstellte Trennebenen, Singularitätsgraphen oder markierte Objektkanten. Unser Volumennetzverfahren ist dabei in der Lage, die Oberflächenvorberechnung zu überspringen und Hexaeder-Netze direkt von einer Punktwolke zu rekonstruieren, natürlich aber auch von einer Oberfläche, falls diese verfügbar ist. Des Weiteren führen wir mit diesem Verfahren eine neue generali-

sierte Netzform ein, die sogenannten Höchstens-Hexaeder-Netze. Dabei wird das Netz auf Primitive beschränkt, die maximal einem Hexaeder entsprechen oder topologisch kleiner sind. Auf diese Weise enthält es keine generellen und beliebigen Polyeder, wie es in anderen Hexaeder-dominierten Netzen der Fall ist. Ein fundamentaler Schritt im Verfahren zur Erstellung dieser Volumennetze ist es, die Hexaeder aneinander, aber auch an der gegebenen Eingabeform, auszurichten. Dafür wird Lloyds Algorithmus, ein iteratives Optimierungsverfahren, in Kombination mit der L_∞-Norm verwendet.

Die dritte Hauptkomponente der vorliegenden Arbeit befasst sich explizit mit dem Energieterm einer Zelle, den es in einer solchen L_∞-Optimierung zu minimieren gilt. Dafür werden die Zellen in einzelne Tetraeder zerlegt, um selbige sodann durch eine weitere Dimension für Dichte zu erweitern. Dies ermöglicht es letztendlich, die L_∞-Energie analytisch zu formulieren und geometrische Schwerpunkte entsprechend herzuleiten. Darüber hinaus zeigen wir, wie es gelingen kann, Masseneigenschaften mit Tetraeder-Geometrie unter variabler Dichte exakt zu bestimmen. Auf diese Weise können parametrisierte Dichtefelder nun auch so optimiert werden, dass bei gleichbleibender Geometrie gewünschte Masseneigenschaften erreicht werden. Dies wird unter anderem an einem 3D-gedruckten Bauteil mit optimierten Rotationseigenschaften demonstriert.

Das vierte vorgestellte Verfahren bezieht sich wieder auf die ursprüngliche Herausforderung, eine Punktwolke möglichst akkurat zu rekonstruieren. Besondere Anforderungen an Genauigkeit gelten hierbei den sehr feinen Oberflächendetails, scharfen Kanten und einer allgemein hohen Netz-Uniformität. Dieses hohe Maß an Präzision wird durch eine wachsende Netzstruktur erreicht: Ausgehend von einer trivialen Startform oder groben Approximation des Zielobjekts, beginnt sich das Netz progressiv der Zielhülle anzunähern. Das zu rekonstruierende Objekt kann hierbei als volumetrische Funktion, Oberflächennetz oder Punktwolke angegeben werden. Diese Methode ist zudem dafür geeignet, Verbesserungen an bereits bestehenden Rekonstruktionen vorzunehmen, was wir an Beispielen unserer eigenen Vierecks- und Hexaeder-Netze experimentell aufzeigen.

In der Einführung dieser Arbeit erläutern wir zunächst den thematischen Zusammenhang der einzelnen technischen Verfahren und gehen vertiefend auf die dazu benötigten theoretischen Grundlagen ein. In den darauffolgenden Kapiteln werden jeweils ein Verfahren, die Motivation dahinter sowie die damit erreichten Ergebnisse detailliert behandelt. Abschließend werden mögliche Weiterentwicklungen und der gesamtwissenschaftlichen Beitrag dieser Arbeit nochmals zusammenfassend zu diskutieren.

Acknowledgments

I want to thank ...

- Prof. Dr. Hendrik P. A. Lensch, first and foremost. I am very grateful for the opportunity to work and research within his Computer Graphics Group at the University of Tübingen and I consider myself very lucky to be able to develop my skills and this thesis under his supervision. Attending his introductory lecture on computer graphics as an undergraduate student really sparked my interest in this field as a scientific discipline. Since then he was more than supportive and always encouraged the development of novel project ideas. His contagious enthusiasm, thought-out remarks and clever insights greatly influenced this work.

- Dr. Benjamin Resch for his kind help and fast support on almost every technical issue imaginable and, together with Lukas Ruppert, for the administration of our outstanding computing infrastructure.

- all (current and past) members and friends of our group for many fruitful discussion, valuable feedback, and an overall very enjoyable and balanced work environment; namely in alphabetical order: Mark Boss, Raphael Braun, Jieen Chen, Manuela Di Paolo, Andreas Engelhardt, Zohreh Ghaderi, Fabian Groh, Sebastian Herholz, Andreas Karge, Simon Holdenried-Krafft, Manuel Lange, Violaine Le Guily, Arijit Mallick, Dr. Benjamin Resch, Lukas Ruppert, Leonard Salewski, Dr. Katharina Schwarz, Hassan Shahmohammadi, Dr. Jian Wei, Dr. Patrick Wieschollek, Faezeh Zakeri.

- Prof. Dr. Andreas Schilling, as my secondary supervisor, for his intermediate feedback rounds on current developments, knowledgeable comments and eventually reviewing this thesis.

- Prof. Dr. Marco Tarini for the inspiring collaboration on the at-most-hexa meshing project and his fantastic work on *HexaLab*.

- my parents Ilka and René, and my sister Carina, for providing such a vibrant and creative environment while growing up, shaping the person that I am today.

- my beloved Sonja for her unconditional love, encouraging support, exceptional patience, and always being my personal ray of sunshine.

Furthermore, this thesis would not have been possible without the excellent, free, and open-source software and tools listed in the following:

Python [Pyt20] was used as the main programming framework for all necessities and fast prototyping. Even greater with the following extensions:

NumPy [Num20]	for fast vectorized math operations.
trimesh [Daw]	for lightweight powerful mesh handling.
Mayavi [Ent20]	for fast 3D visualizations of any kind.
PyCuda [KPL*12]	for simple integration of Cuda GPU code.
SciPy [Sci20]	for eased scientific computing.

Blender [Ble20] was used for intuitive interaction, visualization and editing of 3D input and output, and obviously to render all the magnificent result images in this document.

LaTeX [LaT20] as the typesetting tool for all involved publications and this thesis.

Wikipedia for obvious reasons.

Contents

Chapter 1

Introduction

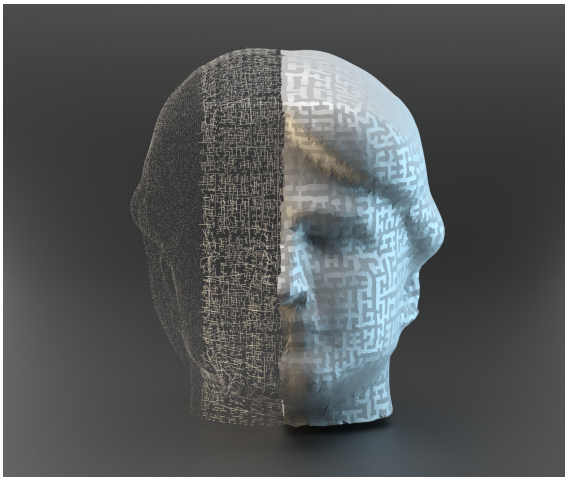
Geometric models of 3D objects are the basis for many classic applications in computer graphics but are also essential to other related fields, such as scientific visualization, light transport and physics simulation, animation, architecture- and manufacturing design, virtual and augmented realities, medical applications and many more. Geometric data persists in various forms and can be arranged on a spectrum[1] ranging from pure syntactic information, i.e., the raw data of a 3D scan, over simple but closed surfaces, then useful qualitative meshes, up to structure-oriented and semantically driven formulations. Dedicated rendering pipelines may already generate plausible visualizations of lower-end geometric information, like sparsely sampled surface data, but other downstream applications usually require a surface description for animation, simulation, manufacturing, or advanced rendering tasks. Manually modeling digital objects is very time consuming and, therefore, the automation of 3D scan reconstructions is an everlasting research topic in computer graphics and also the main objective of this thesis: Explore and assess novel methods for transforming the most basic forms of 3D data into higher-quality meshes.

[1] Leif Kobbelt, 108. Bundeskongress Aachen (2017)

1.1 Motivation and Main Contributions

Dependent on the field of application, the input for geometry processing pipelines can be interpreted to either represent the boundary hull of an object or describe a volume. While both can be described in explicit meshed forms, geometry may also be implied with surface samples as point clouds, isosurfaces, signed distance fields, functions of curves, or volumetric occupancy. The possibility to extract or convert one into, or from, the other primarily depends on their specification: Whereas a closed hull also indicates a volume and a meshed volume is trivially enclosed by a hull, a volumetric *in/out* function does not specify a surface, and oriented hull samples do not contain a volume. Especially the extraction of surfaces and meshes from a point cloud or functional representation can be seen as a very intricate reverse-engineering task. Without any preconception of what a target object may look like and only given a loose set of seemingly arbitrary three-dimensional points, the algorithm is challenged to generate an oriented surface or a meshed 3D model. However, the procedure can be designed to rely on assumptions from the real world where the 3D scan was taken from, for example, the properties of differentiable oriented manifold surfaces. Further constraints derive from the nature of signal processing: While suitable filtering and interpolation techniques may compensate for sampling artifacts, the detail level of a reconstruction will eventually always depend on the quality of the input. We present four novel methods, utilizing specific mesh forms, three of which are primarily focused on reconstructions.

The concepts proposed in this thesis have partially been published on their own - or have overlapping content - with the papers listed in the following.



▶ Chapter 4

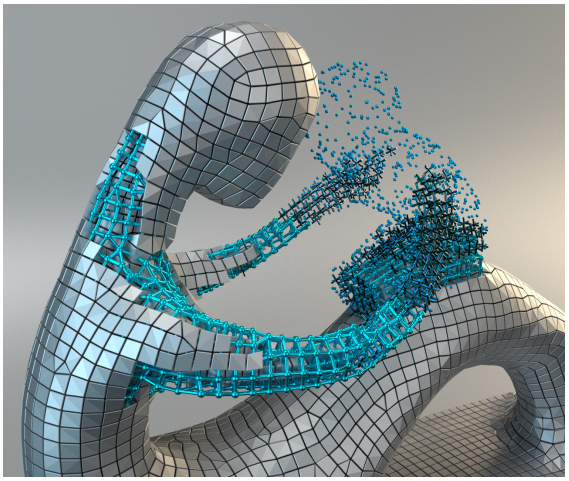
Hierarchical Quad Meshing of 3D Scanned Surfaces

[BL18], SGP 2018
DOI: 10.1111/cgf.13497

Concept This chapter tackles the unorganized chaos of a point cloud with a spatial divide-and-conquer algorithm and a dedicated subsampling technique to reconstruct a closed surface hull mesh. Therefore, space is hierarchically structured with a balanced *k*d-tree that organizes surface samples in geometrically meaningful leaf-node groups. Suitable subsamples of these groups represent the vertices for the final mesh. The hierarchy of the tree structure allows extracting topology by interconnecting mesh tiles from the bottom up. This results in a maze-like tile structure that is eventually entirely closed by incrementally tracing and filling up dead ends.

Contributions Contrasting other methods, this concept does not require the input point cloud to be oriented, i.e., feature normal vectors for the sample points, thus solely relies on their geometric positions. The resulting mesh features quadrangular tiles only, which is often a favorable criterion over the most straightforward form with triangular faces. Furthermore, the topology of the final structure can be aligned to features of the input with specifically oriented nodes in the *k*d-tree. Our algorithm always results in a watertight manifold as it responds to very steep gradients of input sample density with an adaptive mesh resolution, whereas other methods would usually generate holes.

The image above illustrates the intermediate stages of the meshing process with the input point cloud on the left, the *k*d-tree in the middle-left, and the resulting closed quad-mesh on the right. Shading variations on the quad-mesh highlight the maze structure derived from the interconnection hierarchy.



▶ Chapter 5

At-Most-Hexa Meshes

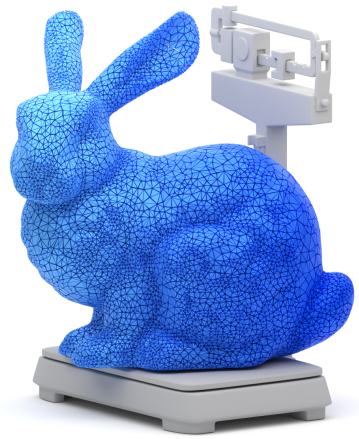
[BTL21], CGF 2021
DOI: 10.1111/cgf.14393

Concept The reconstruction in this chapter focuses not only on the surface of an object but also on the interior structures within the volume. Our algorithm is based on a relaxation scheme, namely a Lloyd relaxation under the L_∞ norm. The volume is populated with cells, which, during the relaxation, eventually approach hexahedral shapes, align to the input as well as to each other. An internal graph structure allows controlling alignment and regularity during the optimization. This provides the basis for the topology extraction, formulated as graph-matching subroutines and designed for efficient execution with minimal branching. The generated individual primitive types are then sorted by quality and assembled as the final mesh.

Contributions Other pipelines dedicated to hex-dominant meshing usually require quite complex and often manually constructed input. Our proposed procedure, however, requires much less information for comparable results, e.g., a surface mesh or point cloud is already sufficient. Furthermore, general volume meshes may feature quite arbitrary polyhedral cells, which are often difficult to handle in downstream applications. Therefore, this chapter introduces a new generalized class of hex-dominant volume meshes, limiting the topology of the featured primitives to be at most hexahedral, hence the name at-most-hexa mesh. The primitives, including tetrahedra and triangular prisms, are specified as linear interpolations of a hexahedron, thus can be easily formulated in common hex-mesh data structures, suitable for further hex-mesh processing pipelines.

Shown above is a beauty-shot, created from the results of our pipeline. The chosen object visualizes the transition from the input point cloud in the form of a child to the final mesh, personified by the mother. Orientations aligned during the relaxation and the neighborhood graph connecting individual cells are also visualized.

The image won 2nd Place in the CGF Cover Contest 2020.



▶ Chapter 6

Tetrahedra of Varying Density and their Applications

[BL21b], CGI 2021
DOI: 10.1007/s00371-021-02189-0

Concept This chapter employs methodologies known from fields of physics and simulation that have yet found very little recognition in computer graphics. The calculation of mass properties, especially under varying density, requires the non-trivial integration of arbitrary polyhedra. A standard fallback solution is to rely on voxelized approximations, which simplify the integration but also introduce aliasing bias. Therefore, our focus in this chapter lies in calculating mass properties for volume meshes, using not voxels but tetrahedra as integrable primitive.

Contributions Proposed applications include the parameterization of a density field suitable for optimization tasks. For example, while keeping the tetrahedral geometry unchanged, adjusting the density dimension allows for moving the object's center of gravity or modifying its rotational inertia. Results are demonstrated on 3D printed examples with improved rotational stability. The main application, however, targets the relaxation part of Chapter 5 that relies on the integration of volumetric Voronoi cells in a non-Euclidean space. With the introduction of a density domain on the relaxing cells, the energy term to be minimized becomes analytically feasible. This eventually allows formulating a novel and hybrid approach on three-dimensional uniformly oriented $L_{2|\infty}$ Lloyd relaxations.

The illustration above shows a tetrahedral volume mesh with density varying linearly from left to right, visualized as a blue shaded gradient. Tetrahedral geometry allows computing analytically accurate mass properties even with vertex-specific density values.

▶ Chapter 7

Be Water my Friend: Mesh Assimilation

[BL21a], CGI 2021
DOI: 10.1007/s00371-021-02183-6

Concept The algorithm proposed in this chapter advances on the initial motivation of this thesis, as it reconstructs high-quality triangular meshes from various kinds of input, such as volumetric functions, other meshes, and obviously also point clouds. Therefore, a simple initialization mesh, placed in the implicit geometry representation, automatically grows or shrinks towards the anticipated hull, iteratively approaching the target shape in a coarse to fine approximation scheme. During this iterative procedure, the geometry and topology of the mesh constantly adapt until the desired result quality is attained.

Contributions The core assimilation method proposed in this chapter was initially designed to improve upon inaccurate reconstructions. With our novel bilateral weighting scheme, mesh vertices individually transition between shape approximation and hull projection. As vertices are effectively pulled towards intersections in the input's tangent space, fine details and sharp features are reconstructed very accurately. Moreover, when combined with dedicated mesh expansion steps, the concept now generalizes standard meshing techniques such as ballooning and shrink-wrapping. We show how to improve upon the accuracy of the quad-meshes from Chapter 4, the quad-dominant hulls from Chapter 5 and demonstrate superior reconstruction accuracy in a large benchmark, competing against common state-of-the-art reconstruction methods.

While this procedure can be applied for remeshing tasks or to produce meshes from signed distance fields or volume data, the illustration above shows its primary target application: point cloud reconstruction. The reconstruction result is shown in the top half, and the bottom half illustrates the 3D scanned input point cloud. However, this image has virtually nothing in common with the procedure itself and is merely a literal visualization of the concept idea: water materializing a point cloud.

The image won 2nd Place in the CGF Cover Contest 2021.

1.2 Structure of this Thesis

Before diving into the technical details of our main contributions, Chapter 2 will first revise common terminology and introduce the theoretical background for the upcoming chapters. Further, will Chapter 3 provide an overview of research related to our main challenges by reviewing its scientific history. Each of the technical Chapters 4, 5, 6 and 7 is framed with a short introduction and discussion, respectively. Finally, a summarizing outlook and positioning of this research along with a conclusive discussion are featured in the final Chapter 8.

While each individual technical chapter provides conclusive results to demonstrate its value as a standalone contribution, the map, illustrated in Figure 1.1, further structures their relations within overlapping fields of research:

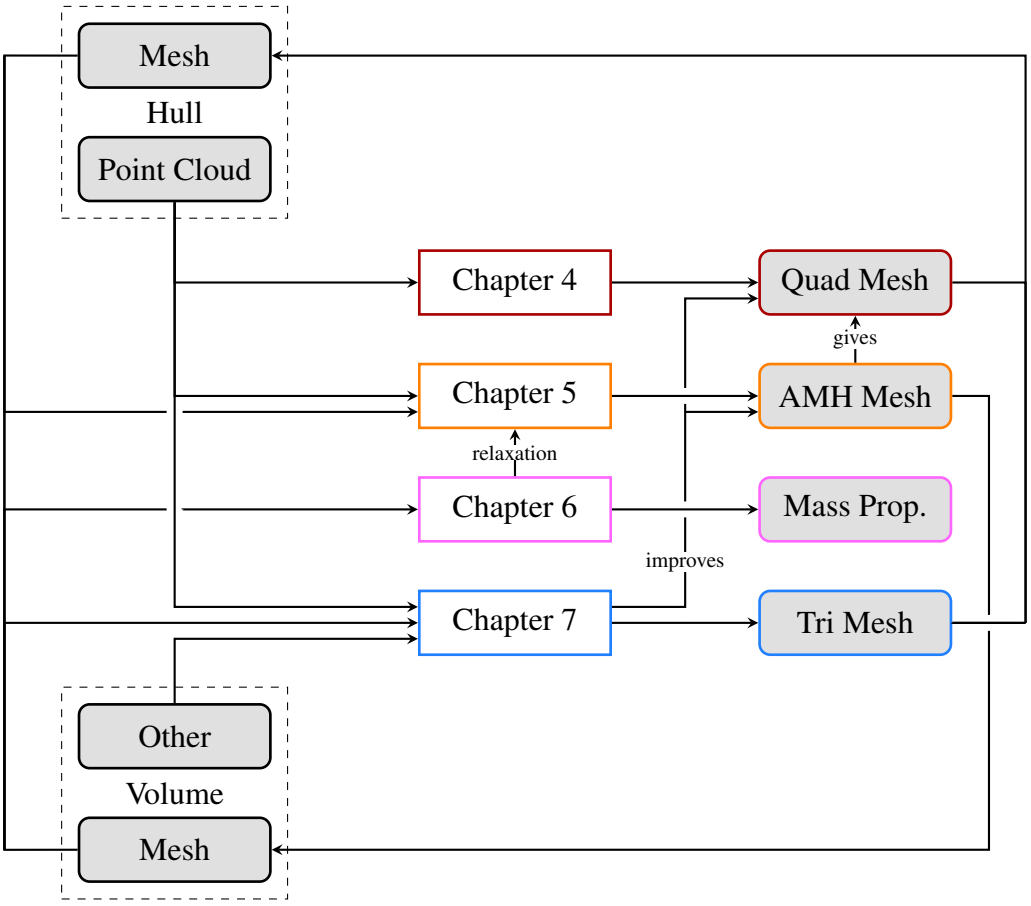


Figure 1.1: Structural Overview: Possible input modalities with implicit/explicit hulls and volumes are listed on the left, the resulting outputs on the right. The technical chapters of this thesis are grouped in the center.

Chapter 2

Background

This chapter provides the theoretical background for the upcoming technical chapters. First, we will give an overview on the formulation of meshes and summarize basic terminology, then also recapitulate involved algorithmic concepts and finally assess employed quality measurements with regard to reconstruction accuracy.

2.1 Mesh Structures

As mentioned in the introduction, there are different forms to describe 3D geometry, e.g., using curves, functions, volumetric representations, surface samples, or meshes. Discrete meshes usually consist of a geometrical and a topological component, both to be discussed in the following. This section will further elaborate on the validity of a mesh and conclude with remarks regarding efficient implementation.

2.1.1 Geometry

A common trait in all indexed mesh data structures is the set of vertices $\mathcal{V}$, where each entry is associated with a geometric position in space, formally expressed as:

$$\mathcal{V} = \{v_0, \dots, v_{V-1}\} \qquad\qquad v_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \in \mathbb{R}^3$$

The implementation of $\mathcal{V}$ as an array of shape $V \times 3$ allows for an easy application of simple transformations in form of vector addition or matrix multiplication.

2.1.2 Topology

The amount of topological information stored in a mesh may vary, dependent on the intended field of application. In the most basic mesh definition, a *Vertex-Vertex* mesh, each vertex also holds a list of indices, referencing its direct vertex neighbors. This is rather impractical as edges (link between 2 vertices) and faces (compound of a least 3 vertices)

are stored only implicitly. Contrasting this frugal formulation are specifications like the *Winged-Edge* [Bau72] and *Half-Edge* [McG00] data structures. These mesh forms store a lot of associative information by cross-indexing vertices, edges, and faces, thus also require a lot of maintenance when the mesh topology changes.

The most commonly used form to represent, handle and store meshes is the *Face-Vertex* form. In addition to the set of vertices $\mathcal{V}$, specified above, a second set $\mathcal{F}$ defines the individual polygonal faces of the mesh by referencing vertex indices:

$$\mathcal{F} = \{f_0, \ldots, f_{F-1}\} \qquad f_i = [j, \ldots], \; j \in [0, V-1]$$

This formulates a balanced compromise between memory footprint, required maintenance and explicitly specified relations between vertex and face entities. Consequently, many popular file formats, such as `.obj`, `.off` or `.ply`, are based on this simple indexed mesh data structure, with individual extensions or restrictions, respectively. The set of edges $\mathcal{E}$ can be extracted from the face specification. Indices of a face are strictly ordered, thus edges are given as two consecutive index entries.

Whereas this definition already allows for specifying surfaces as compounds of polygonal faces, it can be further generalized to also formulate meshed volumes. Therefore, arbitrary polyhedra are stored in a set of cells $\mathcal{C}$, referencing the faces via indices:

$$\mathcal{C} = \{c_0, \ldots, c_{C-1}\} \qquad c_i = [k, \ldots], \; k \in [0, F-1]$$

Individual vertex indices' may be referenced by multiple faces or cells, but the indices within a face or cell are assumed to be unique. Thus, one can state that

$$3 \leq |f_i| \leq V \qquad \text{and} \qquad 4 \leq |c_i| \leq F$$

which defines the minimum face to be a triangle and the minimum cell a tetrahedron. Upper limits are trivially given by the number of entities to be referenced.

2.1.3 Validity

The geometrical and topological components are rather independent of each other, thus can also be modified individually. Nevertheless, only the validity of both components guarantees for meshed objects without artifacts. Geometric defects usually emerge from misplaced vertices, e.g., relative in the form of noise and outliers but also absolute as transformation and deformation. Details on the measurement and quantification of geometric properties, especially with respect to reconstruction accuracy, are discussed in the upcoming Section 2.5. Topological defects, however, are no less critical as they can make all the difference between a loose polygon soup and a solid object. Topological

validity can be formally expressed, as a surface mesh obeys the rules of a 2D manifold. This property states that it will result in a disk if cut with a (small enough) sphere. Another important attribute of a mesh is that it is closed, intuitively also referred to as watertightness (WT). Both principles can be summarized with the following axioms:

- An edge connects 2 vertices.
- An edge is shared by 2 faces.
- A face is surrounded by a closed path of edges.
- A vertex is surrounded by a ring of edges and faces.

As mentioned before, compounds of multiple faces may either specify the hull of a single object or, with many cells, describe the partitioning of a volume. Either way, general polyhedra, i.e., polygonal surface meshes, obey the *Euler characteristic*:

$$\chi = V - E + F$$

which is fix for objects of a certain genus, and where V, E and F give the number of vertices, edges and faces in mesh, respectively. The characteristic states for objects of genus 0 (homeomorphic to a sphere) to have $\chi = 2$, for objects of genus 1 (torus) to have $\chi = 0$, for genus 2 (double torus) $\chi = -2$ and so on.

Regarding volumetric cell structures, our assembly routine (Section 5.6.2) obeys the relaxed interface conformity constraints for hex-dominant meshes, as proposed by Yamakawa and Shimada [YS03]. These constraints basically state that cells are not allowed to intersect or (partially) include each other, for example, two prisms in a hexahedron.

2.1.4 Implementation

So far, all definitions were subject to a purely theoretical formulation. With respect to an efficient implementation, seemingly simple tasks may already pose time-critical challenges. For example, when only given the indices of a face or cell tuple, checking for existence, guaranteeing uniqueness, or identifying a particular tuple requires dedicated search routines and suitable data structures. Therefore, *Cantor*'s paring function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and its recursive generalization $\pi^n : \mathbb{N}^n \to \mathbb{N}$, defined by

$$\begin{aligned} \pi(k_1, k_2) &= \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2 \\ \pi^n(k_1, \ldots, k_{n-1}, k_n) &= \pi(\pi^{n-1}(k_1, \ldots, k_{n-1}), k_n) \end{aligned}$$

can be used as an easy way to map multidimensional index vectors, such as edge or face tuples, to unique integer values. These are then suitable to be utilized in combination with hash-based data structures to efficiently approach the challenges mentioned above.

Furthermore, the introduced definition of faces and cells as index tuples of varying lengths is also rather impractical regarding array-based data structures. Especially the arbitrary complexity of polyhedral volume cells quickly approaches the limits of common indexed mesh structures [BKP*10]. Therefore, in Section 5.3 we propose a dedicated new mesh class, wherein every primitive from sliver, over tetrahedron and prism, up to a hexahedron is expressed as a linear combination of at most eight vertices. This allows to specify cells, not over the indices of faces but as ordered, same-sized tuples, referencing vertices. As 8 unique indices encode a hexahedron, doubled indices correspond to collapsed edges of the hexahedron and, consequently, allow formulating smaller primitives.

2.2 Orientation Representations

The contour or shape of a surface is fundamental for many of the upcoming tasks, e.g., for guiding primitives in Section 5.5.3, as projection space in Section 7.3 or for the qualitative evaluation of a reconstruction as introduced in Section 2.5. This section lists employed best-practice methods on the computation and handling of orientations.

2.2.1 Surface Normals

The principle *sidedness* of a surface at a specific point can be defined with the direction and sign of a normal vector, orthogonal to the surface and pointing outwards. With meshed surfaces as our main objective, this vector is unambiguously defined for each triangular surface facet. For a face f, it computes as the normalized cross product of two of its edges $e_i, e_j \in f$:

$$n_f = \frac{e_i \times e_j}{|e_i \times e_j|}$$

Non-triangular structures like quad-meshes are easily triangulated by inserting diagonals for each quadrangular face. Following a common heuristic, usually, the shortest of the two diagonals in a quad face is created.

Normals on non-face entities are computed with less certainty as the surface is not differentiable on edges and vertices, thus orthogonality is not defined. Nevertheless, edge and vertex normals can be formulated as combinations of the normals from adjacent faces. For an edge, there are only two possible face normals to be considered and, therefore, the normalized average is a common approximation, e.g., for an edge $e \in f_i \cap f_j$:

$$n_e = \frac{n_{f_i} + n_{f_j}}{|n_{f_i} + n_{f_j}|}$$

For a vertex, however, this method is not suitable as it will be biased from the triangulation of the mesh. For example, on a triangulated unit cube, corner vertices can be connected to 3, 4, 5, or 6 triangle faces, thus the averaged face normals would not generally lead to the obvious $\sqrt{3}(\pm 1, \pm 1, \pm 1)^T$ corner normals. Alternatively, vertex normals are usually expressed as a weighted average, based on local geometry [TW98, Max99]. While sometimes the triangle areas are used, another common standard [Daw] is based on the ratios of inner face angles around the vertex:

$$n'_v = \sum_{f \in F(v)} \angle_f(v) \cdot n_f$$
$$n_v = \frac{n'_v}{|n'_v|}$$

where $F(v)$ is the set of faces, adjacent to vertex v and $\angle_f(v)$ is the inner angle of a face f at the corner with vertex v.

2.2.2 Surface Curvature

Another essential component for the shape of a manifold surface is the curvature, which can be derived for each point p on a differentiable surface [PdC76]. Curvature is formally expressed with two principal curvature direction vectors u and v, where:

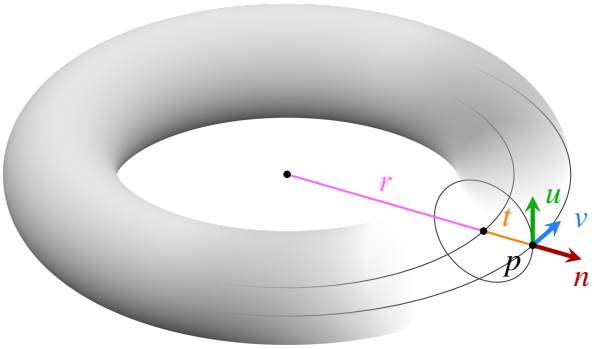
$$0 = u \cdot v \qquad \text{and} \qquad n = u \times v$$

As they are perpendicular to each other and tangential to the surface, they form an orthonormal basis with the surface normal, formally expressed as a matrix:

$$M_p = \begin{pmatrix} n_x & n_y & n_z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{pmatrix}$$

While matrices allow for an intuitive interpretation of orientation, they fall short in many computational challenges. Consequently, orientation matrices are often converted to a quaternion form as described in the upcoming Section 2.2.3.

The two scalars κ_1 and κ_2 specify the maximum and minimum curvature associated with the principle curvature direction vectors u and v, respectively. Curvature is herein defined as the inverse of a circle's radius, which perfectly approximates the surface at a given point. For example as illustrated in Figure 2.1, with the principle curvature for point p on a torus with ring radius r and tube radius t.



$$\kappa_1 = \frac{1}{t}$$

$$\kappa_2 = \frac{1}{r+t}$$

Figure 2.1: Principle curvatures and curvature directions for point p on a torus.

Furthermore, a surface bent inwards has negative curvature, and for a surface bent outwards, the curvature is positive. Their product defines the Gaussian curvature $K = \kappa_1 \kappa_2$, and its sign allows to differentiate between elliptic ($K > 0$), hyperbolic ($K < 0$) and parabolic ($K = 0$) surface points.

Curvature on discrete geometry is usually computed as proposed for triangular meshes [Rus04]: First, a shape tensor is derived for each face using least squares, based on edge vectors and normals of associated vertices. Then, these face-specific tensors are averaged for each vertex from its 1-ring neighborhood, similar to the vertices' weighted average normal computation. Consequently, tensors derived for the tangent frame of a face have to be re-expressed in the tangent frames of each adjacent vertex, respectively.

Concepts introduced in Chapter 5 derive curvature as stated above from meshed hull input as guidance for the volume cell arrangement. However, Appendix 5.B also proposes the possibility for point cloud input, which obviously lacks the topological structures required for the curvature computation defined on meshes. Alternatively, a triangulation of k nearest point cloud neighbors temporarily approximates the 1-ring neighborhood around a vertex for the shape tensor computation. As these orientations only serve as an initial estimate for the alignment of the main procedure of Section 5.5.3, this simple ad hoc solution proved to be an acceptable substitute for actual surface topology.

Another practical application for curvature is to serve as guidance for adaptive mesh resolution: Flat or smoothly bent, low-curvature surfaces can be approximated using larger faces, whereas fine detailed high-curvature geometry requires smaller facets to keep the approximation error minimal. This is indirectly exemplified in Figure 7.13, where our mesh resolution is adaptive to the input resolution, which was importance-sampled based on the object's local curvature. The curvature also allows assessing the smoothness of a mesh or reconstruction, for example, as visualized in the comparisons of Figure 4.20, with (subdivided) meshes of different resolutions.

2.2.3 Quaternions

For a more efficient notation, operational flexibility and numerical stability, rotations and orientations $\in \mathbb{R}^3$ can be expressed using quaternions, formulated as:

$$\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

where a, b, c and d behave like real scalars $\in \mathbb{R}$ and $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ denote abstract symbols with $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$, analogous to complex numbers.

For a unit quaternion with $|\mathbf{q}| = 1$ the corresponding orthogonal matrix is defined as:

$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

$$= \begin{pmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & 1-2b^2-2c^2 \end{pmatrix}$$

The conversion from matrix to quaternion can be formulated in the form of:

$$\begin{aligned} a &= \frac{1}{2}\sqrt{1+m_{00}+m_{11}+m_{22}} \\ b &= \frac{1}{2}\sqrt{1+m_{00}-m_{11}-m_{22}} \cdot \operatorname{sgn}(m_{21}-m_{12}) \\ c &= \frac{1}{2}\sqrt{1-m_{00}+m_{11}-m_{22}} \cdot \operatorname{sgn}(m_{02}-m_{20}) \\ d &= \frac{1}{2}\sqrt{1-m_{00}-m_{11}+m_{22}} \cdot \operatorname{sgn}(m_{10}-m_{01}) \end{aligned}$$

where $\operatorname{sgn}(x)$ gives the sign ($\in [-1, 0, 1]$) of x. For numerical stability, the value in the square root should be clipped to values ≥ 0. Other code-friendly and numerically stable formulations of this conversion feature case distinctions based on the trace of the matrix[1].

2.2.4 Averaging Quaternions

A key advantage in the representation of orientations as quaternions instead of matrices lies in the ability to be interpolatable. A prominent algorithm for this challenge is the spherical linear interpolation (slerp) algorithm [Sho85], which allows linearly interpolating between two given rotation quaternion. In our application, however, we are more interested in averaging several orientations, i.e., computing the mean orientation of

[1] C++ code: http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/

neighboring cells in Section 5.5.3. A robust method [MCCO07] for averaging multiple quaternions q_i, given as an $n \times 4$ matrix Q with a corresponding vector ω of normalized weights w_i is formally expressed as:

$$A = Q^T \cdot Q\omega = \sum_{q_i \in Q} (q_i \otimes q_i) w_i$$
$$A = VDV^{-1}$$

where $\otimes$ is the outer product. The averaged quaternion results from the eigendecomposition of A as the eigenvector $\in V$, which corresponds to the largest eigenvalue $\in D$.

2.3 Optimal Meshes

Snowflakes, honeycombs, or cellular structures all follow some natural order and harmony. Human-made structures are obviously also often purposefully organized and aligned, e.g., dating back to antique mosaic artworks due to their pleasing visual appeal or just out of practical reasons like stability in brickwork bondings or a bunch of oranges stacked on display in a fruit stand. *The Harmony of the World*[2] by *Kepler* is considered one of the first mathematical discussions, using geometry to explain harmony and congruence in nature. This topic can be explored even deeper in group theory with general uniform tilings of mixed shapes or specially optimized tile shapes as in *Escherization* [KS00]. However, with respect to meshes, one is primarily interested in approximating the symmetry and uniformity found in perfectly regular mono-shaped 2D tilings with the three-dimensional faces of a mesh. Consequently, are the criteria introduced in this section to be approached as good as possible.

2.3.1 Uniformity

Figure 2.2 illustrates examples of perfect structures specified for the 2D plane but hard to approach with 3D geometry. Nevertheless, for some applications, the mesh vertices are, at best, subject to certain uniform distributions, allowing for edges of equal length and same-sized faces. A uniform distribution also directly affects the congruence of individual mesh entities, which is explored with more details in the upcoming Section 2.3.2.

[2] *Ioannis Keppleri - Harmonices mundi libri V* (1619)

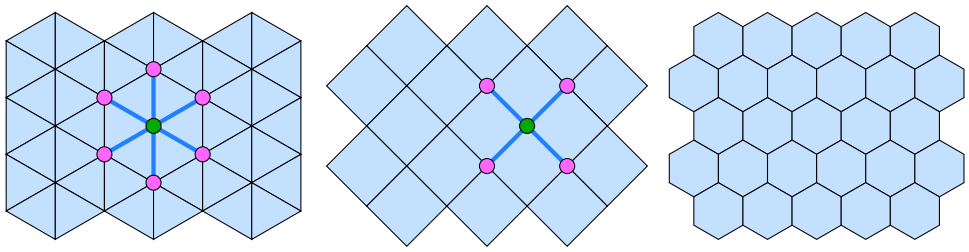


Figure 2.2: The three regular tessellations of the plane with triangular, quadrangular and, for the sake of completeness, hexagonal tiles. Direct adjacent edges of a vertex account for its valence and link to its 1-ring neighborhood, respectively.

A combination of geometrical and topological operations may be required in order to establish true uniform vertex distributions. However, for now, we assume the topology to be suitable and focus on the geometrical operations: These are usually formulated as vertex-individual displacement vectors derived from a local mesh operator, attraction or repelling forces between individual vertices, or as the result of a local or global energy optimization. For example, a simple geometric *Gaussian* smoothing can be achieved with iterative application of an averaging neighborhood displacement vector

$$\Delta v_i = \sum_{j \in i^*} w_{ij}(v_j - v_i)$$
$$v_i' := v_i + \lambda \Delta v_i$$

where i^* is the 1-ring neighborhood of vertex v_i. Weights $0 \leq w_{ij} \leq 1$ may be defined uniformly as $\frac{1}{|i^*|}$ or dependent on the neighborhood geometry, but have to add up to 1. This primitive operation comes with the disadvantage of significant mesh shrinkage when applied multiple times. Alternatively, Taubin [Tau95] proposed to iterate the same displacement operation but with alternating positive λ and negative μ scaling factors where $0 < \lambda < -\mu$. This acts as an effective countermeasure for the shrinkage and is used in this form in our Section 4.5.2 to optimize uneven mesh results.

Our *Equalization* (II) substep in Section 7.3 follows a similar principle, also based on a 1-ring neighborhood i^*, but especially aims for locally equalizing the distances between the individual vertices with the following displacement vector:

$$\Delta v_i = \left(\frac{1}{|i^*|} \sum_{j \in i^*} v_j - l_i \frac{v_j - v_i}{|v_j - v_i|} \right) - v_i$$

where l_i is the average length of all edges, incident on vertex v_i.

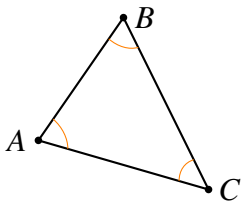
The *Regularization* in Section 5.5.4 aims to establish grid-like structures in an otherwise rather unstructured arrangement of hexahedral cells. As a result, cells are pulled towards the geometric center of *selected* grid neighbors. This directly interacts with the main relaxation scheme, thus the shared influence of individual displacement vectors (geometric regularization $\Delta_g v_i$ vs. relaxation $\Delta_r v_i$) is controlled with a functional expression $\lambda(t)$ to vary over the course of the whole optimization with $0 \leq t \leq 1$:

$$v_i' := v_i + (1 - \lambda(t))\Delta_g v_i + \lambda(t)\Delta_r v_i$$

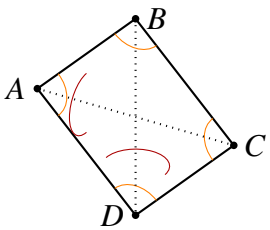
Another prominent algorithm to establish uniform distributions is the Lloyd relaxation which, however, will be further elaborated in the dedicated Section 2.4.

2.3.2 Congruence

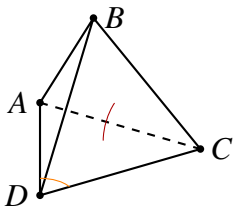
Two elements, equal in size and shape, are called congruent. In our context, the goal is to approach a specified shape, derived from the theoretical congruent optimum, with the individual primitives of a mesh, namely faces of a hull or cells in a volume mesh. Deviations from the following optima are then utilized to measure the quality of a mesh.



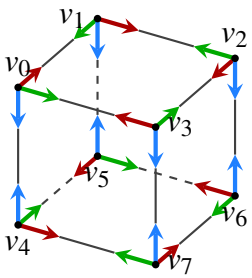
The optimal triangular face is equilateral, thus has three inner angles of $\frac{\pi}{3}$. Overly obtuse or acute inner angles directly coincide with ill-proportioned edge length ratios. In our triangle-meshing pipeline, especially in the *Improvements* (IV) substep in Section 7.3, triangles with ill-proportioned edge lengths, overly obtuse or acute angles are identified and corrected by collapsing or rotating individual edges. These topological operations effectively decrease variations between individual faces and, in combination with geometric equalization, result in much more harmonious meshes, compared to other reconstruction methods (Figure 7.17).



A perfect quadrangular face is a coplanar square with four equilateral edges. But edge-length ratios alone may not always reflect a deformation. For example, parallel shearing or a diagonal fold are only measured robustly, using the angles. Consequently, one also measures the inner angles, which are $\frac{\pi}{2}$ at best. Furthermore, there are also two dihedral angles enclosed between the triangle sides, when divided by a diagonal, which are both π in perfectly flat quad faces. Our quad-mesh assembly routine in Section 4.4.2 materializes well-shaped quad-tiles based on a priority queue. The objective function (Figure 4.11) comprises edge-length ratios as well as angular components. This ensures that high-quality quads, thus similar and of near-optimum shape, are extracted first.



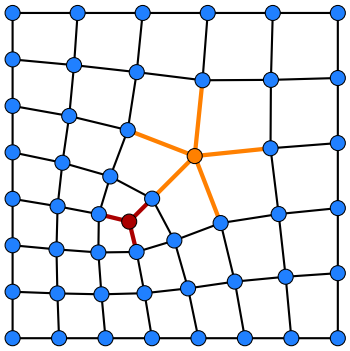
As defined by the minimal requirements for volumetric cells of Section 2.1, a tetrahedron is the topologically smallest possible volume primitive. It comprises four vertices, six edges, and four triangular faces. Common quality measurements are again edge length ratios, inner angles of the (at best equilateral) triangles ($\frac{\pi}{3}$) and the dihedral angles ($\arccos\frac{1}{3}$) on edges, enclosed between two faces. Furthermore, one can also measure the ratio of edge lengths, compared to the radius of the tetrahedron's circumscribed sphere as radius-edge-ratio $\frac{r}{a}$. The optimal tetrahedron, with the four vertices placed equidistantly on a unit-sphere ($r = 1$), has edges of length $a = \sqrt{8/3}$. For our utilization of tetrahedral meshes in Chapter 6, the shape of individual primitives is actually not relevant, as our computations are invariant to the partitioning for linearly varying volumetric fields (Appendix 6.D). Nevertheless, congruent tetrahedra would guarantee evenly spaced vertices, i.e., spatial sampling positions for the density fields, thus improve the approximation accuracy of mass property computations in non-linear density fields.



$$\mathrm{MSJ} = \min_{\forall v_i} \begin{vmatrix} \vec{e}_0^{\,T} \\ \vec{e}_1^{\,T} \\ \vec{e}_2^{\,T} \end{vmatrix}$$

Hexahedral mesh structures are, besides tetrahedralizations, the most commonly demanded form of volume meshes. General polyhedra, topologically larger than hexahedra, are ill-suited for interpolation or integration tasks, but so are heavily deformed hexahedra. Consequently, one is most interested in generating hex-dominant meshes consisting of well-shaped primitives. A popular quality measure is the skewedness of a primitive, formulated with the Jacobian of a corner vertex as shown on the left. Following the right-hand-rule, the three outgoing edge direction vectors $\vec{e}_0$, $\vec{e}_1$ and $\vec{e}_2$ of a vertex (with valence 3) are stacked in a 3×3 matrix, where the determinant gives the Jacobian. The edge vectors are usually scaled to unit length, and, per primitive or mesh, only the minimum value over all vertices is of interest. Therefore, the quality of hex-meshes is often expressed with the Minimum Scaled Jacobian (MSJ). For a perfect rectangular cuboid, the MSJ equals 1, the maximum possible value. Furthermore, this measure indicates (when negative) if the primitive features inverted corners, thus being an inverted primitive and, therefore, not robustly integrable. The MSJ is used as a thresholded priority criterion in our Section 5.6.2 to consider only high-quality primitives in the volume mesh assembly routine. The MSJ is undoubtedly a strong indicator for the usefulness of a result but only points out the single worst part of a mesh. For example, a mesh of considerably high quality with one bad vertex would rank lower than a mesh with overall mediocre elements. Thus, one might want to consider the Average Scaled Jacobian (ASJ), as it better represents the quality distribution over all elements.

2.3.3 Regularity

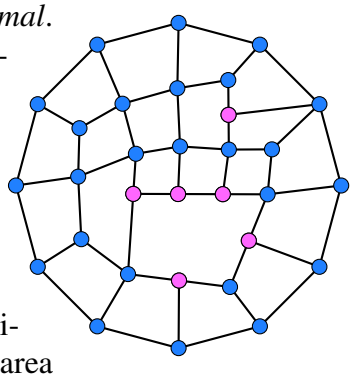


As described in the introduction, a mesh can be seen as a graph and, apart from the geometric positions of vertices, topological relations within the mesh also become a measurable characteristic. We borrow notations from graph theory and observe a mesh as a set of vertices and a set of edges $G = (\mathcal{V}, \mathcal{E})$. The degree (or valence) of a vertex is defined by the number of incident edges

$$\deg(v_i) = |\{e \mid i \in e,\ \forall e \in \mathcal{E},\}|$$

with $\delta(G)$ as minimum and $\Delta(G)$ as maximum valence in the mesh, respectively. Optimally, the valence is equal for all vertices throughout the mesh. As illustrated with the 2D example in Figure 2.2, for a perfectly regular triangular tiling the valence of each vertex is 6 and for pure quadrangular tilings it is 4. With three dimensions, however, one quickly finds that a simple cube solely consists of six perfect quadrangular faces, but all eight corner vertices are only of valence 3. In hexahedral volume meshes, the valence of internal vertices should be 6 as there are direct neighbors on top, bottom, left, right, front and back. Vertices residing on the outer hull of a hex-mesh are missing internal edges, thus their valance can be 3, 4, or 5. Moreover, in quad meshes, irregular vertices with valence 3 or valence 5 denote a singularity but are often necessary to direct the mesh flow. As variations in the valence are unavoidable, it is therefore desirable to, at least, keep the difference $\Delta(G) - \delta(G)$ minimal and avoid extreme outliers.

2.3.4 Conformity



Meshes, like the one included above, are called *conformal*. The image on the right illustrates an also seemingly quadrangular domain partitioning. However, this constellation is considered *non-conformal* as several vertices actually denote T-junctions. Configurations like this are problematic in multiple ways: **i**) In a purely quadrilateral mesh, T-junctions violate the validity constraint from Section 2.1.3, that an edge must be adjoined by exactly two faces. **ii**) Assuming a quad-dominant mesh, featuring triangular faces, T-junction edges can be interpreted as zero-area faces squashed to a straight line. This, however, is also a rather poor solution, considering the goal of congruent elements as defined in Section 2.3.2. **iii**) Another valid but also unfavorable option is to interpret affected faces as general polygons, where three or more vertices just happen to be colinear. Our meshing procedures, proposed in Chapter 4 and 7, produce only *conforming*, fully quadrangular or triangular results.

With volumetric meshes, conformity is also of utmost importance. General hex-dominant meshes usually fall back on case **iii**, as described before, and feature arbitrarily large polyhedra to circumvent the topological conflicts arising from T-junctions. A key concept of our novel mesh class, introduced in Chapter 5, is to only allow for primitives of at most hexahedral shape. Although general T-junctions do not occur in at-most-hexa scenarios, some cases still require special attention: E.g., the topological transition from purely quadrangular hexahedra to primitives, which also comprise triangular faces, like prisms or tetrahedra. The interpretation of two adjoined triangles facing a quadrangle may vary, dependent on the application. However, a common understanding [LA, BTP*19] is that faces unopposed by another face are *open*, thus mark the outer hull of a volume mesh. Internal faces are en*closed* by, or are part of, exactly two primitives. Consequently, two triangles facing a quadrangle would be considered as part of the hull.

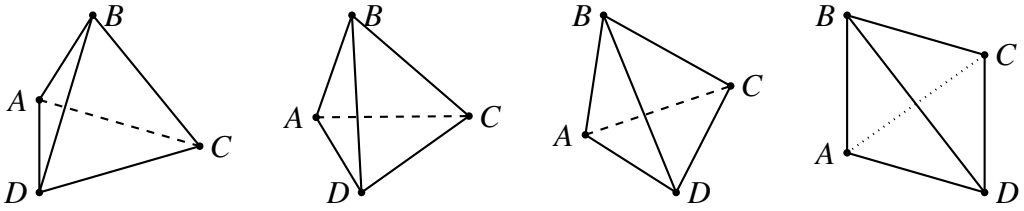


Figure 2.3: From left to right: A tetrahedron is squashed to a sliver, i.e., an ideally flat primitive with minimal volume, comprising either two triangles and a quadrangular face or two triangles per side with opposing diagonals.

The introduction of *slivers* is a common workaround [BLC16, SRUL16] to formulate conformal meshes, analogous to case **ii**. As illustrated in Figures 2.3 and 2.4, a tetrahedron or prism can become *sliver* elements with minimal volume by squashing them flat. These primitives are inserted, where required, to provide suitable opponents to internal (but falsely open) faces. Special at-most-hexa slivers are formulated in Appendix 5.A.

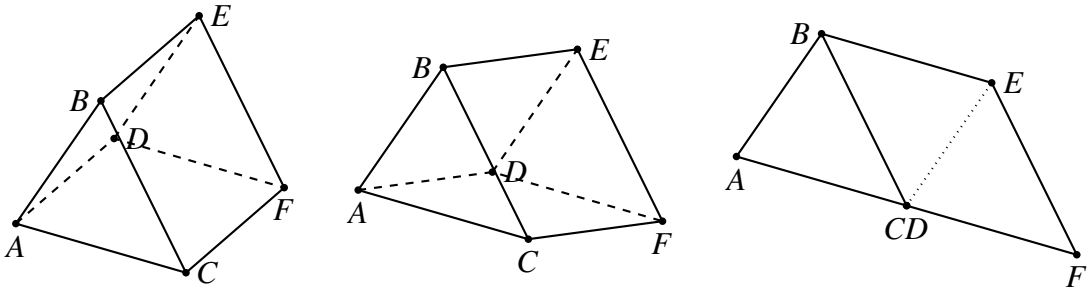


Figure 2.4: From left to right: A prism is squashed to a penta-sliver, also an ideally flat primitive with minimal volume and five vertices ($C = D$). It comprises a triangular and quadrangular face on each side, with real edges for the opposing quads' diagonals.

2.4 Lloyd Relaxation

The general Lloyd relaxation [Llo82] is an algorithm to iteratively reposition points of a set with the goal of an evenly spaced distribution. It shares similarities with k-means clustering and can also be applied to minimize quantization error [Max60]. However, whereas k-means aims for clustering a given set of observations, Lloyd's algorithm operates more freely and relies only on the space partitioning provided by a Voronoi diagram.

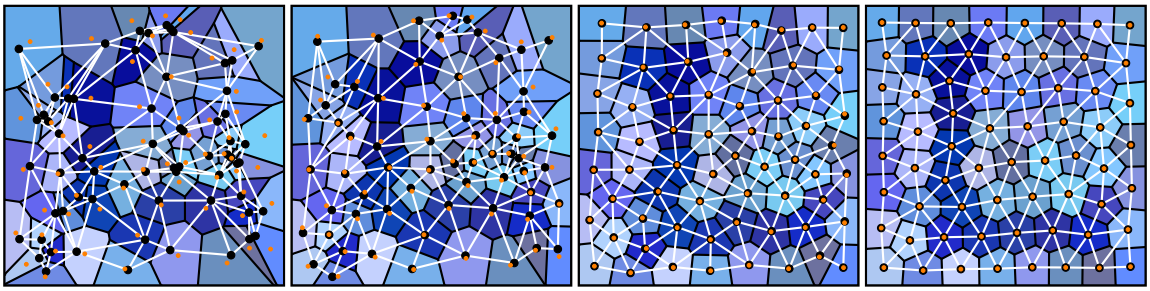


Figure 2.5: An L_2 Lloyd relaxation after 0, 1, 10 and 100 iterations. The Voronoi diagram is shown with colored cells, the corresponding Delaunay graph in white. Black dots denote the sites of the diagram, actual cell centroids (new site positions) are highlighted in orange. Movement magnitudes are the strongest in the first few iterations and then steadily decrease. After just 10 iterations, the sites and cell centroids almost coincide.

The example in Figure 2.5 illustrates a relaxation, starting with a set of randomly positioned points (sites) and quickly converges on a CVT. Results of this algorithm are valuable, not only for sampling or vector quantization tasks but also suitable for geometric mesh smoothing. A Lloyd iteration alternates between the two following steps:

I. Compute a Voronoi diagram based on given sites. The diagram specifies the spatial cluster of all *closest points* to a certain site s_i as a cell c_i, formally defined as:

$$c_i = \left\{ p \in P,\ d(p,s_i) \leq d(p,s_j)\ \forall\ i \neq j \right\}$$

where (P,d) is a metric space with the distance function d. Fortune's sweepline algorithm [For87] for computing Voronoi diagrams in a Euclidean plane also allows for site-specific weights, which influence the distance function and resulting cell-size. An alternative approach is the Bowyer-Watson algorithm [Bow81, Wat81], which first computes a Delaunay graph in any number of dimensions and then extracts the Voronoi diagram as its dual. This relies on the relationship between Voronoi diagram and the Delaunay graph, which states that if cells c_i and c_j are adjacent in the Voronoi diagram, there is an edge (s_i, s_j) connecting the two sites in the Delaunay graph. Furthermore, the line (2D) or plane (3D) separating two cells c_i and c_j is an orthogonal bisector to the edge connecting the sites s_i and s_j, at least in Euclidean space.

II. Compute the geometric center of each Voronoi cell and reposition the sites accordingly. The new centers are obtained via integration over the volume of each cell. In Euclidean 2D or 3D space, cells are convex polygons or polyhedra, thus the new sites can also be expressed as a weighted sum of primitive simplex centroids:

$$s_i' = \frac{1}{|c_i|}\int_{p\in c_i} p \qquad\qquad = \frac{1}{|c_i|}\sum_{\mathbf{c}_j\in c_i} |\mathbf{c}_j|\dot{\mathbf{c}}_j$$

where $\dot{\mathbf{c}}_j$ denotes the centroid, and $|\mathbf{c}_j|$ the weight (area or volume) of a cell simplex $\mathbf{c}_j$. Simplices result from the trivial triangulation of a cell and its site as triangles (2D) or tetrahedra (3D), respectively.

2.4.1 Metric Spaces

The following denotes the common general definitions used in topology [Dug66, Kle89], where a *metric* on set P specifies a function $d : P \times P \to \mathbb{R}_{\geq 0}$, satisfying the three axioms:

$d(a,b) = 0$	iff $a = b$
$d(a,b) = d(b,a)$	symmetry
$d(a,b) \leq d(a,c) + d(c,b)$	triangle inequality

The actual *norm* of a real k dimensional space, is then given by a measure $N : \mathbb{R}^k \to \mathbb{R}_{\geq 0}$, specifying a vector's length such that:

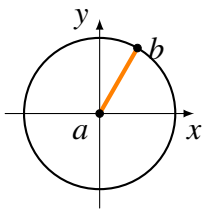
$$N(u+v) \leq N(u) + N(v)$$
$$N(\lambda v) = |\lambda| \cdot N(v)$$

However, the terms *space*, *metric* and *norm* are often used as synonymous expressions. Notable applications of the Minkowski L_p norm family, generally defined as:

$$L_p(v) = \left(\sum_{i=1}^{k} |v_i|^p\right)^{\frac{1}{p}}$$

with $p \geq 1$, are in the following directly specified with a corresponding distance function d_p. When used in relaxations, these different L_p norms allow for distinctive patterns to emerge in the Voronoi diagram and, therefore, also in the resulting CVT.

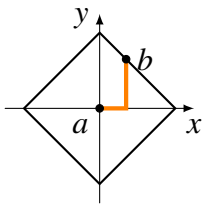
Euclidean The (informal) *default* context wherein Voronoi diagrams, thus Lloyd relaxations, are usually used is the Euclidean L_2 space. Consequently, the associated metric is simply the Euclidean distance:



$$d_2(a,b) = \sqrt{\sum_{i=1}^{k}(b_i - a_i)^2}$$

With regard to a fixpoint a, the level set of this function, i.e., all points b for which d_2 is constant, are located on a sphere or, as shown in the 2D example above, a circle. Following the *Kepler conjecture* about dense sphere packing in Euclidean space, the relaxed CVT will approach a regular face-centered cubic (fcc) lattice. As exemplified in Figure 2.5, the 2D analog approaches a distinctive hexagonal layout [CW10] and, therefore, a very regular triangulation with the Delaunay graph.

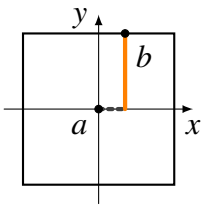
Manhattan When discussing L_p norms, the Manhattan L_1 norm also deserves an honorable mention, although being not directly relevant in our further context. With $p = 1$ the distance function simply computes as the summation of absolute differences between the Cartesian coordinates of two points:



$$d_1(a,b) = \sum_{i=1}^{k} |b_i - a_i|$$

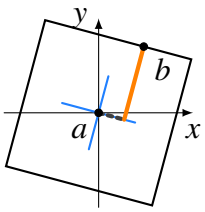
The naming of this norm also called *taxicab* or *cityblock*, reflects on the travel distance between two points in a street grid layout, e.g., as prominently known from Manhattan Island. As the 2D level set is rectangular, Hausner [Hau01] proposed to employ this norm in a Lloyd relaxation to simulate decorative mosaic tilings.

Chebyshev Whereas other L_p norms also have very interesting properties, the upcoming chapters will primarily focus on the Chebyshev L_∞ norm. It is sometimes also referred to as the max-norm, as the corresponding distance function d_∞ is defined as the maximum absolute difference between the Cartesian coordinates of two points:



$$d_\infty(a,b) = \max_{i\in[1,\dots,k]} |b_i - a_i|$$

The 3D level set of this metric is a cube, i.e., a square in two dimensions. However, the main distinction between this norm and the Euclidean is that it is no longer isotropic. To generalize this further, an orthonormal orientation M_a is specified for each point a, independent of the prevalent world space. The distance computes as the maximum absolute difference over the orientation-projected Cartesian coordinates:



$$d_\infty(a,M_a,b) = \max_{i\in[1,\dots,k]} |(M_a \cdot (b-a))_i|$$

With 3D cells, such relaxations eventually approach hexahedral structures, as long as the orientations are sufficiently homogeneous. Therefore, individual orientations must either follow a smooth given frame field or align to each other during the optimization, such that adjacent cells are oriented similarly, however, indifferent to axis permutations.

2.4.2 In Application

As mentioned in the introduction, known algorithms are able to compute L_2 Voronoi diagrams for any number of dimensions, but abstract generalizations are only theorized for the 2D plane [Kle89]. The exhaustive studies of Lévy and Liu [LL10] approach 3D formulations of L_p-CVTs with large p values, to approximate the L_∞ norm. However, to the best of our knowledge, there is currently no efficient algorithm to compute a general Voronoi diagram under the L_∞ norm. The implementation available in CGAL [CDP21] is able to generate 2D L_∞ diagrams for points and line segments, but only for uniformly axis-aligned orientations of all sites. Alternatively, our three-dimensional application of the L_∞ algorithm in Section 5.5.3 relies on a labeled high-resolution volume grid. The integral computation for new cell centroids is approximated by averaging all labeled

grid positions. Our experimental approach in Section 6.4.5 provides an analytic solution for the centroid integration, formulating the Chebyshev distance with a density domain (Figure 6.10). Although it converges on hexahedral cell structures, the algorithm is not yet a full L_∞ relaxation. It is actually an $L_{2|\infty}$ hybrid, as the first step of the iteration still relies on a computable but orientation invariant L_2 diagram. Extensions to also compute an accurate L_∞ diagram will be explored in dedicated future work.

2.5 Evaluating Reconstructions

The definitions introduced so far are mainly focused on measuring the geometrical and topological characteristics of a mesh. As described in Section 3.2, another key capacity of any signal reconstruction pipeline is the result's accuracy with respect to the input. In the following, we will introduce common quality measures used to compare a known ground truth mesh X to a reconstruction Y. For point cloud meshing applications, this usually relies on a simulated scan with sampled input points from a given object. Artificial scans also allow for control over certain artifacts in the sample data like non-uniform sample distribution, missing data, outliers, or the magnitude of noise perturbation.

2.5.1 Hausdorff Distance

A common method to evaluate the quality of a reconstruction is the Hausdorff Distance (HD). The measure on *how far* two shapes are from each other is generally defined as:

$$d_{HD}(X,Y) = \max\left\{ \sup_{x\in X}\inf_{y\in Y} d(x,y),\ \sup_{y\in Y}\inf_{x\in X} d(x,y) \right\}$$

The distance function d is the Euclidean distance, thus the result directly depends on the compared shapes' scale. As a result, ground truth and reconstruction are usually centered and scaled to fit in a unit cube, based on the largest bounding box extents. This effectively limits the maximum possible HD to the unit cubes diagonal with length $\sqrt{3}$.

As per definition, d_{HD} comprises the supremum infimum over all points in both sets. However, when dealing with meshes, the evaluation of *all* points on the surface is not practically feasible. Consequently, the most commonly used approach is a sampling-based approximation [CRS98, CCC*08]. Let X' and Y' define two sets of uniformly sampled surface points from the input ground truth and reconstruction, respectively. With d_Δ as the minimal point-to-triangle distance for a given sample and mesh, the symmetric Hausdorff Distance is then formulated as in the following:

$$d_{HD}(X,Y) = \max\left\{\max_{x' \in X'} d_\Delta(x',Y),\ \max_{y' \in Y'} d_\Delta(y',X)\right\}$$

As the computation relies on discrete sample positions, there is also sampling bias. Different sample distributions cause variations in the results, which, however, can be approached by increasing the number of samples, i.e., linearly dependent on the object's surface area. If there is no ground truth mesh given, i.e., only a point cloud X^*, one can still use the Hausdorff Distance to compare different reconstruction methods, competing against each other. The Hausdorff Distance is a very rigorous concept where even a single flaw in an otherwise totally accurate reconstruction will result in a high error.

2.5.2 Mean Distance

The mean distance [BLN*13] between to shapes also comprises symmetric closest point-to-triangle distances based on sampled meshes but computes their average instead of the absolute maximum value:

$$d_\mu(X,Y) = \frac{1}{|X'|+|Y'|}\left(\sum_{x' \in X'} d_\Delta(x',Y) + \sum_{y' \in Y'} d_\Delta(y',X)\right)$$

This concept is a much softer error measure, as single flaws are leveled out by the average computation and, therefore, not penalized as hard as with the HD.

2.5.3 Root Mean Squared Error

Similar to the mean distance is the root-mean-squared-error (RMSE) also well suited to express the overall closeness of two shapes, rather than just identifying the worst part. But in contrast to the other measures it is often used asymmetrically [DMSL11]:

$$d_{RMSE}(X,Y) = \sqrt{\frac{1}{|Y|}\sum_{y^* \in Y} d_\Delta(y^*,X)^2}$$

where y^* are triangle barycenters of all faces in mesh Y. Therefore, the RMSE is solely based on the reconstruction geometry without the need for sampling. With this specification, the mesh Y could be quite jagged and rippled but as long as all barycenters are $\in X$ the error will be 0. Nevertheless, this loophole is trivially covered by also including the actual vertices of Y in the computation.

2.5.4 Normal Angle Deviation

Scenarios where the reconstruction Y is riddled with noise but fluctuates close enough to X will not lead to significant errors with distance-based measures. The Normal Angle Deviation (NAD) allows to quantify such inaccuracies by measuring the angle between surface normals of closest point pairs from ground truth and reconstruction.

$$d_{\angle}(X,Y) = \frac{1}{|X'|+|Y'|}\left(\sum_{x'\in X'} \angle(x',Y) + \sum_{y'\in Y'} \angle(y',X)\right)$$

where $\angle(p,Q)$ computes the angle between the normal at point p and the face normal at its closest point $q \in Q$. This allows to express and measure the reconstruction accuracy not only in a local sense of distance but actual surface similarity.

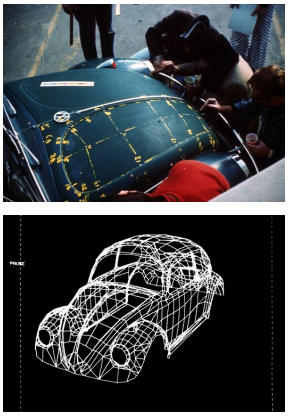
2.5.5 In Application

As mentioned before, each error measure comes with its positive and negative aspects, respectively. Thus, the different measures allow for examining and highlighting different characteristics of a reconstruction pipeline. As a result, they are usually listed separately, like in our bulk comparison against state-of-the-art reconstruction methods in Section 7.5.1. This comparison, furthermore, features a Surface Area Deviation (SAD) and Total Volume Deviation (TVD), which simply express reconstruction deviations from the ground truth in percent, given as the average over all objects. As a mesh is either closed or not, watertightness is a binary attribute. Therefore, WT comparisons usually account for the overall percentage of reconstructions that ended up as a closed hull.

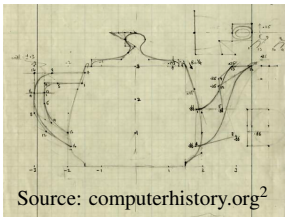
Chapter 3

History and Related Research

The recreation of real objects as digital models is a fundamental subject in the field of computer graphics. The earliest models were already created in the 1970s, using manual and analog 3D scanning techniques: *Catmull's hand* was digitized in 1972, based on a plaster mold of the hand with 350 hand-drawn triangles [Pri08] on it. These triangles and their corners were then measured and meticulously cataloged. In the same year, *Sutherland's Volkswagen* was "scanned" in a similar manner, where manual markers drawn on the car were measured by hand, then mapped and encoded in a digital file. The procedure is shown on the right, with the resulting mesh as a wireframe rendering.



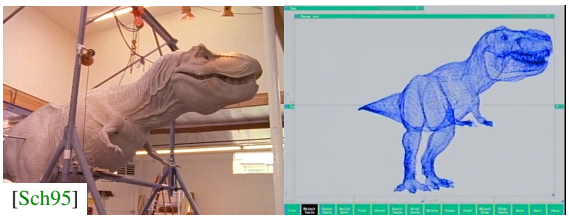
Source: computerhistory.org[1]



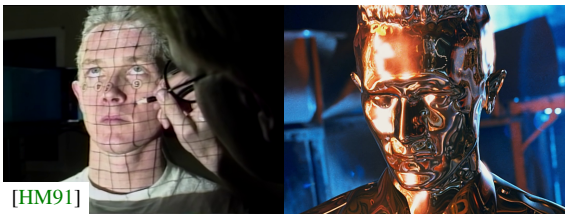
Source: computerhistory.org[2]

One of the first modeled objects in computer science is the famous *Utah Teapot*, described in 1975 using Bézier-patches derived from hand-drawn sketches. Today, objects used in games, commercial illustrations, or animated (parts of) films are often also modeled and digitally sculpted by skilled artists. This process allows for unprecedented amounts of creative freedom in object, character, and scene design. Further advancements of this technology laid the foundation for computer-aided design (CAD), nowadays used in almost every digital design pipeline. From architecture to production and manufacturing design, CAD is now the industry standard to digitally specify 3D geometry for simulations or technically accurate visualizations.

[1] https://www.computerhistory.org/revolution/computer-graphics-music-and-art/15/206/560
https://www.computerhistory.org/revolution/computer-graphics-music-and-art/15/206/558
[2] https://www.computerhistory.org/revolution/computer-graphics-music-and-art/15/206/556


[Sch95]


[Mar93]


[HM91]

Visual effects have a long tradition in cinema, dating as far back as the 19th century with the first stop-trick in *The Execution of Mary Stuart* (1895). The first-ever dinosaurs moved on screen in *The Lost World* from 1925, using stop-motion animated miniatures. In 1993, *Jurassic Park* marked a milestone for computer-generated imagery (CGI) used in film, by blending life-sized robotic creatures with fully rendered digital models. For bridging the different technologies as realistically as possible, a crucial step was to 3D-scan the detailed, handcrafted models [Sch95] and then rebuild them for animation. These techniques, pioneered by *Cyberware* and *Industrial Light & Magic*, were prior also used for the time-travel sequence in *Star Trek IV* (1986), the water-pseudopod in *The Abyss* (1989), and the liquid-metal T-1000 in *Terminator 2* (1991). Various static facial expressions of the actors were laser-scanned and then morphed for animating the transforming, liquid characters [Mar93, HM91]. Modern techniques go even further and capture not only the geometry or appearance of inanimate objects or scenes but also record movements via motion tracking, all together allowing for the visual wizardry we are nowadays so used to see.

Nevertheless, the classic challenge to reconstruct real-life objects still remains and has many possible fields of application; to name a few: In a medical context, computed tomography (CT) scans and techniques like magnetic resonance imaging (MRI) generate discrete volumetric representations of the internals of a body, revealing skeletal structures and various kinds of soft tissue. However, in recent years the common external 3D scans also gained popularity in medicine [HJ19], e.g., for studying human body metrics or designing patient-specific facial prostheses. In industrial applications, 3D scans and reconstructions may be utilized for production control, reverse engineering, or fast prototyping. For both medical and industrial but also entertainment purposes, it will probably soon be a valuable extension to virtual and augmented reality scenarios. Furthermore, 3D scanning is a commonly used tool to analyze historical artifacts and digitize precious artworks. With a history of multiple decades, the reconstruction of 3D scanned objects is a well-studied subject and also the focus of this thesis.

3.1 3D Scanning and Point Clouds

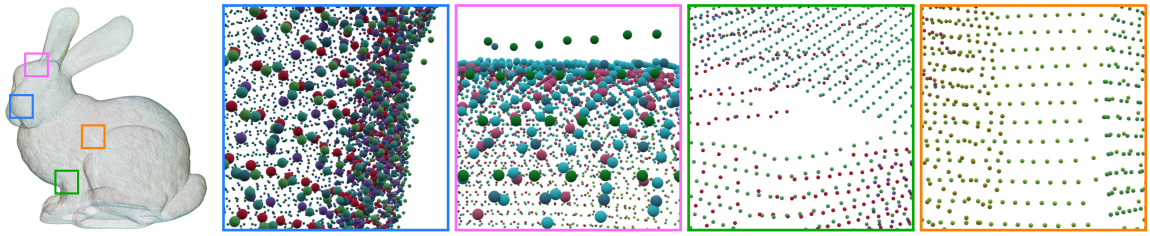
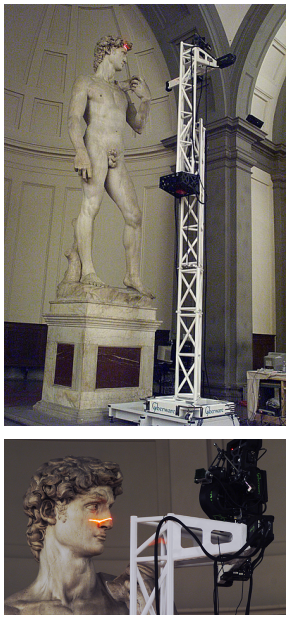


Figure 3.1: The original point cloud of the *Stanford Bunny* [TL14] is a composition of 10 individual partial 3D scans (visualized with differently colored samples). The shown examples illustrate the most common artifacts in 3D scanning data: Noise (especially in overlap regions), outliers, missing data and varying sample density.

The process to gather digital information of an object's shape for reconstruction starts with a 3D scan. Many popular 3D test objects used in computer graphics originate from such scans, like the *Bunny* shown in Figure 3.1. Other scanning techniques may also recover an object's appearance [IKL*08], CT or MRI scans reveal the interior of solid bodies or range-imaging describes the depth of a scene, but our focus lies on the surface's geometry. The early methods for 3D scanning, mentioned before, relied on markers applied to the object, which are then manually traced, measured, and cataloged. This only allows for very coarse approximations of the object and, furthermore, is not always possible: Surfaces of organs or bones are, at best, not directly visible and the surfaces of precious artworks and cultural artifacts usually should neither be touched nor altered. While there also exist contact 3D scanners, which rigorously poke the object to measure 3D positions, the majority of 3D scans are usually captured with non-intrusive methods. This may include the use of modulated or structured light [GRL17] where information, encoded in grids or patterns, is projected onto the object and again captured by a camera. The resulting deformation of the projection then allows computing the 3D structure of the surface. So called time-of-flight scanners systematically probe a scene with laser pulses and measure the round-trip time for the impulse to return, thus the result's precision is only limited by the accuracy in time measurement. Image-based methods [LKG*01, LKG*03] do not necessarily require special lighting but can utilize multiple observers to derive 3D positions, normals, and appearances of points captured from different angles. There are also specialized methods to recover *shape from shading* by solving for the inverse of a specific shading model from a single image. Machine learning-based approaches may also produce reasonable guesses for depth information or the shape of an object from a single viewpoint.

Related to the multi-observer methods, handheld 3D scanners are also based on triangulation to infer 3D positions of points, found in multiple images of a static scene, captured over time by one moving observer. Furthermore, triangulation is also used in the context of high-precision laser scanners [DAL*11] to derive positions where a laser ray and diffracted laser plane intersect the object's surface. The rig, build for the ambitious *Digital Michelangelo Project* [LPC*00] is shown on the right. It combined a laser, range- and color-camera to infer 3D points along with their surface color simultaneously. Methods like the multi-camera triangulation produce better results on textured materials as individual points can be distinguished and matched more precisely. On the other hand, structured light approaches work more robustly with homogeneous materials because there is no interference from varying surface materials. Optical methods often assume an, at least, opaque and non-reflective surface. However, translucent [GLL*04] or mirroring objects [TLGS05] can also be reconstructed with specialized methods [IKL*08].



The Digital Michelangelo Project[3]

The retrieved sample data is encoded as simple three-dimensional coordinates in one or multiple partial point clouds, depending on the scanning method. Whereas the *Bunny* from Figure 3.1 composes from 10 individual scans, the 5.17 m tall statue of *David*, shown above, was captured with 480 individually aimed scans. A known observer position may give an initial estimate on the partial scans' relative poses, further alignable with the Iterative Closest Point (ICP) [RL01] algorithm. Without any pose estimate, partial scans can be aligned, nevertheless, using global registration [GMGP05]. Some methods also allow for acquiring, or at least robustly estimating, individual normal vectors, pointing outwards from the surface at each point, respectively. If dense enough, this representation as an oriented point cloud is already sufficient to render an object [LMR07] by interpreting each point as a small disc, oriented orthogonal to the corresponding surface normal. Nevertheless, the majority of downstream applications usually expect 3D models to be expressed as a surface, usually a mesh.

While there are many different forms and approaches of surface reconstruction [BTS*17], we will mainly keep the scope of this thesis on point-cloud-to-mesh techniques.

[3] https://accademia.stanford.edu/mich/more-david/more-david.html

3.2 Surface Meshing

Considering the point samples gathered from the scanning process, one is now challenged with the task to reverse-engineer its original shape, i.e., formulate a surface, most faithfully approximating the given data. Functions, isosurfaces or parameterized curves can describe types of differentiable surfaces but also require more effort in terms of handling, storing and modifying an object. Compared to that, a discrete surface representation in the form of a mesh is by far not the most accurate solution but presents a reasonable compromise with more flexibility in application. Consequently, polygonal meshes are currently the most commonly used data structure for 3D objects, with millions of models available online [ZJ16, KMJ*19].

Surface meshes can be generalized to feature arbitrary polygonal faces, but the most commonly used variants feature triangular- or quadrangular-faces; sometimes also mixed with an emphasis on as many quads as possible, then called quad-dominant meshes. General mesh definitions are formulated in Section 2.1.

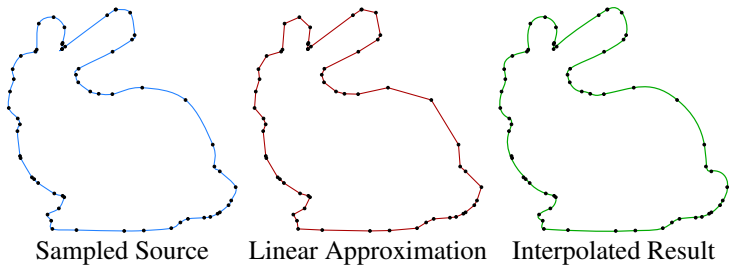


Figure 3.2: 2D example of a signal reconstruction problem. The source is sampled at discrete positions with varying sample density; linearly connecting the points only poorly approximates the original shape; a constrained interpolation of the given data allows for more accurate results.

The illustration in Figure 3.2 visualizes the scanning process on a 2D example: The original shape is measured with samples, which are located at discrete positions that may vary in spatial density, can be disturbed by noise, feature holes, or, in case of merged partial scans, have misaligned overlap regions. Simply *connecting the dots* as in a children's puzzle would be fairly easy if they were numbered, but they are not. Furthermore, the objects are three-dimensional, therefore, require triangles instead of lines to specify a hull. A possible approach to interconnect the points with faces is to take a sphere with a fixed radius, move it around the point cloud and create triangular faces whenever three points simultaneously reside on the sphere's surface. To give the procedure some more structure, the ball starts on an initial triangle and then pivots over one of its edges until

it finds a new third point, thus incrementally gathering more and more surface triangles. Whereas result meshes of this *Ball Pivoting Algorithm* (BPA) [BMR*99] may feature every single input point, the result is a quite linear approximation, which is rarely the best choice in a signal reconstruction task. Similar results, incorporating the input itself, can be generated using Voronoi cells grown around the sample points. The extracted Delaunay complex then directly specifies the triangle interconnections between the samples, but the outcome still heavily depends on a chosen search radius for the cells. As a result, these methods, called *Alpha Shapes* [EM94], tend to struggle with topological noise as there is not *the* one radius for an arbitrary point cloud.

While these approaches try to connect the given points, solutions like the popular *Poisson Surface Reconstruction* (PSR) [KBH06] aim to formulate an isosurface of the subject's hull, merely guided by the oriented sample points. Normals of the point cloud are used in a least-squares manner to specify an indicator function, which is able to differentiate between *in-* and *outside* of the object. By design, PSR clusters the point cloud in an octree structure. Therefore, octree leaf nodes cells can be individually materialized using the *Marching Cubes Algorithm* (MCA) [LC87]. The general idea of the MCA technique can also be easily applied to various other meshing tasks, and it basically comes down to the following algorithm: First, the space enclosing the shape of interest is subdivided with a grid structure or, in the case of PSR, with an octree. Either way, axis-aligned cubical cells are categorized depending on how they intersect with the implicit surface. A catalog of all possible intersection variations then provides a suitable triangle constellation to be inserted for each cell, respectively. Whereas an initial isosurface, describing the hull, is not yet affected by any discretization measures, the resulting MCA meshes clearly are. The triangles inserted for two adjacent cubical cells are simply connected with a shared edge. This causes the distinctive grid structure prominently visible in the resulting meshes. In the case of differently sized octree cells, this may also cause abrupt jumps in mesh resolution. Consequently, the aggregation of small partial solutions leads to a somewhat irregular and non-uniform result mesh. Still, PSR, or its derivatives, in combination with MCA are very popular as it is a very robust method, and the listed downsides can sometimes be neglected, dependent on the application's context.

Suitable interpolation [VPBM01], smoothing [Tau95] or subdivision [Cat74, Kob00, VPBM01] steps can significantly improve upon coarse or uneven meshing results. However, an optimal reconstruction mesh should be designed to have the faithful interpolation of the input as its top priority. Furthermore, it should be robust to noise and outliers, try to minimize reconstruction errors, all while maintaining appropriate levels of smoothness, sharpness, and watertightness. More vertices, smaller triangles, thus an overall higher resolution of the approximating mesh allow for smooth, curved regions and fine details but would unnecessarily blow up the memory footprint of a mesh with mostly flat surfaces. This conflict can simply be approached, as in most signal compression techniques, with an adaptive resolution. On the other hand, equally spaced vertices may also

be a desirable attribute for the resulting mesh. Strong features like creases or corners in the implicit input, can be reconstructed by aligning vertices and edges of the approximating mesh accordingly. Furthermore, the mesh's edge-flow can be aligned to the input features, i.e., with the orthogonal edge structures in quad-meshes oriented to the (also orthogonal) principle curvature directions of curved surfaces.

While specialized remeshing algorithms [HZM*08, ZHLB10, LHJ*14] can turn low-quality triangular meshes into nicely feature-aligned quad-meshes, this intermediate stage could be omitted with quad-meshes directly extracted from the point cloud. The approach of *Instant Field Aligned Meshes* (IFAM) [JTPSH15] is able to do so by operating on a graph, either trivially given as an input mesh or as an oriented point cloud where adjacency information is temporarily established within a nearest-neighborhood. Positional- and rotational symmetric fields are locally optimized for each node in the graph, i.e., the vertices. Superimposed integer grid maps then specify final vertex positions, further optimized by a greedy graph-collapse algorithm. Due to the rotation field optimization, the edge flow of the resulting quad-dominant mesh nicely aligns with curved shapes and dominant features. *Field-Aligned Online Surface Reconstruction* (OSR) [STJ*17] is based on the same concept but aims to reformulate the scanning and reconstruction challenges as one interactive process. Both of these concepts excel in terms of feature alignment but tend to create holes if samples are too sparse.

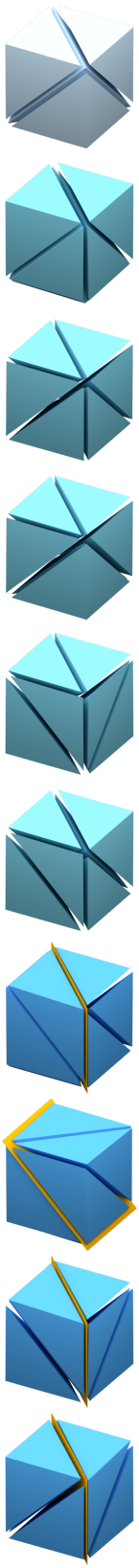
In computer science, and especially graphics, the developments over the most recent years were dominated by approaches utilizing some form of machine learning. For some time, the unorganized nature of 3D data, i.e., point clouds, was quite an obstacle for convolutional neural network (CNN) based approaches, as they usually rely on formulations with discrete blocks, grids, kernels, and tensors. However, novel methods like *Occupancy Networks* [MON*19] propose a draft on how 3D data could be incorporated in a learning network architecture and *Scan2Mesh* (S2M) [DN19] already allows for the reconstruction of truncated signed distance fields (SDF). Here, the network first creates an initial guess for sets of vertices and edges, from which a dual graph is constructed, eventually allowing for the extraction of the final mesh's faces. However, the results demonstrated in the paper only feature objects of 8 distinct classes on which the network has to be specifically trained and is then only able to reconstruct objects from. While this might be sufficient for a specialized target application, i.e., exclusively reconstructing furniture from room interiors, it does not generalize very well to an arbitrary 3D scan reconstruction task. A more general concept is proposed with *Point2Mesh* (P2M) [HMGCO20], which can reconstruct point clouds of an arbitrary shape from an unknown object class without prior training. This is achieved as the network learns from the input itself and how to converge on it by shrink-wrapping a subdivided convex hull mesh around the given point cloud. Our mesh assimilation algorithm in Chapter 7 operates similarly but solely relies on geometric information acquired from the input point cloud.

In this thesis, we present two surface reconstruction techniques: The method described in Chapter 4 introduces a novel approach to produce pure watertight quad-meshes from unoriented point clouds with control over feature alignment and adaptive resolution. It is based on a divide-and-conquer technique, first spatially clustering the point cloud in a hierarchical *k*d-tree structure which is then reconstructed bottom-up by interconnecting quadrangular tiles between its nodes. In Chapter 7 we explore a very robust mesh assimilation procedure, for generating high-precision 3D scan reconstructions. Its core projection scheme is very versatile and also capable of generating meshes from volumetric data, signed distance fields (SDF), remesh existing objects, or improve other reconstruction results as demonstrated in Appendix 7.B.

3.3 Volume Meshing

Whereas the meshes discussed so far merely specify a hull, separating *in-* from *outside*, volume meshes actually permeate the space within an object, effectively approaching complicated tasks with a divide-and-conquer strategy: An arbitrarily shaped polyhedron is not trivially suitable for complex applications, e.g., involving deformation, simulation, energy transfer, or forms of integration. However, a volumetric mesh clusters the object into reasonably sized volume chunks, which are then much more convenient to handle.

Analogously to the basic kinds of surface meshes with triangular and quadrangular faces, volume meshes can also be categorized into two main (common) types. They are distinguished by their primary primitive type and contrast each other with their advantages and disadvantages, i.e., ease of construction vs. computational versatility in downstream applications. First, there are the tet-meshes, exclusively containing tetrahedra, and secondly, the hex-meshes, featuring hexahedral primitives. While at least low-quality tet-meshes can be quite simple to construct, hex-meshes pose a more complex challenge. Therefore, the constraints are often softened to also allow tetrahedra or even general polyhedra in, then so-called, hex-dominant meshes with the emphasis on as many hexahedral primitives as possible. The main point for hexahedral primitives in a volume mesh is to be simpler and more robustly integrable in finite element method (FEM) and simulation applications by featuring fewer primitives over the same volume than with tetrahedral cells. However, to be actually advantageous over tet-meshes, the individual hexahedral cells have to meet specific quality standards, usually expressed in terms of skewedness and measured as the MSJ (Section 2.3.2). Analogously to surface meshes, adaptive resolution, feature or shape alignment, uniformity, and regularity are also desirable criteria in volume meshes.



When meshing a volume, one rarely starts with a point cloud but in most cases with an existing hull, meshed or CAD-like, or a different type of volume representation. The most trivial volume clustering technique possible would be a voxelization, subdividing space with a regular axis-aligned grid, or a resolution-adaptive octree. However, this grossly oversimplifies the problem as a discretization of the object's actual hull introduces significant aliasing errors. With tetrahedra being the smallest possible three-dimensional primitive, the triangulated hull of an object can be directly incorporated as a boundary for the volume mesh. Analogous to the 2D Voronoi diagram and its Delaunay triangulation, shown in Figure 2.5, the dual of a 3D Voronoi diagram gives a Delaunay tetrahedralization. This can either incorporate the source mesh itself [Si15] or extract a volume mesh from loose triangle soup [HZG*18]. However, such ad hoc tetrahedralizations are not always trivially suitable for further processing as they usually comprise primitives of relatively poor quality (Section 2.3.2), being squashed, stretched, or featuring huge volume discrepancies. Nevertheless, for our proposed applications in Chapter 6, using tet meshes as the basis for mass property computations, we can prove (Appendix 6.D) the calculations to be invariant of the actual volume partitioning, at least in a linearly varying density space. If available, a more uniform tet-mesh can even be used to extract a hexahedral mesh. These strategies [MT00] are based on the fact that a hexahedron can be assembled using a set of tetrahedra. Up to symmetry or rotation there exists 1 solution using 5 tetrahedra, 5 solutions using 6 tetrahedra and 4 solutions using 7 tetrahedra; the latter ones include 1 sliver element (Figure 2.3: a squashed, almost planar tet) each. This extraction may be formulated as an elegant graph matching algorithm [SRUL16]. In this case, the quality of the resulting hex-dominant mesh obviously depends on the given tet-mesh input (Figure 5.21, right). However, as proposed by Lévy and Liu [LL10] tet-meshes can be specifically optimized for this scope of application, using L_p Lloyd relaxations, which significantly improve the hex-dominant results. A related concept, by Pellerin et al. [PJVR18], first identifies all possible and hex-suitable tetrahedralizations, based on the vertices alone.

Besides a *simple* hull, most established techniques usually also require more advanced auxiliary input data, like a quad-meshed boundary surface [CI00, KBLK14]. Moreover, as shown by Lyon et al. [LBK16], the quad-mesh extraction concept from triangulations [EBCK13] can be transferred to tet- and hex-mesh extraction, using parameterizations [NRP11]. Similar to the edge-flow optimization for quad-meshes, frame-fields can provide orientational guidance for aligning hexahedral structures to the shape of the object [SVB17, GJTP17, RSR*18, CAS*19]. Discretizable param-

eterizations or mappings to a regular *PolyCube* [THCM04] representation may push the hex/non-hex ratio, or even allow for all-hex meshes [GSZ11], but often on the expense of low quality primitives. Other methods can utilize handcrafted input like dual-sheets [Tak19], basically a set of curved surfaces bisecting the object in a dual space to specify a basic segmentation, suitable for further hexahedralization. Octahedral frame fields can form constellations, not resolvable with hexahedral topology [SJ08]. Consequently, singularity-restricted fields [LLX*12], or user-specified singularity graphs are often used to improve the mesh quality [LZC*18, CC19] as this takes one of the most challenging tasks out of the algorithm. In an extension of the dual-sheet concept, manually annotated feature edges [LPP*20] may also further ensure proper alignment of hexahedral primitives. However, too demanding constraints or forced alignments may provoke other mesh defects to occur, like heavily deformed or inverted primitives.

With our hex-meshing procedure in Chapter 5 we introduce a novel mesh class, restraining the size of general polyhedra in hex-dominant meshes to a hexahedron, thus being called at-most-hexa meshes. It does not require complex auxiliary input for its relatively easy construction while at the same time inheriting many benefits from pure hex-meshes. The mesh generation is based on a relaxation, followed by a geometry extraction step. Cells align during the relaxation to the input shape itself; guidance from superimposed orientation fields or additional geometry can be utilized but is not an input requirement. Chapter 6 exemplifies various applications of tetrahedral meshes with spatially varying density, for example optimizing mass properties for 3D printing. Moreover, one application specifically targets the analytic formulation of an L_∞ energy term for tetrahedralized cells of a Voronoi diagram, as the one used in the at-most-hexa relaxation.

Chapter 4

Hierarchical Quad Meshing of 3D Scanned Surfaces

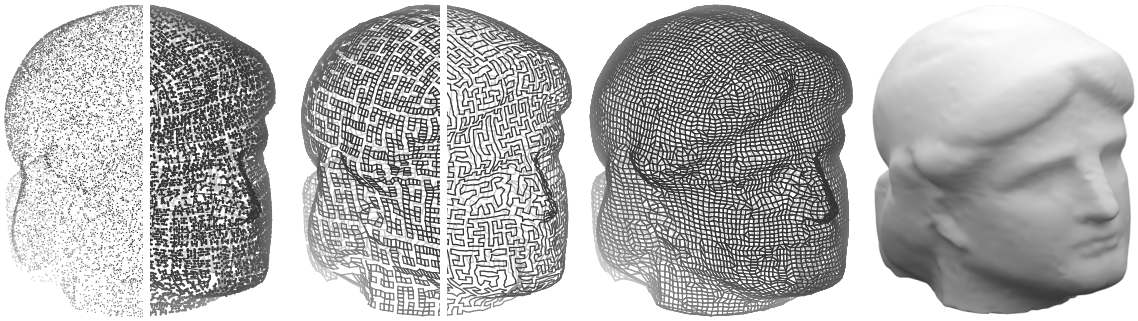


Figure 4.1: Snapshots of our quad meshing process - from left to right: *Minerva* 3D Scan Point Cloud [Mor09] / *k*d-tree with subsampled vertices, interconnected tile maze / open edge path around the maze, the final quad mesh and a rendering of the reconstructed surface. The initial point cloud may be sampled from an existing surface for remeshing or originate from a real 3D scan as demonstrated here.

4.1 Abstract

This chapter presents a novel method to reconstruct watertight quad meshes on scanned 3D geometry. There exist many different approaches to acquire 3D information from real-world objects and sceneries. The resulting point clouds depict scanned surfaces as sparse sets of positional information. A typical downside is the lack of normals, connectivity, or topological adjacency data, making it difficult to actually recover a meaningful surface. The concept described in this chapter is designed to reconstruct a surface mesh despite all this missing information. Even when facing varying sample densities, our algorithm is still guaranteed to produce watertight manifold meshes featuring quad faces only. The topology can be set up to follow superimposed regular structures or align naturally to the point cloud's shape. Our proposed approach is based on an initial divide-and-conquer subsampling procedure: Surface samples are clustered in meaningful neighborhoods as leaves of a *k*d-tree. A representative sample of the surface neighborhood is

determined for each leaf using a spherical surface approximation. The hierarchical structure of the binary tree is utilized to construct a basic set of loose tiles and to interconnect them. As a final step, missing parts of the now coherent tile structure are filled up with an incremental algorithm for locally optimal gap closure. Disfigured or concave faces in the resulting mesh can be removed with a constrained smoothing operator.

4.2 Introduction

As previously introduced, 3D reconstruction is a well-studied subject in computer graphics that also gained interest in other fields like manufacturing, medicine, or robotics. Whereas common acquisition techniques reproduce the scanned scenery as loose sets of 3D points, downstream applications usually require surfaces or meshes to make further use of an object. Many popular meshing algorithms are based on parameterized surfaces or oriented point cloud samples. Our proposed approach can utilize the unoriented point cloud of an object's surface to reconstruct a watertight mesh directly. By construction, the output of our algorithm is a pure quad mesh. Thus, it does not require any additional remeshing for producing quads, while it is trivial to generate triangular meshes from our output by inserting a diagonal to each face. Our algorithm relies on a binary *k*d-tree for spatial 3D clustering: This hierarchical data structure is utilized to generate 2^n meaningful vertices on the point cloud. Furthermore, the *k*d-tree captures valuable neighborhood relations during its construction, which are then used in the meshing step.

4.2.1 Motivation

To briefly summarize Section 3.1 on 3D scanning, resulting surface points are either sampled pointwise, i.e., with a laser, or result as triangulated correlations of multiple input images. Either way, scanned surfaces will be represented with sparsely distributed singularities of three-dimensional position data. Some techniques are able to recover color information or estimate normal directions per sample. Rendering point clouds is challenging and not trivially possible with common ray-tracing or rasterization approaches, detecting collisions or intersections is not directly possible, and the object‘s actual shape is hard to grasp for humans and machines without explicit neighborhood information.

There exist many algorithms and mathematical approaches to recover surfaces from point clouds by fitting primitives or solving Poisson equations. However, often additional work is required to clean up and prepare meshes for downstream applications. The goal of this approach is to produce clean and watertight quad-meshes. We achieve that with a mesh structure that is adaptive to the input as the result of *k*d-tree-based subsampling, exploiting the trade-off between resolution and regularity. While common meshing techniques produce triangulations or quad-dominant meshes [BLP*13], our algorithm creates quad-only meshes directly from point clouds without further remeshing or subdivision steps.

4.2.2 Overview

The main strengths of our proposed meshing algorithm can be summarized as: **Points only:** In contrast to other meshing algorithms, ours does not require any further information as normals, faces, color, or motion. **Quad mesh:** Whereas other approaches may only promise quad-dominant meshes, ours is guaranteed to feature only quads without remeshing. **Arbitrary input:** Point clouds can be of any given size, the resulting mesh resolution can be chosen independently. **Watertight:** Results produced by our algorithm are closed continuous manifold surfaces and do not need additional care to seal holes, e.g., as for 3D printing or volume rendering. **Adaptability:** Tiles vary in size, adaptive to the point cloud resolution instead of leaving holes as other procedures do. **Alignment:** The mesh topology may follow superimposed axis-aligned regularity for organic shapes or align adaptively to point cloud features.

Figure 4.2: Schematic flowchart of the proposed procedure. The stages can be categorized as geometry acquisition and meshing, both further described in the following.

▶ ***k*d-Tree & Subsampling:** A balanced binary *k*d-tree is constructed by recursively subdividing the input point cloud with axis-aligned splits. A sphere is fitted to all samples in a leaf, and a suitable proxy sample on the sphere's surface is selected as a representative vertex. Based on the *k*d-tree, four vertices create a basic quad tile. Section 4.3 elaborates on this in more detail.

▶ **Hierarchical Interconnection:** Dictated by the tree-node's relationships, special tiles are added to interconnect two related parts of the tree on each level. As the algorithm progresses from the leaf's down to the root, all tiles will be interconnected. The resulting structure is one coherent set of quad tiles resembling a maze. By construction, it is guaranteed that all connected quads are enclosed by one coherent path of open edges after this step. This algorithm is discussed in Section 4.4.1.

▶ **Incremental Maze Fill-Up:** All dead ends of the coherent maze-like structure are closed in this last step. A priority queue is processed incrementally until all open edges, and therefore, the mesh is closed. The resulting surface is one watertight manifold featuring quad faces only. This algorithm is introduced in Section 4.4.2 and by default converges to objects with genus 0. Higher genera are possible but require manual intervention as shown in Section 4.6.4.

4.2.3 Related Work

Scans, Point Cloud Generation and Meshing: While Section 3.2 already provides a broad overview on common meshing and reconstruction techniques, this section is more about the details that separate our proposed method from the established state-of-the-art. Many standard approaches rely on the assumption of rather regularly sampled point clouds, and irregular input often requires specialized techniques for surface reconstruction [HDD*92]. For moving-least-squares methods [FCOS05, LCOL07, SSW09] the sample density has great impact on reconstruction accuracy. Ohtake et al. [OBS03] proposed a technique to approach the problem as a hierarchical fitting problem and is, therefore, more robust to non-uniform sampling. Methods that are specialized in the reconstruction from non-uniform point clouds are so far able to generate resampled point sets [LCOLTE07], or triangulated surfaces [HK06]. Our procedure is adaptive to sample density and always results in a watertight quad-only mesh. Field-aligned and quad-dominant results tend to be quite regular but may also feature holes or fail completely if the sample density is varying too much (Figures 4.13, 4.20, 4.22).

Quad Remeshing: Once a surface is recovered from a 3D scan, it may not yet fulfill the requirements for further processing: Many algorithms only produce triangulated meshes, and further processing steps are required to produce quads. Subdivision surfaces [CC78] result in quad meshes but also artificially increase the mesh resolution, which is not always tolerable. A popular approach to produce quad meshes from given surfaces uses a vector-field based parameterization [Knu95] and spawned many successful extensions [KNP07, RLL*06, BZK09]. The approach for remeshing by Daniels et al. [DSC09] introduced a novel hierarchical mapping method to reconstruct arbitrary polygonal meshes with quads from simplified base domains. Mapping a grid of integer iso-lines from $\mathbb{R}^2$ to surfaces creating quad meshes was later introduced as *integer grid map* by Bommes et al. [BCE*13]. Techniques for remeshing can be based on user interaction [ESCK16] or rely on autonomously detected feature lines of the input geometry [PNA*21]. However, all quad remeshing approaches require a surface (parameterized or meshed) which is not given when starting from a point cloud. General qualitative and quantitative characteristics and desirable criteria for quad meshes are introduced in Section 2.3.

Relevant work on geometry: Mousa et al. [MCAG07] employ a *k*d-tree in a similar way as we do but aimed at efficient spherical harmonics representations of 3D objects rather than meshes. The process of *deep points consolidation* [WHG*15] associates each surface point with a deep (interior) point, forming a meso-skeleton. The reconstructed surface is an improvement to naive solutions but involves a Poisson solver, nevertheless. The resulting meshes of our procedure sometimes appear faceted. Cosmetic improvements, e.g., for rendering purposes, can be achieved with *bilateral mesh denoising* [FDCO03]. Needless to say, our results are also suitable input for smoothing algorithms (Section 2.3.1), subdivision surfaces [CC78] or quad mesh simplification [TPC*10].

4.3 *k*d-Tree: Divide and Conquer on 3D Point Clouds

Point clouds of different sources bear challenges as varying sample-count, -density, or -distribution. We propose an algorithm to face these challenges using abstraction: A specialized divide and conquer scheme allows us to subsample given information and recursively construct a hierarchical data structure modeling spatial coherence. The only assumption in our procedure is that input samples originate from surfaces and not volumes. Nevertheless, the algorithm is tolerant to noise and can handle (single) outliers.

Our reconstruction is based on a balanced *k*d-tree. The benefits of this tree are that it is easy to construct and traverse and has low computation and memory overhead. Furthermore, an essential feature for our follow-up meshing strategy is that the *k*d-tree can be constructed to be balanced, which is not very suitable for more general binary-space-partitioning (BSP) approaches or non-binary octrees. The grouped samples contained in our tree's leaf nodes are used to construct a new representative vertex. Four of these new samples will later be connected to one quadrilateral face, in the following also called a tile. New tiles are added on each tree level to interconnect two sets of tiles each, obeying the tree hierarchy. This is why the tree's internal structure is essential for the upcoming interconnection and meshing operations in Section 4.4.

As listed in the introduction, the first step in our procedure is to establish a *k*d-tree. This section is dedicated to its construction. Section 4.5.1 introduces a *k*d-tree extension for feature alignment where splits are no longer axis-aligned but derived from input features.

4.3.1 Split Characteristics

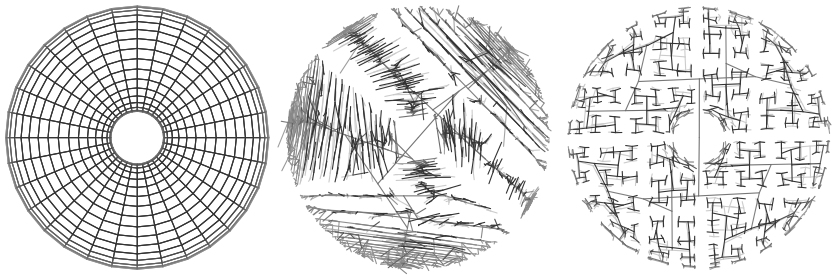


Figure 4.3: Seen from above from left to right: Source mesh of a torus; BSP-tree using mean splits in eigenvector direction; our balanced *k*d-tree with axis aligned median splits.

The tree construction has an impact on the distribution of our proxy vertices and the resulting mesh. Figure 4.3 compares results of two different split approaches: Recursively splitting the point sets along a plane, orthogonal to the largest eigenvector, creates a very far-flung and dispersed tree. We achieved a more regular, thus more suitable structure

with splits along the three world-space axes x, y, z. The largest variance among these axes determines which dimension is used for the split. If not stated otherwise, we used $k = 100$ as criteria to split a node with more than k samples, which yields at least 50 samples per leaf node. Figure 4.16 in the results Section 4.6.2 compares outcomes using four different ks on the same point cloud. The subsampling technique in Section 4.3.2 approximates all leaf samples with a least-squares-fitted sphere. Consequently, at least 4 samples per leaf are advisable, and $8 \leq k$ should be satisfied.

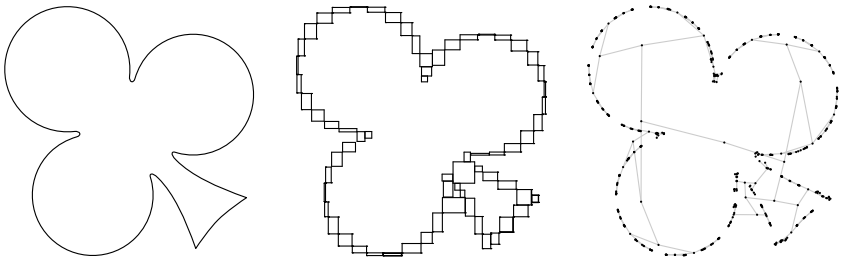


Figure 4.4: 2D example: An asymmetric, not axis aligned shape with rounded and sharp features. The spread of the kd-tree already approximates the hull after only a few levels.

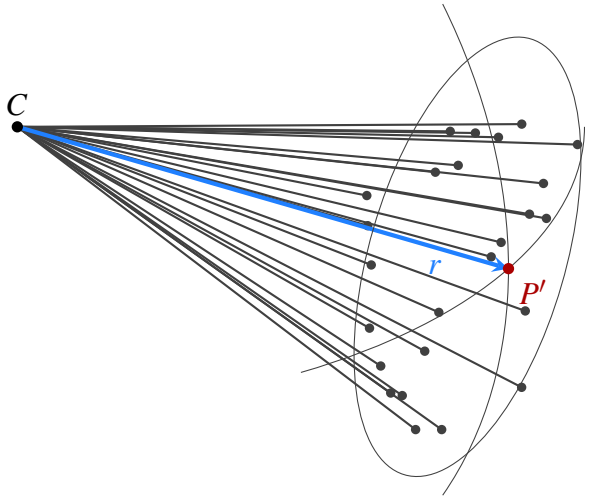
As illustrated in Figure 4.4, the impact of axis-aligned splits is only minor, and the recursive decomposition is adaptive enough to approximate organic shapes and structures without axis-aligned symmetry. Nevertheless, in Section 4.5.1 we demonstrate the possibility to use split planes adaptively aligned to the point cloud. Furthermore, the example on the right in Figure 4.4 visualizes the importance of the assumption that the point cloud samples model surfaces and not volumes. Only the first few nodes close to the root actually populate the interior of the sampled space. Nodes on or close to the leaf level are already distributed over the sampled surface itself.

Balance: To ensure the binary tree is balanced and all leaves end up on the same level, we use the median as split position. Furthermore, a leaf count of 2^i with $2 \leq i$ is always divisible by 4, which is crucial to construct quadrilateral faces. For the following, let the root node be level 1 and leaf nodes on level n. Four leaf nodes are required to construct at least one quadrilateral face. As a result, all leaf nodes must be on a level $3 \leq n$.

4.3.2 Subsampling: Spherical Surface Fits

One representative vertex for all samples in a leaf is created by subsampling the points inside a node. The synthetic vertices created in this step are the sole geometric base for the generated surface mesh; no additional vertices are added.

A leaf node may contain up to k samples of the point cloud. Due to the spatial decomposition in the kd-tree, the samples $s \in \mathbf{S}$ in a leaf are, in some sense, a nearest neighborhood. A single point P' is determined to represent all samples in a leaf node. The geometric mean P (Equation (4.3)) of all samples' 3D positions is not suitable to model bent or curved surfaces properly and is subject to noise. Alternatively, a sphere is fitted to the set of samples: The over-determined system of Equations (4.1) is solved in a least-squares sense and yields the sphere's center C and radius r (Equation (4.2)). The mean of all samples P is then projected onto the sphere's surface as point P' to better represent the properties of the actually sampled surface. This is illustrated in Figure 4.5 and formulated in Equation (4.4).



$$\begin{bmatrix} |s_0|^2 \\ \vdots \\ |s_m|^2 \end{bmatrix} = \begin{bmatrix} 2s_0^T & 1 \\ \vdots & \vdots \\ 2s_m^T & 1 \end{bmatrix} \begin{bmatrix} C \\ t \end{bmatrix} \tag{4.1}$$

$$r = \sqrt{|C|^2 + t} \tag{4.2}$$

$$P = \frac{1}{|\mathbf{S}|} \sum_{s_i \in \mathbf{S}} s_i \tag{4.3}$$

$$P' = C + r\frac{P - C}{|P - C|} \tag{4.4}$$

Figure 4.5: Samples $s \in \mathbf{S}$ (with $|\mathbf{S}| = m + 1$) of a single leaf node. The projected vertex P' is found on the surface of a least-squares fitted sphere with center C and radius r.

4.3.3 From Leafs to Quad Tiles

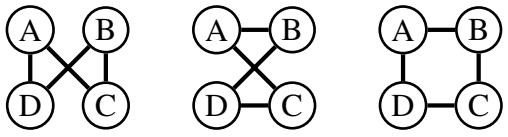


Figure 4.6: Three possible constellations to form a quad with four vertices A, B, C, D. Our algorithm will select the rightmost solution, which has the smallest perimeter.

All the final quad mesh vertices are essentially the kd-tree's leaf node's P' points. To join vertices to become faces, one has to determine how to connect them: Our algorithm starts with nodes on level $n-2$. Each of those nodes has two children on level $n-1$, which again have two leaf nodes each. This makes four leaf nodes (each has a vertex P') for a node on level $n-2$ to form a tile. Besides their affiliation, there is no information on how these vertices should be arranged as a quad face. In a triangular case, there is only

one option, but in a quad face, there are three possibilities, as illustrated in Figure 4.6. For our goal of a regular mesh, faces are favorably convex and without twists. The best solution out of the three possibilities is found as the one with the smallest perimeter. P' points are usually inside their leaf's bounding box, otherwise, they may be clipped. Since bounding boxes do not intersect each other and a tile is constructed from four nearest leaf nodes, we can guarantee that created tiles will also not intersect each other.

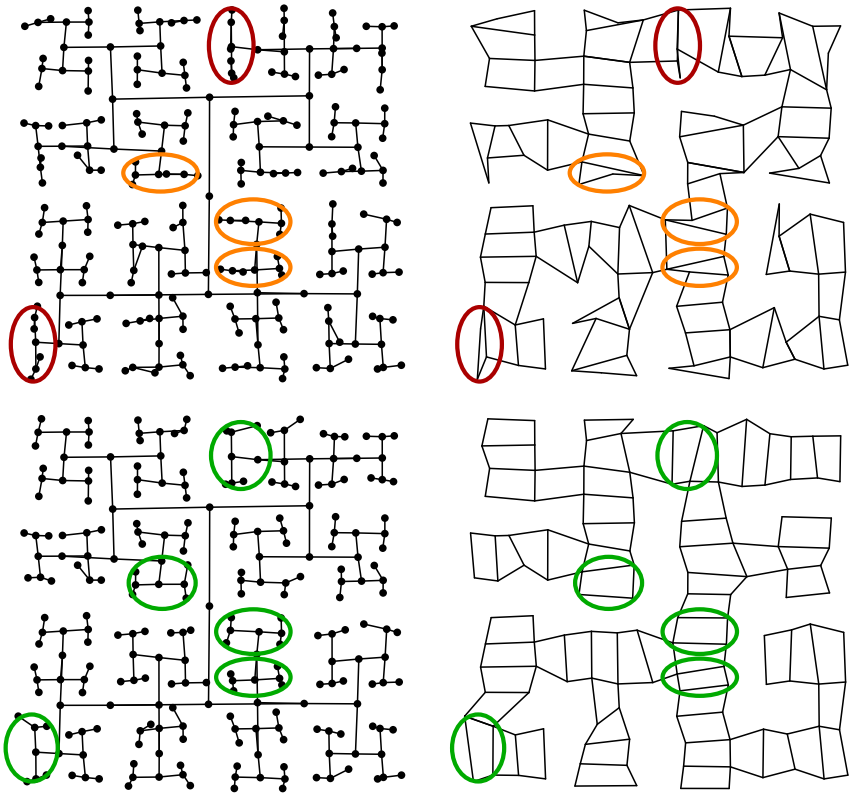


Figure 4.7: Examples of a *k*d-Tree (left) and tiles (right) from samples of a simple 2D plane. Top: When nodes on level $n-2$ and $n-1$ split in the same dimension. Bottom: Results with our split alignment correction.

4.3.4 Subsequent Split Alignment

It is possible that nodes on level $n-2$ and $n-1$ split along the same dimension. As highlighted in the top row of Figure 4.7, this causes tiles to become very narrow. If only one level $n-1$ node has the same split dimension as its parent, tiles become disfigured. Either of these cases might be the best local split according to the applied condition, however, there is no point in keeping those tiles if their shape is counterproductive for further use. Therefore, we extended our tree construction algorithm to detect such constellations and make use of the second-best option for the split dimension. Fixed results of realigned level $n-1$ nodes are shown in the bottom row of Figure 4.7.

4.3.5 Triangulated Patches instead of Quad Tiles

Apart from quad faces, there is also the option for a triangulation based on the Delaunay criterion. Under the assumption of local spherical curvature, all sample points per leaf node cluster are projected onto the fitted sphere. One can determine a perfect Delaunay triangulation of these projections on the sphere's surface. This solution can be used as a reasonably good triangulation of the original samples. Where our default quad mesh relies on one subsample per leaf node, this solutions creates a triangulated patch containing all samples of a node. While this allows for a higher resolution of the final mesh, the downsides predominate: The mesh again only features triangles, the original sample points may still feature noise and the procedure to sew individual patches together is computationally very exhaustive. Furthermore, the seams between triangulated patches will not necessarily obey the Delaunay criterion and require additional processing.

4.4 Meshing

This section elaborates further details on our primary meshing method, which proceeds in two steps: First, the basic set of loose quad tiles from the *k*d-tree is interconnected to form one coherent set of tiles. Second, dead ends of the resulting maze-like structure are filled up incrementally to finalize the meshing.

4.4.1 Hierarchical Interconnection

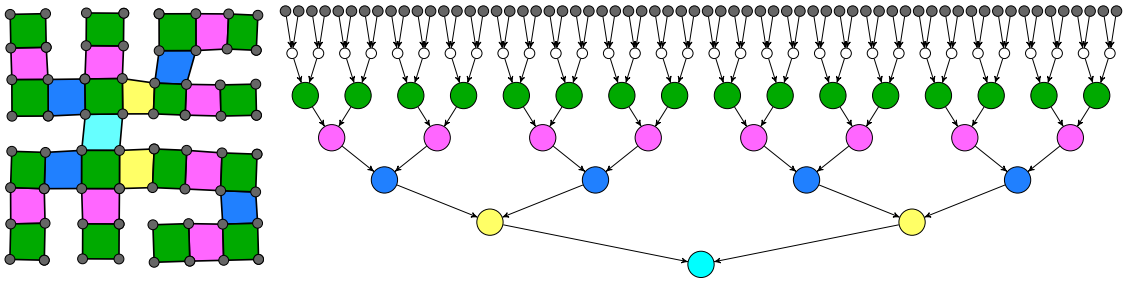


Figure 4.8: Left: Color coded quad-tiles. Right: The corresponding *k*d-tree. Levels n and $n-1$ are illustrated in gray and white, respectively; green nodes are level $n-2$.

The following algorithm is illustrated in Figure 4.8. As described in Section 4.3.3, the initial set of tiles (■) is created from descendants of level $n-2$ nodes (⁘). By progressing through the tree from leaf to root level, the algorithm determines which sets of tiles are about to be connected next: A level $n-3$ node (●) specifies which two tiles from level $n-2$ (■) should be connected by inserting a new one (■). A level $n-4$ node (●) specifies which two sets of three tiles (■■■) should be connected with a new one (■), and so on. Although there are usually more possible ways, only one tile is created per node, which yields $2^{i-1}-1$ tiles per level $1 \leq i$.

This constraint is crucial: The created structure will always feature only one closed path of coherent open edges around the geometry. The yet unoccupied tile positions are resolved with the fill-up algorithm detailed in Section 4.4.2.

Finding a Connection: Each node has two children, and each child comes with a coherent set of tiles, starting on level $n-2$ with only one tile per node. Each node has to determine how the tile-sets of its children can be connected by inserting one new tile. Therefore, the algorithm has to select one edge of each child's tile-set, create two new edges and add them as the connecting tile.

Similar to the situation illustrated in Figure 4.6, the algorithm has to determine an order for four vertices in the new tile. Since two edges are already given, say AD and BC, the center case in Figure 4.6 is not possible, and there are only two options left. As one can easily comprehend, one constellation of edge pairs is usually more suitable to be connected than the other. It has to be determined where the connection can be established: We chose the combined length of the two new edges as the objective. These are illustrated in red in Figure 4.9. The tile that requires the shortest two new edges will be added. However, the number of possible edge pairs grows very quickly: For a level $n-2-i$ with $1 \leq i \leq n-3$ the number of possible edge pairs to chose from is 2^{2+2i}.

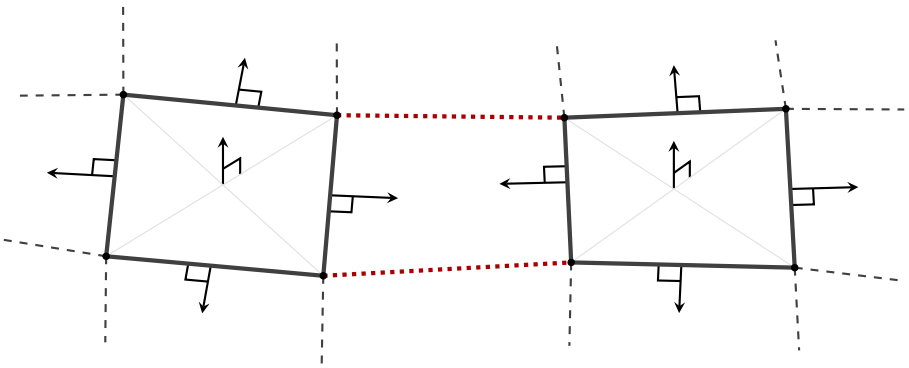


Figure 4.9: Only edges with normals facing other geometry can create new faces. The combined length of potential new edges (dotted red) indicates where to create a new face.

To keep this computation feasible, we filter out most edges beforehand, based on the criteria that their edge normals have to point into the direction of connectable geometry. The four vertices of a tile are not necessarily coplanar, so the tile normal is computed as the cross product of the tile's diagonals. An edge normal is perpendicular to the edge and computed as the cross product of edge direction and tile normal. As illustrated in Figure 4.9, from 16 possible edge pairs, there is now only one pair of edges left where both edge normals point towards the other set of tiles. However, if the number of edge

pairs is still too large, we can more rigorously limit the set of pairs to a fixed number k. Then only the first k edge pairs with the smallest distance between their midpoints will be considered for the creation of a new tile. We also achieved great filter results with a maximum threshold λm on the midpoint to midpoint distance of an edge pair. We chose m as the mean length of all existing edges from the basic tile-set and a scaling factor $1 \leq \lambda \leq 10$ dependent on the desired mesh regularity.

4.4.2 Fill-Up: A-Maze-ing Surface Reconstruction

The interconnected tiles structure from the tree traversal resembles a maze, spanning through the whole point cloud. As commonly known, the distinction between a maze and a labyrinth is that a labyrinth has only one way through and no dead ends, whereas a maze can have multiple paths and also dead ends. There is only one path connecting two tiles in our mesh, but it still features dead ends, thus technically qualifies as a maze.

The maze outline is one coherent path of all open edges, *topologically equivalent to closed loop*. By construction, the number of open edges is dividable by 4 so that it is always possible to close this circle iteratively with quads. The following algorithm only considers the maze's negative so that empty space is actually interpreted as closable. To fill the *loop* hole, the algorithm inserts one quad at a time by closing the surface that is spanned by three connected, open edges. A priority queue ensures to process the most well-formed quad candidate first. In general, this will lead to the maze hole being filled up from the dead ends to the more open space.

(A,B,C)	(I,J,K)
(B,C,D)	(J,K,L)
(C,D,E)	(K,L,M)
(D,E,F)	(L,M,N)
(E,F,G)	(M,N,O)
(F,G,H)	(N,O,P)
(G,H,I)	(O,P,A)
(H,I,J)	(P,A,B)

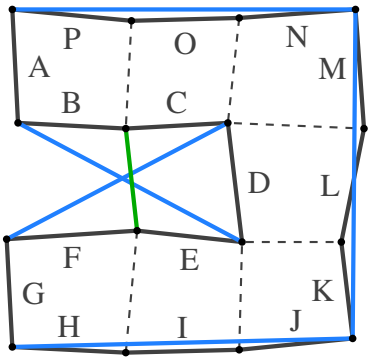


Figure 4.10: Adjacent *open* edges $A,\ldots,P$ ordered in triples, *inner* edges (dashed) are ignored. Only orange triples feature edges of three distinct tiles, possible closing edges are drawn in on the right. The green edge is the best, so (C,D,E) is closed up.

Three edges are considered as a dead-end if they are all open (only one face attached) and form one coherent edge path. Therefore, all open edges are assigned to groups of three adjacent edges each, as illustrated by the example in Figure 4.10. However, not all triples

should be considered: Negative dead-ends like (F,G,H), (P,A,B) and corner-cases like (A,B,C), (E,F,G), (G,H,I), (I,J,K), (J,K,L), (L,M,N), (M,N,O) and (O,P,A) should be discarded. We can filter out these cases with the additional constraint that all three edges in a triple must originate from three distinct tiles. This is also illustrated in Figure 4.10. The remaining six triples are highlighted in orange, and their corresponding close-up edges are drawn in on the right (blue, green).

Our algorithm maintains a priority queue, featuring all triples to be considered, sorted by the objective function $f(\ldots)$ in ascending order. Objective $f(e_L,e_C,e_R,e_N)$ measures the well-formedness of a potential quad with new edge e_N in context of a left-center-right edge triple (e_L,e_C,e_R). It is formulated in Equation (4.5), with $1=\lambda_\varphi+\lambda_\theta$. Components $\varphi(\ldots)$ and $\theta(\ldots)$ are illustrated in Figure 4.11.

$$f(e_L,e_C,e_R,e_N)=\lambda_\varphi\cdot\varphi(e_L,e_C,e_R,e_N)+\lambda_\theta\cdot\theta(e_L,e_R) \tag{4.5}$$

The first term of the objective function is expressed in Equation (4.6) and determines the ratio of the new edge compared to the perimeter of the whole face. In the worst case, the new edge e_N has the same length as e_L,e_C and e_R combined and the quad would have no surface area at all.

$$\varphi(e_L,e_C,e_R,e_N)=4\left|\frac{|e_N|}{\sum_{i\in[L,C,R,N]}|e_i|}-\frac{1}{4}\right| \tag{4.6}$$

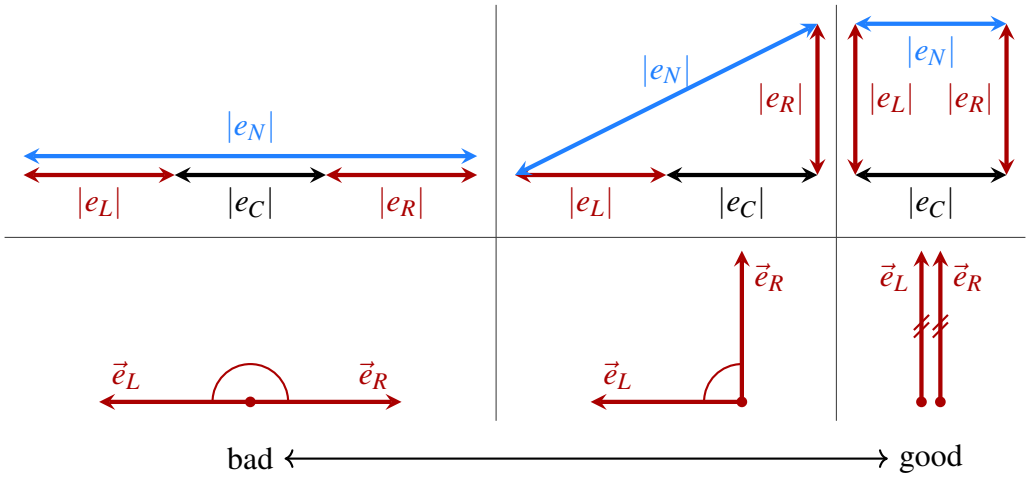


Figure 4.11: Good and bad edge constellations according to Equation (4.5). Edge-length ratios in the upper row are evaluated by the φ-term of Equation (4.6), enclosed angles in the bottom row by the θ-term of Equation (4.7).

Equation (4.7) expresses the second part of the objective function, which considers the angle between opposite edges e_L and e_R. Here the worst case is defined as e_L and e_R pointing away from each other. The most desirable scenario is that both are parallel and point into the same direction.

$$\theta(e_L, e_R) = \frac{\frac{e_L}{|e_L|} \cdot \frac{e_R}{|e_R|} - 1}{-2} \tag{4.7}$$

The triple with the smallest value for $f(\ldots)$ is popped off the queue and instantiated as a new face. In Figure 4.10 the best possible edge corresponds to the triple (C, D, E) and is marked in green. This step is repeated incrementally on all open edges until the queue is empty, and therefore, all dead ends are closed up. Figure 4.12 shows exemplary progress of this algorithm based on the mesh from Figure 4.8.

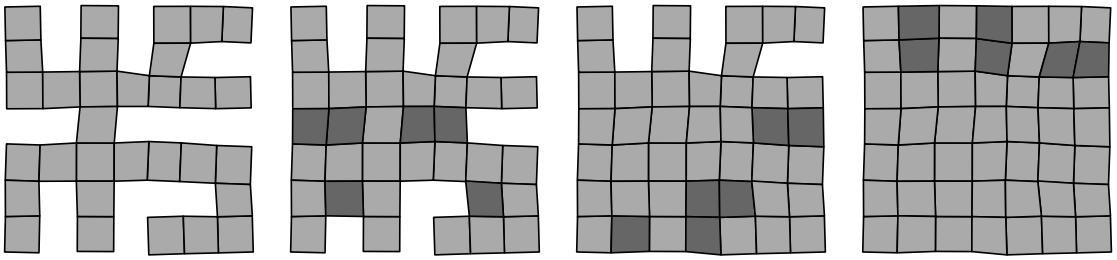


Figure 4.12: Progress steps of filling up the maze structure with new quad faces. From left to right after 0, 6, 12 and 18 iterations.

4.5 Extensions

This section introduces two techniques to advance on the algorithms described so far in Sections 4.3 and 4.4, e.g., with the ability to produce feature-aligned structures (Section 2.2) and more regular mesh topology [Section 2.3.3).

4.5.1 Feature Alignment with Robust Principal Axes

The orientation and layout of our final mesh are founded in the underlying *k*d-tree and its axis-aligned splits. While organic shapes as the busts in Figure 4.17 often benefit from this superimposed orientation, an adaptive alignment is suitable in other scenarios, e.g., as shown for the finger in Figure 4.13.

Mesh flow alignment to the scanned object's shape is an essential feature in many quad (re)meshing applications. Our default axis-aligned setup is easily extendable to incorporate feature alignment as well. Therefore, we introduce alignment layers in our *k*d-tree: Instead of axis-aligned splits, each node determines a unique orientation for its subset of the point cloud. Four nodes on level 3 are illustrated with their orientation axes in Figure 4.13e. We employ the method introduced by Liu et al. [LR09] for determining robust principal axes along which the *k*d-tree splits are performed. Decedents of alignment nodes may inherit the orientation of their parent nodes for subsequent recursive splits along the same axes or determine their own orientations if necessary.

Figure 4.13 compares reconstructions with different algorithms based on the same point cloud. The field-alignment algorithms (a) and (b) natively align mesh topology to principal directions of the oriented point cloud samples. The official implementations of IFAM [JTPSH15] and OSR [STJ*17] crashed if the input point cloud did not feature faces or normals. Consequently, artificial normals were created using Meshlab's [CCC*08] neighborhood-based algorithm with default settings. For comparability, the algorithms were tuned to produce a target resolution of 2k faces. Due to the low and varying point cloud resolution, the algorithms of Jakob et al. (4.13a) *degraded gracefully* with a gap close to the fingertip and Schertler et al. (4.13b) produced okay-ish results only around the knuckle joint. With default settings, the application of Schertler et al. is able to recover the finger but fails if tuned to match this mesh resolution.

Our default solution with axis-aligned splits (4.13c) is a watertight quad mesh, but the mesh flow is not yet aligned to the fingers structure. A feature aligned result of our method using the automatically oriented nodes from (e) is shown in Figure 4.13d where our quad mesh follows the fingers shape.

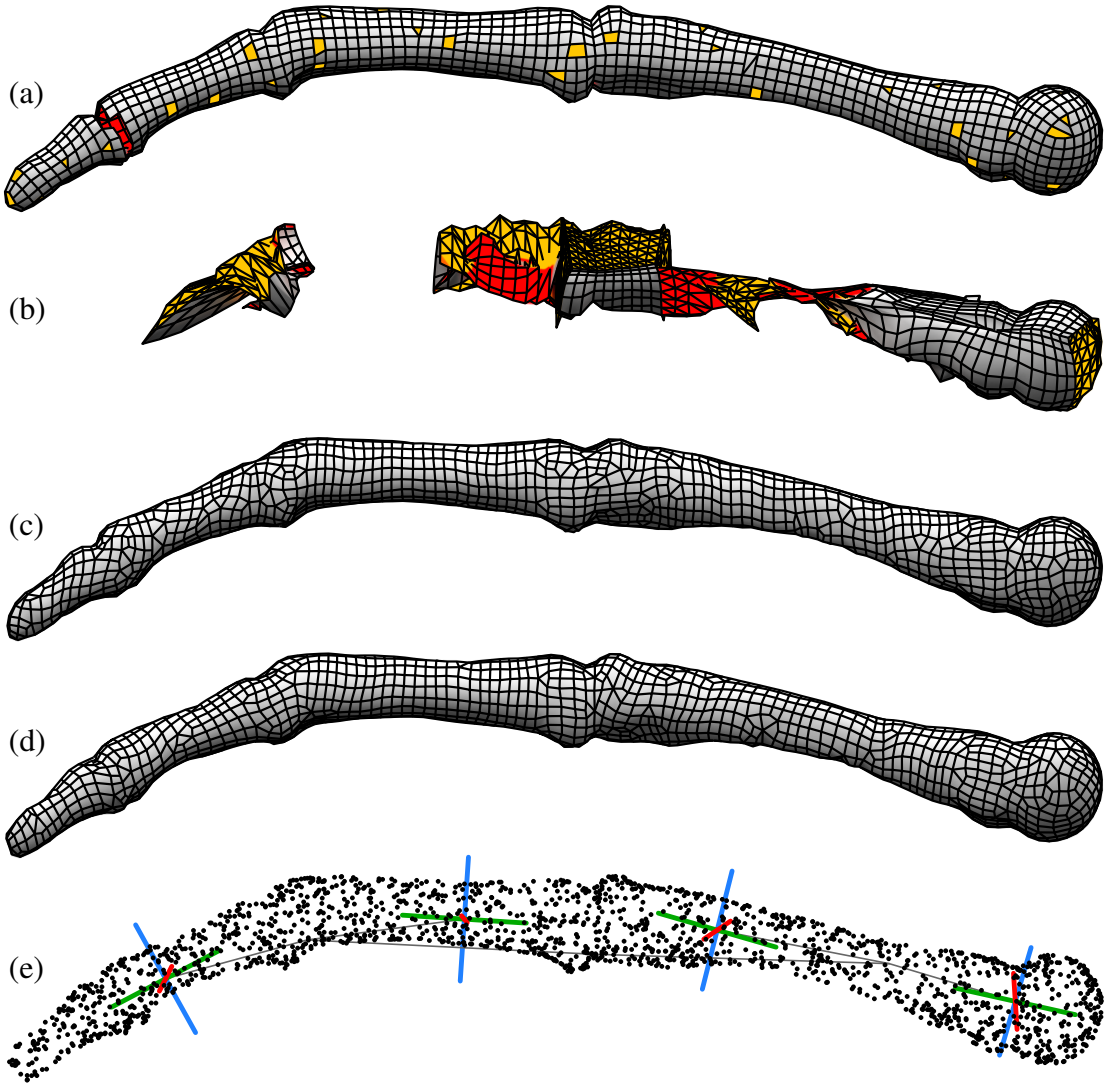


Figure 4.13: (a) Bones of a finger reconstructed with [JTPSH15], (b) [STJ*17], (c) Ours with axis-aligned splits, (d) Ours with adaptive splits, (e) Point cloud and automatically aligned nodes used in (d). Legend: ■ normal quad, ■ backfacing, ■ non-quad.

4.5.2 Additional Mesh Optimization

To improve mesh quality, we apply optimizations in two stages: The first step is a rectification operation, locally applied on each quad tile before the tile interconnection in Section 4.4.1. Therefore, vertices of a tile are repositioned around its initial center of gravity by leveling out the diagonals of the tile. With an increasing number of iterations, tiles more and more approximate a square shape. Due to more regularly shaped tiles, this step has an actual influence on the interconnection result afterwards.

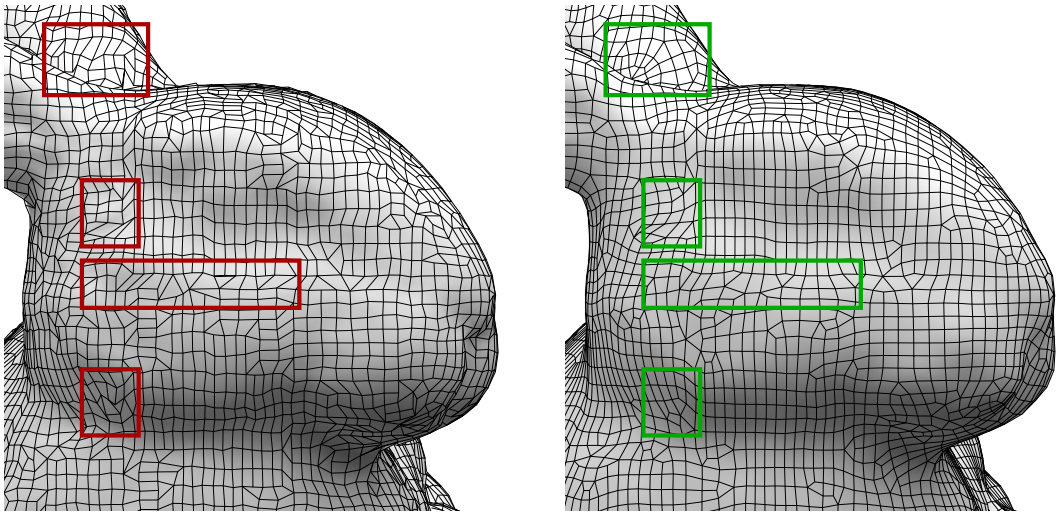


Figure 4.14: Our raw output mesh on the left and after regularization and constrained smoothing on the right. Point cloud source: *Bunny* [TL14].

After the final step, when the mesh is filled up and watertight, we can apply a constrained smoothing operator. Vertices are moved with respect to their mesh neighborhood but constrained by their affiliated point samples. This relaxation has no impact on the mesh topology. A comparison of a raw output mesh and an optimized result is shown in Figure 4.14. Colored boxes point out improvements due to the first regularization step, with improved mesh flow.

Whereas our proposed sphere-projection (Section 4.3.2) is a quite robust method to generate vertices under the presence of noise, it tends to smooth out fine or sharp details, as it is common in many reconstruction methods with a smoothness prior. However, as demonstrated with the experiments shown in Figure 4.21 and Appendix 7.B, our proposed mesh assimilation technique (Chapter 7) can be applied, compensating exactly those flaws. The reconstruction accuracy of our quad-mesh improves as vertices are projected into the tangent space of fine point cloud details and align on sharp feature edges.

4.6 Discussion

In this section, we discuss the performance and capabilities of our proposed concept. Results on real and synthetic data are featured in Section 4.6.2, comparisons to state-of-the-art quad-meshing procedures are documented in Section 4.6.3.

4.6.1 Requirements and Assumptions

A strength of our approach is that it requires not much more than a 3D surface point cloud to generate a meshed surface. There are no assumptions made about a surface's orientation, i.e., requiring normals or similar data. Due to our *k*d-tree basis, the resulting mesh automatically adapts to the density of the provided point cloud. The only assumption is that samples predominantly originate from surfaces and that important surface details are sampled with sufficient density. Otherwise, our *k*d-tree leaf nodes could not populate the outer hull, and the resulting tiles would feature vertices inside the object. However, outliers inside a volume may still be handled by considering them as noise.

4.6.2 Results

Meshes shown in Figure 4.15 demonstrate the remeshing capabilities of our procedure. The source mesh, shown on the left, features only triangular faces and heavily varying mesh resolution. However, our remeshed results feature only quadrilateral faces of similar size and regular distribution. In these three examples, the number of surface samples was increased from left to right. With the same split threshold k for leaf nodes, more samples cause more tree levels. As a result, representative samples are closer to each other, reconstructed tiles become smaller, and the quad mesh resolution increases.

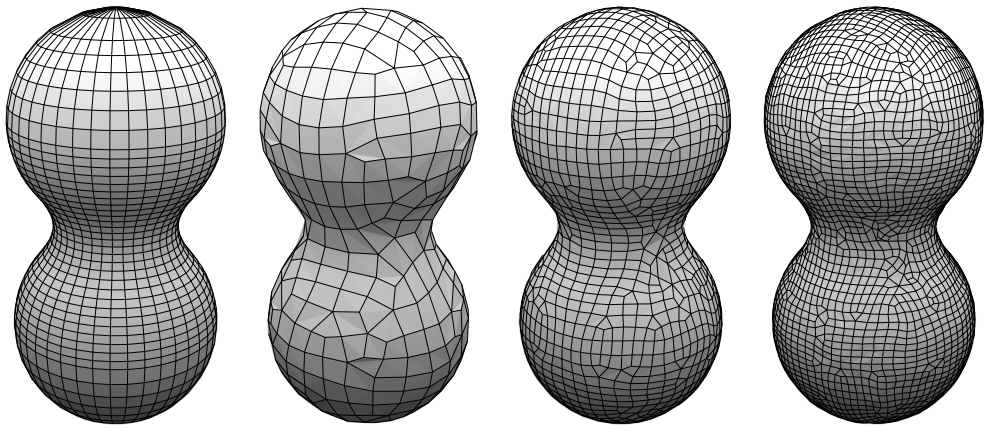
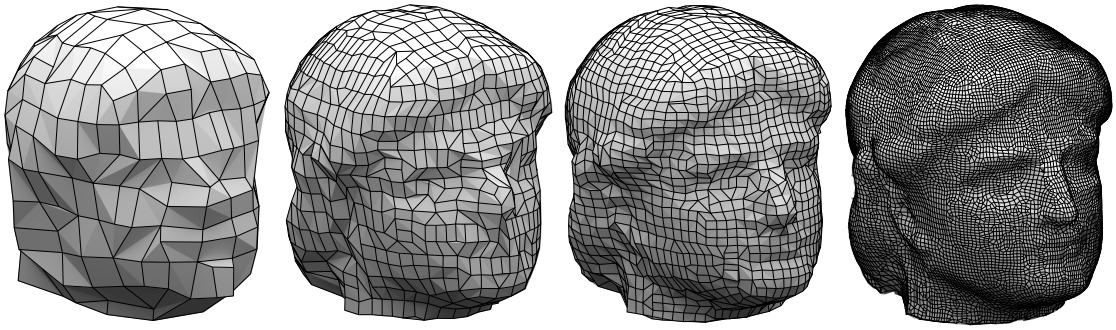


Figure 4.15: Remeshing based on a given input object (left) with varying tile sizes. The number of surface samples was increased from left to right, causing a deeper tree decomposition, thus a higher quad mesh resolution.

Examples for meshing with different resolutions are illustrated in Figure 4.16. With a fixed number of real 3D scan samples, the split condition k was lowered from left to right. This directly influences the number of tree levels and, therefore, the number of vertices and tiles of the mesh. As one can observe, the rough shape and key features are captured in all resolutions.



k=500, n=8 k=100, n=10 k=50, n=11 k=10, n=14

Figure 4.16: Lowered split conditions k increase the tree depths n and therefore the resulting mesh resolution. Source: *Minerva* scan [Mor09], 98.5k samples.

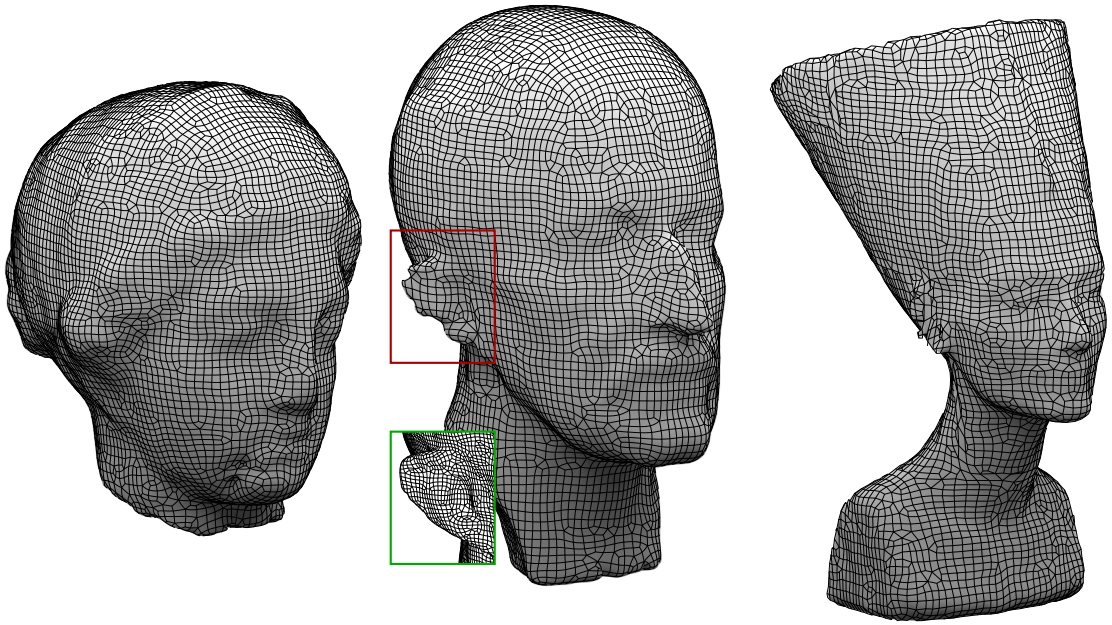


Figure 4.17: Meshing results of *Igea*, *Max Planck* [LGS02] and *Nefertiti* [NAB]. Sampling artifacts (red) on fine structures are resolved with higher sample density (green).

The images shown in Figure 4.17 illustrate results of simulated 3D scan meshing. An arbitrary number of surface samples is distributed randomly over given 3D models. To approximate 3D scan results with noise, all samples were perturbed with random jitter vectors. Our resulting meshes form a complete watertight manifold of genus zero. Although axis-aligned *k*d-tree splits were used, the organic and rounded nature of the surfaces is captured nonetheless. Insufficient sample density on fine details may cause mesh artifacts, as pointed out with the red box. The green box shows a flawless reconstruction of the same region when sampled with a higher density.

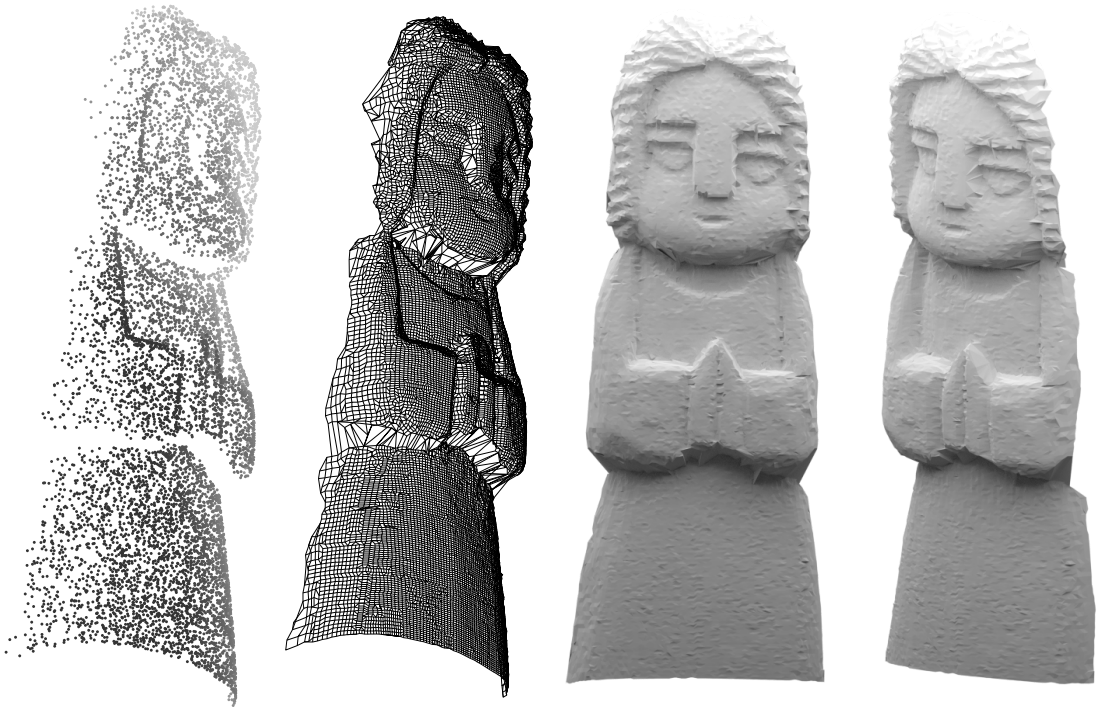


Figure 4.18: Reconstruction of a 3D scan [GRL17] with 1.4M samples (only 10k visualized). Our mesh is adaptive to the scan's resolution and covers even the empty areas under the chin and arms.

Our results on an actual 3D scan are shown in Figure 4.18. Since the angel is captured only from the front, the point cloud's resolution decreases on surface regions that are close to parallel to the view direction as on the side of the model. Our result mesh adapts this varying resolution with quads of different size. The scan features blind spots under the chin and arms without any samples. Despite this depth discontinuity and lack of data, our algorithm recovers a continuous surface. The object's back-plane was removed manually for illustration purposes.

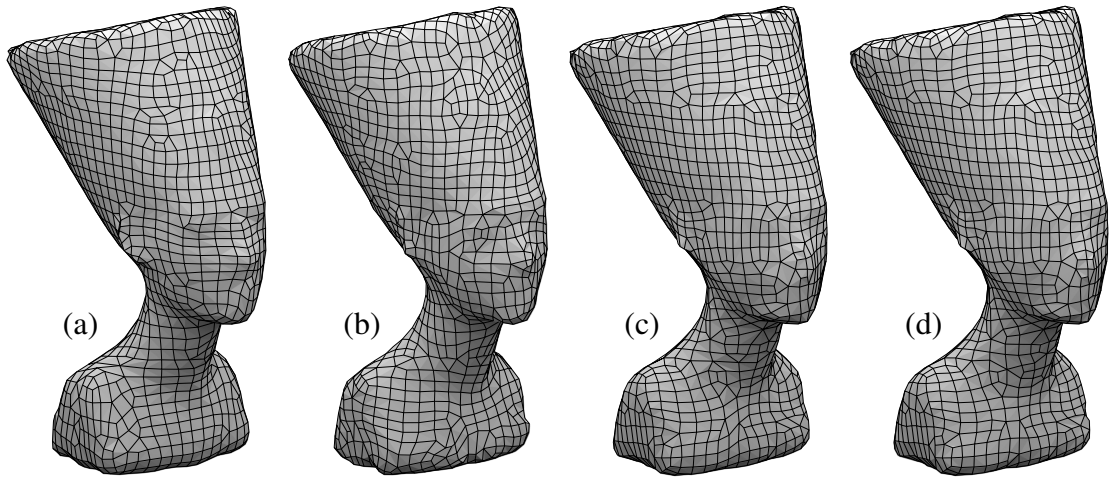


Figure 4.19: Influence of orientation: (a) Object sampled as-is, (b) rotated vertically by 45°, (c) same as (a), (d) same as (b). But (c) and (d) both include an alignment node (Section 4.5.1) on level 0. Thus, results are invariant to input orientation and (c) ≈ (d).

The examples shown in Figure 4.19 demonstrate the impact of the input's orientation on our result meshes. Whereas a) was sampled as-is from a given axis-aligned model, b) was rotated by 45° on the vertical axis prior to the following axis-aligned *k*d-tree subdivision. The resulting meshes differ, but neither one drastically suffered from the assumed axis-alignment. c) and d) show examples derived from the same oriented input as a) and b) respectively but with an alignment-node, as introduced in Section 4.5.1, on tree level 0. This allows the subdivision to start on an approximately identically oriented input and c) ≈ d). Minor differences may still occur due to the RANSAC-based algorithm [LR09] for determining the orientation.

4.6.3 Comparison to state-of-the-art quad-meshing procedures

A comparison of our work against the state-of-the-art for quad-meshing from point clouds is illustrated in Figure 4.20. All algorithms used the same input point cloud and were tuned to produce the same output resolution. As required for the available implementations of Jakob et al. and Schertler et al., normals were again estimated from the basic point cloud in a preceding step. Our procedure is adaptive to the underlying point cloud density, and therefore, the resulting mesh might appear not as homogeneous as the competitors. This can be observed in the neck region, where our result features bigger tiles to compensate for lower sample density and the others just produced holes. The same effect is illustrated in Figure 4.22 which explicitly compares behavior under varying sample density. Furthermore, since both other approaches allow triangles or arbitrary n-gons in their mesh, singularities are easier to avoid than in our pure quad mesh.

Zoom-in boxes on the meshes in Figure 4.20 point out regions where sample density was insufficient for this level of detail or high curvature. This results in holes in the IFAM mesh and an increased number of non-quads in the OSR result, whereas ours adapts to the input and returns a watertight quad-only mesh. As the curvature comparison in the bottom row shows, our result models coarse and fine details with comparable quality.

The *subdiv* columns show results from coarser and subdivided meshes. The IFAM result was subdivided in the official implementation, the OSR result with subdivision surfaces [CC78]. This additional step yields quad-only meshes but also introduces curvature artifacts, which do not occur in results that natively meet the target resolution.

	Vertices Valence			Vertices per Face		
	≤ 3	4	$5 \leq$	≤ 3	4	$5 \leq$
IFAM [JTPSH15]	3.3%	95.1%	1.6%	1.3%	98%	0.7%
OSR [STJ*17]	1.5%	91.5%	7%	7.2%	92.8%	
Ours	8.2%	84.3%	7.5%		**100%**	

Table 4.1: Statistical evaluation of result meshes shown in Figure 4.20, accounting only native target-resolution meshes.

A quantitative comparison of the results from Figure 4.20 is listed in Table 4.1. On the one hand, the IFAM result counts the least amount of singularities but, on the other hand, left some open regions in the mesh. The OSR result gracefully covers such regions but features an increased amount of non-quads in these areas. Our mesh natively features quad tiles only and is also the only closed manifold, therefore, straightforwardly suitable for 3D printing or volume rendering without further steps required.

The visualization and stats shown in Figure 4.21 further exemplify a qualitative comparison against IFAM and OSR on the *Minerva* 3D scan. Plots in the upper row show surface deviations from the given point cloud samples. That our result is more speckled with local inaccuracies than the competing ones may be attributed to the fact that IFAM and OSR are again based on normal information, which is not required for our technique. This higher density of small errors is also reflected in the RMSE measure. However, with the HD accounting for the absolute maximum error, our result is on par with OSR in terms of deviation magnitude, and IFAM falls behind. While our result is the only closed mesh, IFAM and OSR both feature holes, especially on sparsely sampled sections of the point cloud, e.g., under the nose, at the ear or between neck and chignon. The comparison also features results of our method after a finalizing pass with the technique introduced in Chapter 7. As this step also incorporates normal information, the error levels are significantly lowered, thus the reconstruction becomes more accurate. Details on this finalizing pass and more dedicated experiments are discussed in Appendix 7.B.

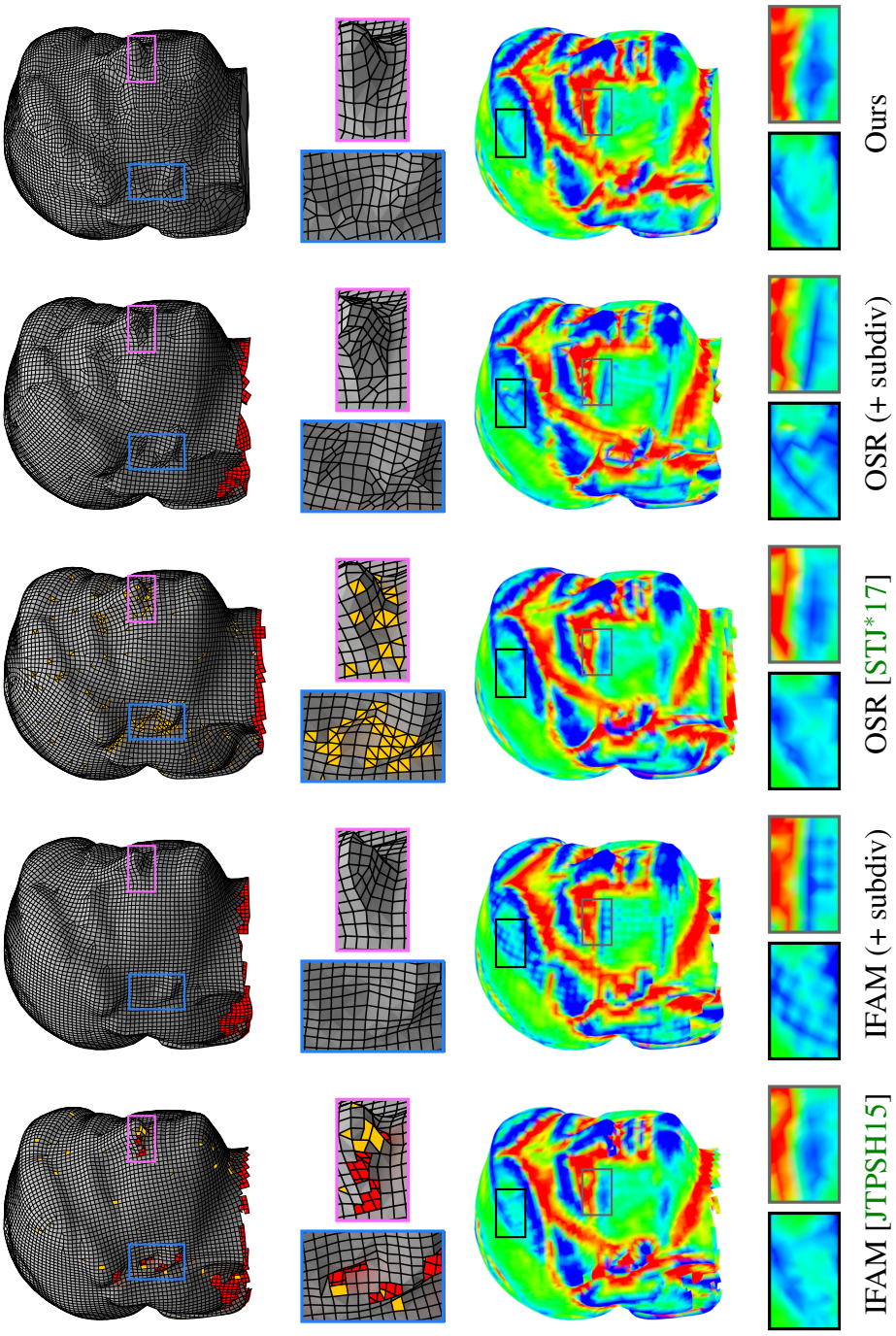


Figure 4.20: Reconstructions on the same point cloud (with artificial normals for IFAM and OSR) for a target resolution of ∼8k faces. Colors on the left encode interior and non-quad faces. Curvature is color-coded on the right. *subdiv* columns show results from coarser and subdivided resutls to also feature quad faces only and to match the target resolution. Subdivision introduces unwanted curvature artifacts as pointed out with the gray boxes. Table 4.1 lists a statistical comparison of these meshes.

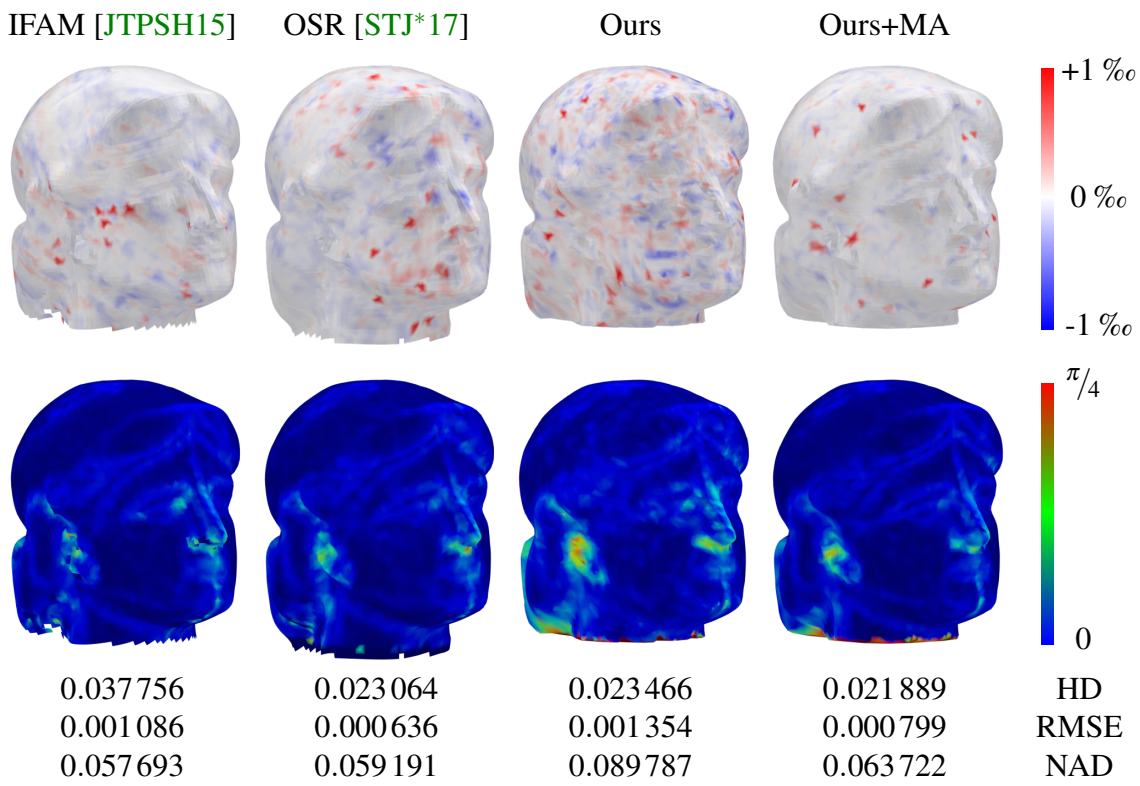
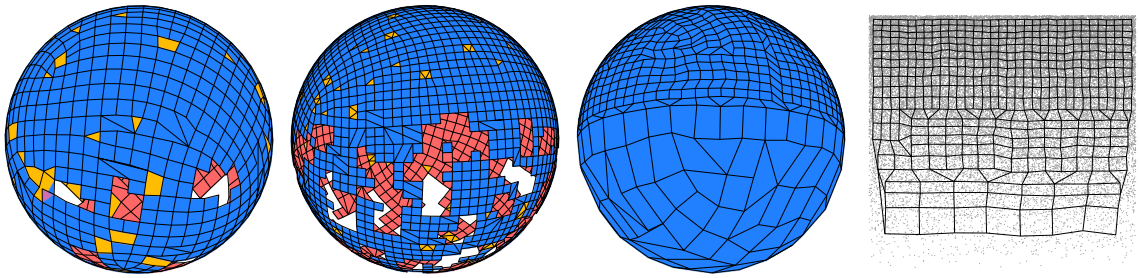


Figure 4.21: Approximation error: Objects are scaled to fit in a unit-cube. Errors list the HD and RMSE as deviations from the original point cloud, visualized in permill and the NAD in radians. Our error measures are in the same order of magnitude as the stat-of-the-art, while being derived from a point cloud without normal information. IFAM and OSR again feature holes, whereas our result is closed. The rightmost column shows our results after a separate pass with the mesh assimilation (MA), introduced in Chapter 7.

Adaptivity Two extreme examples with varying sample density are illustrated in Figure 4.22: The input point cloud sphere for the examples on the left features 90% of all samples on the upper and 10% on the lower hemisphere. Whereas field-aligned methods for quad-dominant meshes [JTPSH15, STJ*17] failed to come up with useable results in this case, our algorithm was able to handle a density gradient even as steep as at this equator line. Our adaptive mesh tends to become a bit more inharmonious but recovers a closed manifold nevertheless. Examples in Figure 4.13 show another comparison focused on feature alignment where competing results also contain holes due to the same reason. The 2D plane on the right of Figure 4.22 was sampled uniformly over its width and with inverse quadratic decreasing density over its height. In both scenarios, our mesh adapts flawlessly to the change in sample density.



IFAM [JTPSH15] OSR [STJ*17] Ours Ours

Figure 4.22: Meshing results under mixed sample density. Left: 100k samples with 90% on the upper and 10% on the lower hemisphere. Right: A unit plane with vertically inverse quadratic decreasing density.

4.6.4 Limitations

The first part of our procedure, which is the *k*d-tree decomposition, subsampling and hierarchical tile interconnection, succeeds to produce a coherent maze of quad tiles on any given input. However, the default reconstruction algorithm, introduced in Section 4.4.2, properly converges only on object structures of genus zero. Per definition, the algorithm incrementally fills up dead ends of the maze structure until the surface is closed. For objects with holes (genus larger than zero), the mesh will eventually run out of dead ends to fill up. In Figure 4.23 b) the dead ends are actually the cross-sections through the torus. The algorithm will start to close up these cross-sections with caps and converge to the c-shape of genus zero shown in c). However, after insertion of the two quads pointed out in d), the algorithm is able to finish the mesh with genus one.

A critical point in every signal reconstruction is the issue of sample density versus the highest feature frequency. As one can easily comprehend, samples that are placed too sparse can not contribute to reconstructing fine details. In our case, the sampling artifacts show up as dents or wrinkles in the final mesh, e.g., at very small features. Examples of artifacts in meshes can be observed at the ears of the two busts on the right in Figure 4.17. The thickness of the ears and, therefore, the distance of vertices on these surfaces is less than the regular tile size. These unfavorably sampled tiles are interconnected nevertheless, causing the edgy outer ear shapes. Since these artifacts can be justified as sampling issues, increasing the sample density and therefore mesh resolution can resolve these situations. Other small features as dents or bulges are captured and reconstructed with the available resolution: See the small dent on the chin of *Igea* or the characteristic applications on *Nefertiti*'s crown.

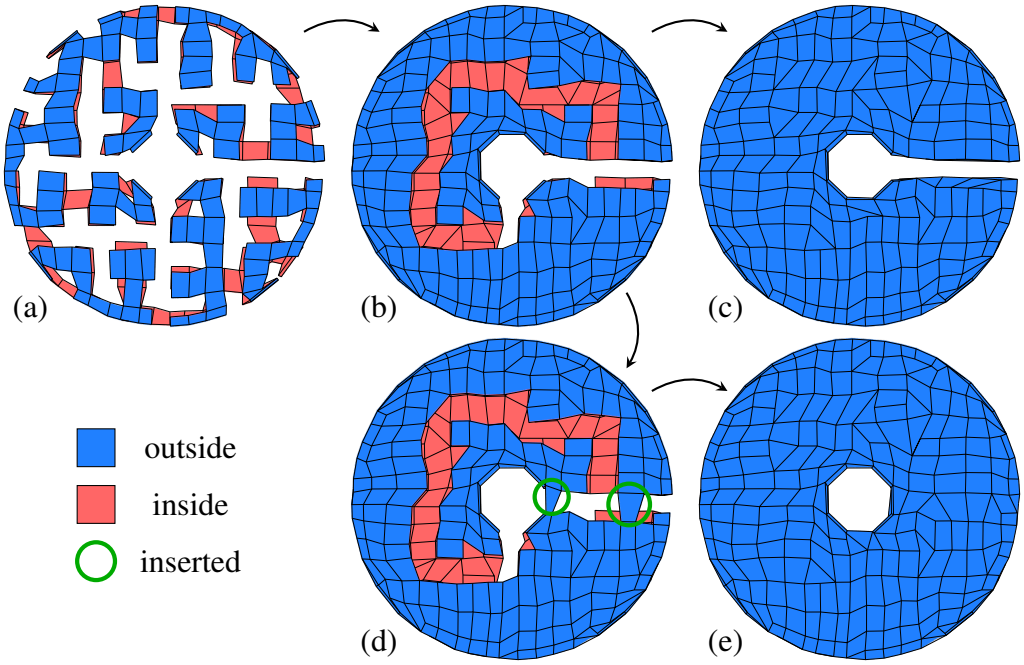


Figure 4.23: With our default algorithm, the torus in the upper row results with genus 0. After insertion of two quads, pointed out in green, the torus is resolved with genus 1.

Results with varying levels of noise are shown in Figure 4.24. For this test, 500k samples on *Igea* (Figure 4.17) are perturbed with random jitter noise of varying magnitude. The noise scales are given in the *in*-row. Values listed in the *out*-row are the means of absolute errors against ground truth, with the object bounding box's diagonal as 100%. With such heavy noise as in the rightmost example, it becomes unclear if samples belong to the same surface and how they can be subsampled. The fill-up algorithm will do its best on the ill-placed tiles but can not guarantee to avoid self-intersections.

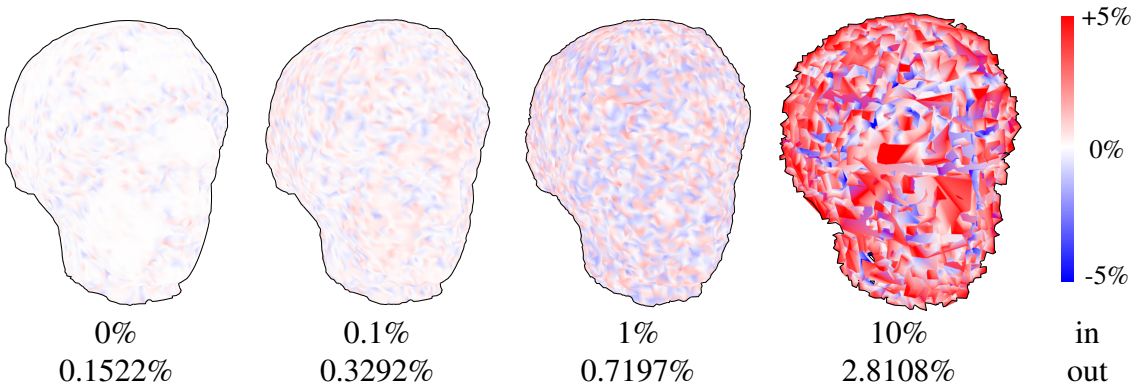


Figure 4.24: Mesh results: 500k samples perturbed with different magnitudes of input noise (*in*). Mean errors (Section 2.5.2) of the reconstruction are listed in the *out*-row.

4.6.5 Performance and Complexity

The algorithm for hierarchical interconnection in Section 4.4.1 derives its complexity from the size of the binary *k*d-tree. However, connectable edges are prefiltered and sorted based on normal directions and reasonable distance to each other. For filtering, the algorithm once has to process possible edge pairs, but due to the simple geometric nature of this operation, the theoretical $\mathcal{O}(n^2)$ complexity is neglectable, nevertheless. The default fill-up approach from Section 4.4.2 maintains a priority queue of edge triples. Each iteration introduces one new edge by closing and removing three open ones. The fewer edges in the queue, the faster the algorithm, so most of the time is spent on the first few steps. Therefore, this algorithm operates in linear time by effectively removing two edges each step. There is no overhead due to global optimization since the maze structure already provides a global basis, and further, only locally optimal solutions are required.

The result shown in Figure 4.18 is one of our most extensive reconstructions with more than 1.4 M samples. The tree decomposition, subsampling and tile interconnection was resolved in less than one minute with Python code on a single CPU. With the same conditions, the fill-up routine consumed less than 10 seconds to resolve 2^{14} open edges.

The data structure of the *k*d-tree has very little memory overhead since point cloud samples are split up and stored without redundancy in leaf node clusters. Mesh information is gathered and stored as sets of four indices for each quad face, indexing the corresponding vertices array. Each node on a level $\leq n-2$ stores four integers $[0,1,2,3]$ with varying permutations. To access a node's tile, a simple power-of-two index offset is determined based on the node's tree-position and added to the indices on-the-fly.

4.6.6 Outlook

Results shown in this chapter, where our procedure is applied as remeshing tool, are based on samples randomly distributed over the surface. However, the limitations in Section 4.6.4 caused by sampling issues can be resolved very easily with a more sophisticated sampling strategy, as demonstrated in Figure 4.17. Samples placed in correlation to surface curvature would cover small features with more samples, so they could be recovered with appropriate mesh resolution. Furthermore, in a remeshing application, an input mesh is given and can be used to optimize our remeshed vertices further.

Figure 4.23 illustrates a method to overcome the topology issue for the reconstruction of objects with a genus larger than zero. This could be automated using our algorithm for hierarchical interconnection, introduced in Section 4.4.1. The algorithm determines reasonable pairs of edges based on their normal orientation and distance to each other. Our incremental fill-up algorithm from Section 4.4.2 could run interleaved with one of these steps every *n*th iteration in order to handle objects of even higher genus.

Many strengths of our technique can be derived from features of the underlying *k*d-tree. Our created maze structure acts as a very powerful abstraction of a point cloud. It is one coherent mesh of $2^{i-1}-1$ quad tiles, guaranteed not to intersect itself and is surrounded by one coherent path of 2^i open edges. The *k*d-tree furthermore represents meaningful neighborhood affiliations and adjacency information. This data could be valuable input for a diversity of machine learning approaches featuring convolutional neural networks, e.g., for classification, mapping or segmentation.

While this procedure is designed to operate on unoriented point clouds, further advancements in terms of reconstruction accuracy could be achieved by incorporating such information, if available, and improving upon the spherical surface projection vertices. The core component of the assimilation method, introduced in Chapter 7, is specialized to do exactly that: As shown in Appendix 7.B, the assimilation can be applied to our quad meshes and successfully lower reconstruction errors.

4.7 Conclusion

The proposed concept presents a novel and innovative way to deal with the task of mesh reconstruction from unoriented 3D scans. Our framework creates watertight quad meshes from noisy input point clouds without any topology information or prior estimated normal orientations. As demonstrated in exemplary scenarios with respect to varying sample density, our method recovers watertight manifold quad-only surfaces where state-of-the-art field-aligned approaches tend to fail. However, compared to these methods when supplied with sufficiently uniform samples, there is still potential to improve our results regarding mesh regularity and the number of singularities. Advanced robustness to noise and more complex topology are important topics that lie beyond the scope of this chapter and have to be explored in future work. Further minimizing the approximation errors is approached with the upcoming method of Chapter 7. Nevertheless, since our resulting meshes are natively free of holes, they are trivially suitable for 3D printing or volume rendering where closed meshes are crucial. Our novel approach to construct tile vertices is not only suitable to approximate rounded organic structures but can also robustly recover coplanar regions. Independent from the approximated mesh, axis-aligned *k*d-tree splits favor regularly distributed vertices. However, special tree nodes also allow to automatically determine the orientation of point cloud segments and adapt the split direction for feature-aligned mesh topology.

Chapter 5

At-Most-Hexa Meshes

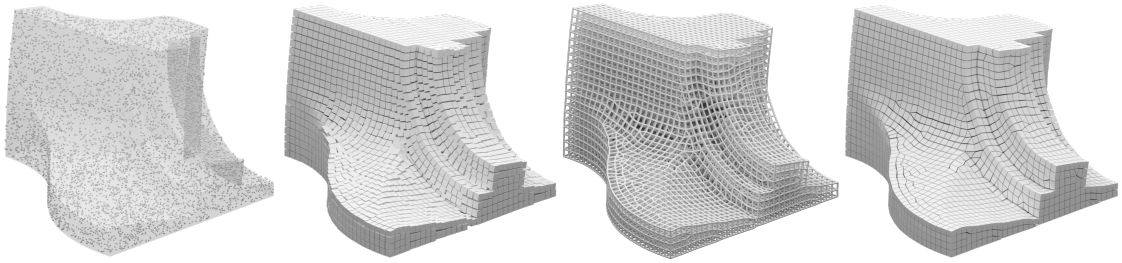


Figure 5.1: Left to right: Stages of our pipeline from input samples to the final volume mesh with 9.5k primitives (89.5% hex).

5.1 Abstract

Volumetric polyhedral meshes are required in many applications, especially for solving partial differential equations on finite element simulations, but their construction bears a number of additional challenges compared to boundary-based representations. Tetrahedral meshes and (pure) hex-meshes are two popular formats in scenarios like CAD applications, offering opposite advantages and disadvantages. Hex-meshes are more intricate to construct due to the global structure of the meshing, but feature much better regularity, alignment, are more expressive, and offer the same simulation accuracy with fewer elements. Hex-dominant meshes, where most but not all cell elements have a hexahedral structure, constitute an attractive compromise, potentially unlocking benefits from both structures, but their generality makes their employment in downstream applications difficult. In this work, we introduce a strict subset of general hex-dominant meshes, which we term “at-most-hexa meshes”, in which most cells are still hexahedral, but no cell has more than six boundary faces, and no face has more than four sides. We exemplify the ease of construction of at-most-hexa meshes by proposing a frugal and straightforward method to generate high-quality meshes of this kind, starting directly from hulls or point

clouds, e.g., from a 3D scan. In contrast to existing methods for (pure) hexahedral meshing, ours does not require an intermediate parameterization other costly precomputations and can start directly from surfaces or samples. We leverage a Lloyd relaxation process to exploit the synergistic effects of aligning an orientation field in a modified 3D Voronoi diagram using the L_∞ norm for cubical cells. The extracted geometry incorporates regularity as well as feature alignment, following sharp edges and curved boundary surfaces. To enforce consistency, specialized neighborhood structures are introduced during the relaxation on a three-dimensional graph structure. The resulting algorithm allows for an efficient evaluation with parallel algorithms on GPU hardware and completes even large reconstructions within minutes.

5.2 Introduction

As mentioned in the previous chapter, due to reduced element count and more harmonic structures, quad-meshes are often preferable over triangular-meshes for specific tasks like solving partial differential equations or CAD applications [BLP*13]. The same rivalry arises for volumetric meshes, where hexahedral meshes are often preferred over tetrahedral meshes [SHG*19]. Hex-meshes feature considerably fewer cells than tet-meshes for the same simulation accuracy, which is a desirable criterion for certain numerical solvers or finite element simulations.

Hex-dominant meshes, where the majority of cells are still hexahedra, are a category of polyhedral meshes that are of interest because previous work [SRUL16, GJTP17] suggests that their construction can be achieved more reliably than in the more complicated case of *pure* hexahedral meshes (where all elements are hexahedra).

We propose to adopt a strict subset of hex-dominant meshes, called *at-most-hexa meshes*, where no cell exceeds six bounding faces, which are at maximum quadrangular, otherwise triangular. More specifically, all cells are either a hexahedron or are any valid polyhedron that can be obtained by starting from a hexahedron and collapsing a few of its edges (see Figure 5.2). Resulting at-most-hexa meshes are still conforming (Section 2.3.4: free from T-junctions).

Conceptually, the motivation for this choice is to ease the construction of the mesh as much as possible by relaxing the definition of hex-meshes while at the same time not sacrificing their usability by the downstream application. A linked motivation is that, because our cells can be considered special cases of hexas, many techniques applicable to pure-hexa meshes can be easily extended to deal with this new type of meshes. For example, internal representations, adjacency structures, and file formats, designed for hex-meshes can be readily adapted (Section 5.3, Appendix 5.A).

The present work exemplifies how the construction of at-most-hexa meshes can be at least as reliable and fast as the construction of hex-dominant meshes. We are motivated by the intuition that at-most-hexa can be easier to process than the more general, hex-dominant meshes, which can feature arbitrary complex polyhedra while still inheriting most of the advantages of pure-hexa meshes.

To this end, we propose a pipeline based on a 3D Lloyd relaxation under the L_∞ norm for a harmonious hexahedral cell layout (Section 2.4.1). Extracted geometry serves as the basis for a graph-matching algorithm to identify all possible at-most-hexa primitives. The final mesh is then assembled with an iterative construction algorithm, prioritizing high-quality (Section 2.3.2) primitives over deformed or smaller polyhedra.

Many recently proposed approaches excel in solving a specific sub-problem that contributes to the overall challenge of hexahedral meshing, but that has to be viewed in the context of what specifically tailored input is required. Our proposed concept is not explicitly designed to supersede all state-of-the-art techniques in every domain but to extend this collection of possibilities with a novel start-to-finish procedure. A CAD-like specification, surface mesh, hex-dominant volume mesh or solely a point cloud as acquired by 3D scans, is already sufficient for our fully autonomous pipeline to produce at-most-hexa meshes. Nevertheless, we demonstrate how our results can compete or improve on some established approaches in terms of hex-quantity and quality.

5.2.1 Related Work

Construction of Pure Hexa-Meshes There is a variety of hex-meshing algorithms dedicated to automate specific steps of the procedure or improve particular criteria of the resulting mesh, e.g., high-shape-quality, low-singularity, feature-aligned or all-hex meshes [LBK16, SRUL16, GJTP17, LZC*18, CAS*19, Tak19, GSP19, LPP*20]. The required input for these approaches is, however, far from trivial to generate and primarily dictates the achieved result quality. Input tet-meshes, parameterizations, mappings, frame fields, or singularity graphs often have to be determined beforehand, sometimes with heavy computation or some manual effort to guide automation in the right direction. The relationship between source-mesh and input frame field also often poses a non-trivial causality dilemma of which to compute, i.e., derive from the other, first.

More details on established general hex-meshing concepts are featured in Section 3.3. One of the most closely related concepts to ours would probably be the approach of L_p-CVT based hex-dominant meshing by Lévy and Liu, a detailed review on the distinctions is featured in Section 5.7.2.

Analogies with other concepts Our approach is inspired by the image stylization technique of Hausner [Hau01] to simulate mosaic images using Lloyd relaxations with an adapted metric and faces similar challenges as the quad-meshing approach by Pellenard et al. [PAM11] but with three dimensions. Furthermore, a smooth object-aligned octahedral flow field [SVB17, GPW*17] is created alongside our relaxation. We extended the method for fast computation of generalized Voronoi diagrams using a GPU [HIKL*99] to three-dimensional space. Our procedure utilizes a specialized graph-matching algorithm to extract at-most-hexa primitives for the final mesh. As our approach relies on faces instead of tetrahedra (Section 3.3, [SRUL16]), the graph-matching search space is significantly reduced to mainly one, at most two, valid traversal paths per primitive type.

Hex-dominant Meshes Other common approaches to hex-construction can be categorized as hex-dominant meshing and have been studied as an interesting and convenient relaxation to pure hex-meshes. In recent works [SRUL16, GJTP17], hex-dominant meshes are obtained by fusing or conglomerating tetrahedral cells into more complex polyhedra, which are often but not always hexahedra. Further distinctions between these approaches can be made, according to the type of polyhedra that are allowed as non-hex cells. In the proposal of Sokolov et al. [SRUL16], polyhedra are limited to quad-based pyramids, prisms, and tetrahedra (plus slivers). Our at-most-hexa meshes are a further relaxation, allowing for these shapes but also others, making the construction considerably less intricate. The concept proposed by Gao et al. [GJTP17] makes no assumptions on the shape of the resulting non-hexahedral cells. While this maximally simplifies the task of construction, the results are less usable for the reasons discussed in the following.

5.3 At-Most-Hexa Meshes

In this section, we define at-most-hexa meshes, which are the output of our construction technique, and outline a few of their favorable characteristics. In our meshes, each cell is definable as the locus of a tri-linear combination of the position of (possibly coinciding) vertices of a hexahedron. In other words, each cell is a polyhedron that can be obtained by starting from a hexahedral cell and collapsing zero or more edges.

In Figure 5.2 we exhaustively list all the cell topologies that can be obtained in this way (up to symmetries). However, not all possibilities are useful for our purposes. Specifically, we can exclude the following cases from further consideration:

• The "*Prysm*" (case 3) is topologically equivalent to the triangular *Prism* (case 6), with an additional diagonal on one quad face. Following the rules for extracting prisms (Figure 5.A.1), prysms would be trivially included, thus result in a subset of all prisms. As prysms would actively promote more triangular constellations and *sliver* elements, the regular prism is always the more favorable option, and prysms are not extracted.

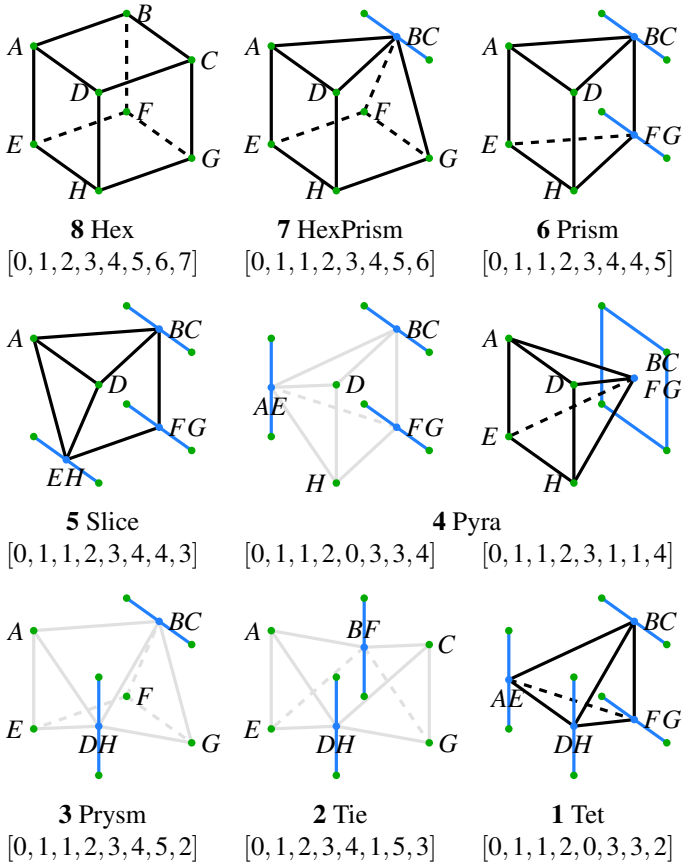


Figure 5.2: A collection of at-most-hexa primitives featured in our meshes (except the grayed-out ones). The list is exhaustive up to symmetries. Each configuration can be obtained by collapsing up to four non-adjacent edges of a hexahedron. This is modeled by duplicating the indices of the list of 8 vertices to a hex.

• The "*Tie*" (case 2) can be seen as the combination of two *Tetrahedra* (case 1) connected over a common edge.

• The "*Pyra*" (case 4) is a quad-based pyramid, featured with two possible configurations to illustrate another intuitive hexa mapping but only one extraction is specified.

Volumetric Definition of Solid Objects Both pure tetra-meshes and pure hex-meshes, but not generic hex-dominant meshes, allow for a straightforward definition of the interior (and the boundary) of the represented object as the union of their cells. A tet cell is trivially defined as the set of all linear interpolations of the mesh vertices at its four

corners; in hexahedral meshes, *and also in our meshes*, all cells are polyhedra which can be defined as the set of the trilinear interpolations of the eight vertices at their corners. Conveniently, the interiors of any two adjacent cells (sharing a face) are disjoint sets, and their union is a *simply connected* locus of points (i.e., no gap is left between them, despite cell faces not being necessarily planar). This construction does not extend trivially to generic hex-dominant meshes because it is unclear how to combine the vertices at the corner of an irregular polyhedron (with non-flat faces). Conversely, our at-most-hexa meshes generalize this situation. The interior of the cell is defined as the trilinear interpolation of the corners, but, in the occasional non-hexahedral cells, a few of its corners are instances of the same mesh vertex. Additionally, when a cell is reduced to a tetrahedron, the above definition is equivalent to a linear interpolation of the 4 surviving distinct vertices, meaning that the proposed structure generalizes both tet- and hex-meshes.

Signal Interpolation Any scalar or vectorial signal sampled at the vertices of a tet- or hex-mesh can be trivially interpolated for any point inside its interior (or boundary). To do so, the same weights used to define an interior point p as a linear combination of its cell‘s corners are employed to linearly combine the signal defined at the same vertices. This results in the definition of a scalar or vectorial field inside the mesh, which is C_0 everywhere (and C_∞ within the cells). Again, this useful principle is directly inherited by at-most-hexa meshes, preserving all the properties, but not by general hex-dominant meshes, as a generalization of barycentric coordinates is not trivial even in 2D [HS17].

Compact Representations Both pure tet- and hex-meshes can be internally represented as *indexed meshes* [BKP*10], a succinct data structure consisting of a set of vertices and a set of cells; tetra cells and hex cells are stored as a sequence of 4 (respectively, 8) indices of vertices at their corners, in some prescribed order. This can be useful for storing the mesh on drives, for example, in interexchange formats. A general hex-dominant mesh does not allow for such representations because non-hex elements have no structure that is known *a priori*. Conversely, cells of an at-most-hexa mesh can be represented the same way as hexahedra, where a few of the vertex indices at the corners (in non-hex cells) are repeated as listed in Figure 5.2. This representation is not only convenient for storing meshes, allowing, for example, to reuse the same file formats of hexa meshes, but also allows for easy application of common hex-meshing operations.

A note on cell convexity Our meshes are analogous to pure hex-meshes, in that their cells are not necessarily convex unless special care is taken to ensure that every quadrangular face is exactly planar. While face planarity is implicitly pursued as a soft objective by our construction strategy, we did not experiment with its strict enforcement. The same issue arises with quadrangulated surfaces, where strictly enforced solutions have been proposed with planar-quad-meshes [LPW*06].

5.4 Overview of the Meshing Algorithm

The key characteristics and motivations of our proposed meshing approach can be briefly summarized as follows:

Simple, versatile input We do not assume input volume data, surface or volumetric parameterizations, frame-fields, or a consistent meshing of the surface. Hull meshes or sparse surface samples of arbitrary size and resolution are sufficient. Our method *can* be guided by an input orientation field if one is available, but this is not required.

At-Most-Hexa The primitives in our result meshes never exceed the base case of a hexahedral cell, which, as described before, comes with many benefits over general hex-dominant meshes. We provide explicit graph-matching routines with minimal branching for efficient primitive extraction.

Object hull Our volumetric cells materialize as final mesh primitives, thus there is no need to compensate for hull shrinkage as it is common with Delaunay graph-based meshing, e.g., in L_p-CVTs [LL10]. While this is trivial when the input is a closed mesh, we can also guarantee closed and feature-aligned result meshes starting from point cloud input. The employed kNN-graph provides simple and robust in/out labels, even for complex objects of higher genera.

Alignment Initial orientations are sampled and extrapolated from the input, then aligned to an orthogonal vector field, which is optimized during the process.

Regularity and isometry The method strives to obtain regular meshing. Most cells are hexahedral, most edges are regular, and cells are equally sized and well-shaped.

Implicit parallelism Our method leverages several sub-steps, such as constructing the kNN-graph, relaxing the Voronoi diagram, and the graph matching, that can rely on massively parallelized implementations on the GPU.

5.4.1 Steps Breakdown

The diagram in Figure 5.3 schematically outlines the basic steps of our pipeline:

▶ **Input Preparation:** The volume of the input object (plus margins) is populated with points (sites) on a regular 3D grid, serving as seeds of a Voronoi diagram. Each site creates a cell in the diagram and will generate one at-most-hexa cell in the final output.

▶ ***k*-Nearest-Neighbor Graph:** This graph stores the nearest-neighbor relationships between sites and is crucial for an efficient evaluation of all subsequent steps. This is greatly beneficial for an efficient evaluation of the Lloyd relaxation, interpolation and propagation tasks, and eventually for extracting geometry information.

▶ **Orientation Field:** Orientations from the input hull or point cloud are extrapolated once and further on jointly aligned during the relaxation following parallel and orthogonal flow constraints.

▶ **Lloyd Relaxation on Hex-like Cells:** By exchanging the Euclidean norm (L_2) in the Lloyd relaxation with the Chebyshev norm (L_∞), the cell structure becomes more cubical (Section 2.4.1), which is suitable for further interpretation as a hex-like mesh structure.

▶ **Geometry Generation:** Geometry is established by materializing the hex-like cells as proportionally scaled unit cubes. Vertices and edges are created with a simple match-and-merge operation.

▶ **Topology Extraction:** With geometry established, tri- and quad-faces are collected to form the base for all at-most-hexa primitives. The primitives are then collected via graph-matching, using small dedicated state-machine algorithms, effectively minimizing the required search space.

▶ **Output:** The final assembly routine allows for prioritizing regular hexahedra of high quality. Therefore, result meshes solely feature the at-most-hexa primitives listed in Figure 5.2 where the absolute majority are hexahedra.

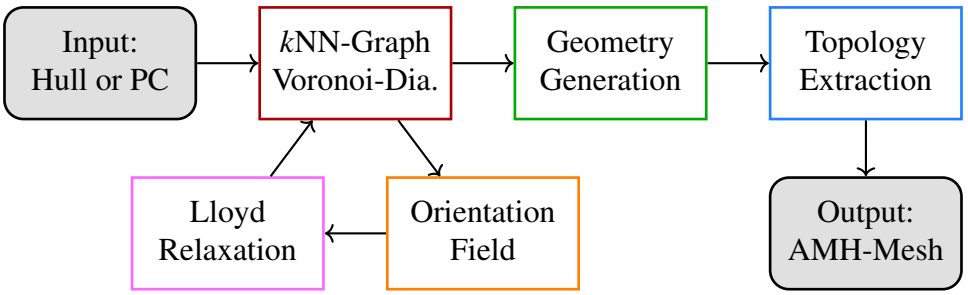


Figure 5.3: The first three points around the relaxation are covered in Section 5.5. Mesh generation steps are both elaborated in Section 5.6.

5.4.2 Terminology

To counteract misunderstandings, we first want to establish a uniform terminology for involved entities: We will refer to the points in a Voronoi diagram as *sites* while the same point is a *node* in the associated graph structure. A *cell* refers to the associated space around each site. Points on the input surface, or point cloud points themselves, are referred to as *samples*. For simplicity we call these entities s_i in all instances and the unique index i may also be used to identify corresponding properties like depth d_i, in/out-label l_i or a normal n_i. In the upcoming section for geometry extraction, cells will be materialized with simple scaled unit cubes, each defined with eight *virtual* vertices. As the geometry extraction progresses, *virtual* vertices are merged to *real* ones. The geometric entities emerging from the topology extraction process feature hexa-, tetra-, and other polyhedra, collectively called *primitives*.

5.5 Relaxation

The relaxation is essential to capture characteristic features of the input. Therefore, the space inside and outside of the input object is populated with volumetric cells, which eventually align with the model's shape and curvature. Rather than relying on a predefined frame-field providing the orientation for volumetric cells, our relaxation optimizes the orientation and position of all sites in a joint process. Besides alignment to surface features, the goal of this stage is to obtain equally sized and cube-shaped cells.

Therefore, the necessary optimization is performed similarly to a Lloyd relaxation with specific constraints to favor the generation of hex-like cell structures. Eventually, the relaxation process outputs sites with optimized location and orientation, defining the input to the mesh extraction stage of Section 5.6.

5.5.1 Voronoi Diagram

The basis for the Lloyd relaxation is the underlying Voronoi diagram, which computes on a face-centered cubic (fcc) lattice [CS98, HAB*17] with at least 12^3 times more lattice points than sites in the diagram. This has proven to be a sufficiently high resolution that is still practically feasibly with limited GPU memory. Further, the fcc lattice is preferable over a regular cubical grid to avoid axis-aligned bias but is in contrast to lattice-guided approaches [YS03, NZH*18] merely a convenient way to label space. The diagram is computed on the GPU using a z-buffer [HIKL*99] extended for three dimensions. Obviously, the publication Meshless Voronoi on the GPU [RSLL18] comes to mind. But, as elaborated in the following, our distance metric is not orientation-invariant as in a standard Voronoi diagram, which drastically complicates the integration of a cell. Therefore, this concept is not trivially suitable for our objective.

In order to propagate information between all sites $\mathbf{S}$, the relaxation relies on two different mechanisms: Lloyd iteration to optimize site positions and cell extents and a kNN-graph to align orientations and to eventually promote hex-favorable grid structures.

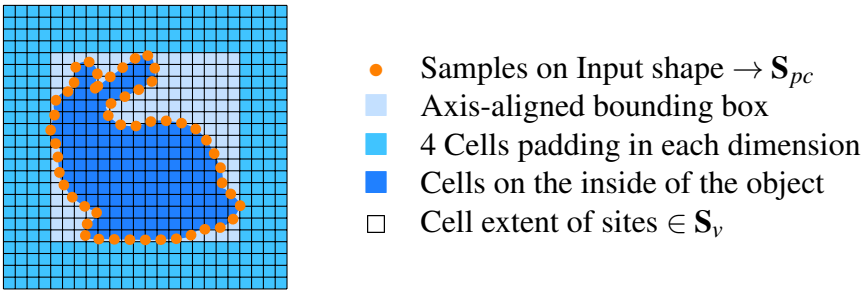


Figure 5.4: A low resolution 2D example on how the input object is populated by sites on regular initialization positions.

Site Population The first step is to populate the entire bounding box plus margin with sites $\mathbf{S}_v$ initialized on a regular or jittered grid filling the entire volume as shown in Figure 5.4. The whole set of sites in the diagram consists of two disjunct subsets $\mathbf{S} = \mathbf{S}_{pc} \cup \mathbf{S}_v$. Sites of $\mathbf{S}_v$, which are close to the hull, spawn a second set of surface samples $\mathbf{S}_{pc}$ positioned directly on the input hull. In Appendix 5.B we propose to replace the $\mathbf{S}_{pc}$ set with an actual point cloud as an alternative to meshed input. However, for now, $\mathbf{S}_{pc}$ solely serves as query points for the orientation extrapolation.

Input Hull As the relaxation treats all cells equally, there is no distinction between cells in or outside the object. But in the end, only the inside cells are relevant for further use. Therefore, inside-outside labels for all cells are determined using fast winding numbers [BDS*18] of the input surface. In most hex-mesh application scenarios, it is perfectly reasonable to assume object boundaries given in CAD or meshed form. Nevertheless, in the context of this thesis, we also propose, in Appendix 5.B, the option to use point clouds as input for volume mesh extraction. In both input scenarios, the mesh or point cloud hull act as a natural boundary during the relaxation, limiting the individual cell's extents and protects them from crossing the hull.

Distance Metric As elaborated in Section 2.4, in a general Lloyd relaxation, sites thrive to increase the distance between each other, which eventually creates cells (primitives) of homogeneous size and maximum mesh isotropy. In the 2D example of Figure 2.5, using the Euclidean L_2 norm, the majority of relaxed cells resemble hexagons (like a honeycomb) because this is the densest 2D packing of circles [CW10]. Consequently, we employ the Chebyshev L_∞ norm for our relaxation: 2D cells would now approximate squares [Hau01, MB12] and respectively, 3D cells actually become cubical [LL10, BRM*14]. In each Lloyd relaxation step, sites are updated with the geometric center of their cell, computed as the averaged position of their labeled lattice points.

5.5.2 *k*NN-Graph

In order to allow for fast information propagation during the relaxation, we incorporate a decentralized network between the sites, namely *k*NN-graphs where each site is linked to its k nearest neighbors. Particularly, N_{26} and N_6 neighborhoods are used. N_{26} is purely based on geometric distances. $k = 26$ corresponds to $3\times3\times3-1$ cubes stacked in a 3D grid. In irregular arrangements, N_{26} might also contain sites that are not direct neighbors, but this hardly impacts the optimization. In experiments with larger ks, the most notable impact was that the performance decreased, as a larger neighborhood had to be considered. The improvements in terms of mesh homogeneity or regularity were not significant. In Section 5.5.4, another N_6 neighborhood is used to promote hexahedral grid alignment. Whereas N_k includes neighbors based on their geometric distance alone, N_6 also incorporates a site's orientation: It features only the most suitable six neighbors from each direction (left, right, up, down, front, back) at an edge-length's distance e. As formulated in Equation (5.1), the N_6 can be derived as a subset of the N_k where M_i is a site's orientation and $\vec{r}$ corresponds to the coordinate axes.

$$N_6(i) = \left\{ \min_{\substack{j \in N_k(i) \\ \vec{r} \in [\pm x, \pm y, \pm z]}} || \left(s_i + (M_i \cdot \vec{r})e\right) - s_j ||_2 \right\} \tag{5.1}$$

Furthermore, each node also maintains an n-hop-distance d_i (Equation (5.2)), which counts the number of steps required to reach the closest sample nodes on the hull.

$$d_i = \begin{cases} 0 & \text{if } s_i \in \mathbf{S}_{pc} \\ \min\limits_{j \in N_k(i)} [d_j] + 1 & \text{else} \end{cases} \tag{5.2}$$

Construction & Maintenance The kNN graph is initialized by setting the neighbors of each node randomly. With a simple parallel update routine on all nodes $\mathbf{S}$, the randomly initialized graph becomes an actual nearest neighbor graph:

- For node s_i collect the neighbors of all neighbors.

  $N_k^2(i) = \bigcup_{j \in N_k(i)} N_k(j)$.

- Sort by geometric distance $||s_i - s_j||_2$ where $j \in N_k^2(i)$.

  $\mathbf{N}_k^2(i) := \text{sort}(N_k^2(i))$

- Update $N_k(i)$ with the first k elements in $\mathbf{N}_k^2(i)$.

It can be shown [DML11] that only 7 update steps are required to get an almost perfect kNN approximation from random input connections. As the relaxation progresses, sites in the Voronoi diagram change their position with every step, and therefore the graph also has to be updated with every iteration. However, once the graph is established, neighborhood fluctuation is marginal and usually only one, or to be sure two, update cycles have to be performed.

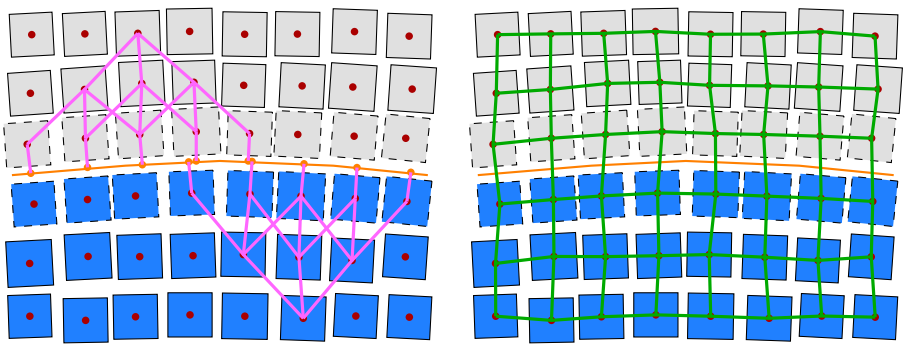


Figure 5.5: Nodes with $d_i = 1$ (dashed) determine their initial orientation from samples on the hull. Nodes of $d_i > 1$ derive theirs from neighboring nodes closer to the surface using portions of the N_{26} graph. The right side shows the N_6 graph, transcending the outer hull, so that adjacent cells on the in- and outside are able to align.

5.5.3 Constrained Relaxation

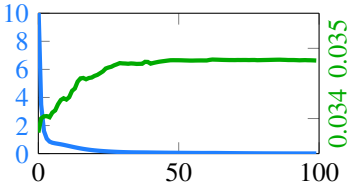
The Lloyd relaxation process is basically an iteration alternating two steps: 1. compute a Voronoi diagram based on the given site positions, 2. reposition each site to the geometric center of its cell. But as our employed distance metric is no longer orientation invariant, we also have to maintain individual orientations for all sites.

Orientation Initialization Each node in our graph carries its own orientation, defined by the three orthogonal base vectors: normal, tangent and bitangent, represented for interpolation by quaternions. See Section 2.2 for more details.

For samples, $\mathbf{S}_{pc}$ on the input hull, orientation is determined as the surface normal plus principal curvature vectors [PdC76, Rus04]. As initialization, the orientations of $\mathbf{S}_{pc}$ are extrapolated once through the volume for all sites in $\mathbf{S}_v$. This is done in a wave-front propagation manner [OBB*13] over the discrete node positions of the graph. On the left of Figure 5.5, portions of the N_k graph are shown. This illustrates how discrete n-hop-distances d_i as well as real geometric distances are employed to weigh individual orientations during propagation.

Maintaining Orientations During relaxation, the orientation *and* the spatial arrangement of neighboring cells jointly align, resulting in the best fitting constellation with respect to the geometrical constraints imposed by the input hull. In contrast to a predefined volumetric frame-field, orientations emerge from the alignment itself and are bound to the individual cells and their discrete site positions. Orientations of adjacent neighboring sites are optimized to be consistent, herein defined with invariance to axis-permutating rotations. This constraint is beneficial for the mesh extraction step in Section 5.6, where neighboring cells shall become adjacent primitives, forming hex-like structures. Therefore, in each iteration, the orientation of a site is aligned to a distance-weighted combination of all orientations from its neighbors in N_k, using quaternions as described in Section 2.2.4. Eventually, the base vectors for each site point in close-to parallel or orthogonal directions compared to their neighbor sites (regardless of their signs), resulting in a smooth and homogeneous orthogonal flow field throughout the whole volume.

Convergence Whereas Lloyd relaxations are known to converge to Centroidal Voronoi Tesselations using the L_2 norm [DEJ06], it has yet to be shown that the same holds for higher dimensions or other norms. However, in practice, we could not provoke scenarios that showed tendencies of non-convergence or one that resulted in a bi-stable state.



This graph plots the accumulated movement of all sites and the average volume of their cells over 100 relaxations. While the movement drops below numerical accuracy within the first 50 iterations, the volume also approaches a steady fix-point.

5.5.4 Regularization

So far, sites $\mathbf{S}_v$ freely move around during the relaxation, maximizing the distance to each other and orient themselves accordingly. However, sites should be positioned to form a grid for the upcoming geometry extraction step if possible. The constellation on the left in Figure 5.6 resembling a brick wall is not unlikely to emerge with aligned orientations alone and without positional constraints.

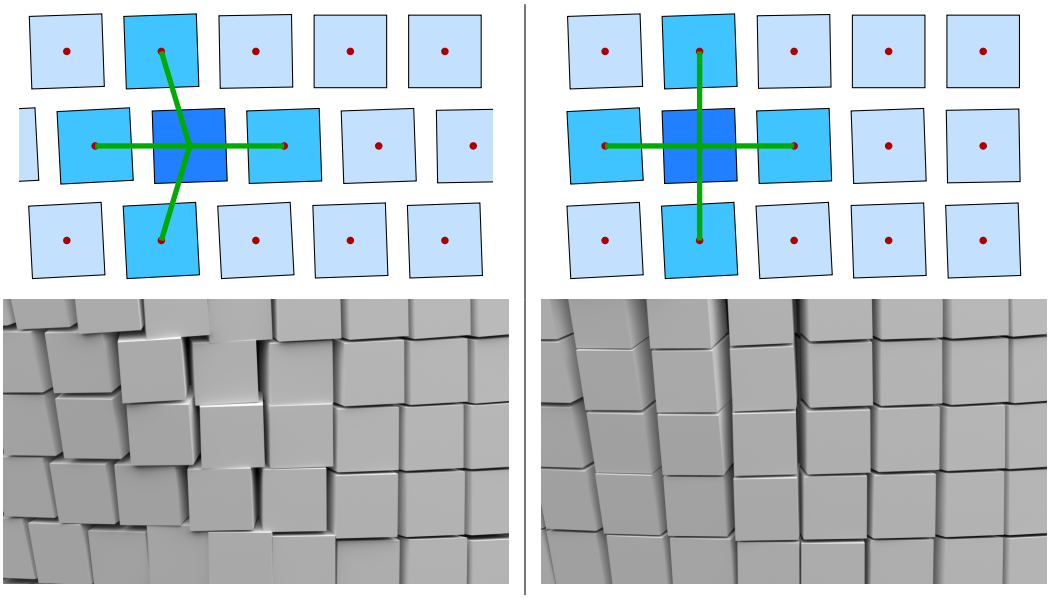
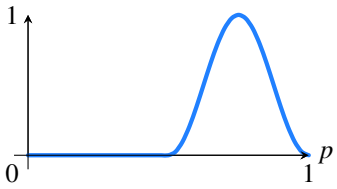


Figure 5.6: Coherent orientation is no guarantee for valid alignment. The N_6 graph reintroduces regularity to the relaxed system. Bottom: Unit cubes of an unconstrained and a regularized relaxation result.

To counteract this brick-wall alignment, we introduce a positioning scheme using the N_6 neighborhood. Equation (5.3) formulates the updated center c as weighted sum of the Voronoi cell's geometric center c_g and the center of its six neighbors c_{N_6}. For constant $w(p) = 0$, the process is equivalent to Lloyd's algorithm.

$$c = (1 - w(p)) \cdot c_g + w(p) \cdot c_{N_6} \tag{5.3}$$

Here it is crucial to note that N_6 neighborhoods strictly exclude hull samples ($N_6 \cap \mathbf{S}_{pc} = \emptyset$). As a result, the effect shown on the right in Figure 5.6 can benefit from outside cells too, as illustrated in Figure 5.5. Improved results can be achieved using a variable weighting function that changes over the course of the relaxation.



$$w(p) = \frac{1}{2} - \frac{\cos\left(4\pi\left(\max(\frac{1}{2}, p) - \frac{1}{2}\right)\right)}{2} \tag{5.4}$$

Heuristic experiments suggest letting the first half of the relaxation run based on c_g centers alone, then increase the contribution of c_{N_6} centers (forcing the sites to from a hex grid) with a cosine curve peaking at 75% of the procedure and have them converge to 0 again towards the end of the relaxation. Other strategies for $w(p)$ like a linear, squared, quadratic or sinusoidal decrease, increase, or both (as a peak) are obviously possible but were outperformed by the curve expressed in Equation (5.4) with progress $p \in [0, 1)$.

Split Cells Some geometry might cause unfavorable constellations in the relaxed graph, like neighborhood clusters. A cluster occurs if a node is considered as a direct neighbor by more than 6 other nodes. As the N_6 graph is constantly updated during the relaxation, one can quickly determine and resolve such clusters by splitting the affected node. But if a node is split too early, for example, with a cluster size of 7, the two resulting split nodes will have an underpopulated neighborhood, which is why we chose a split limit of 10. The to-be-split node is replaced by two new nodes, inheriting its neighbors and linking to each other. Their geometric position is based on the split-node's site position, shifted in the positive or negative direction of the cells principal direction vector, respectively.

5.6 Mesh Extraction

The focus of this section is the post-relaxation domain, schematically outlined in Figure 5.7, which is to extract the geometry and topology of the mesh from the relaxed sites. Since every site position represents the center of a primitive, the relaxed Voronoi diagram, hence the kNN graph, only gives the dual of the actually anticipated hex-mesh, which should be defined by its vertices. Geometry and topology are gathered for the at-most-hexa mesh by utilizing all the information that was accumulated for each site during the relaxation: 3D position and orientation, cell extent and volume, N_6 direct neighbors, in/out state, and n-hop distance. Simply put, our approach is to place actual hexahedra in all cells and fuse them wherever trivially possible. A closed mesh can still be assured by introducing non-hex primitives where necessary.

The strict mechanisms for extracting the at-most-hexa mesh are designed to make inverted primitives impossible. Thus our result mesh does not contain negative Jacobians.

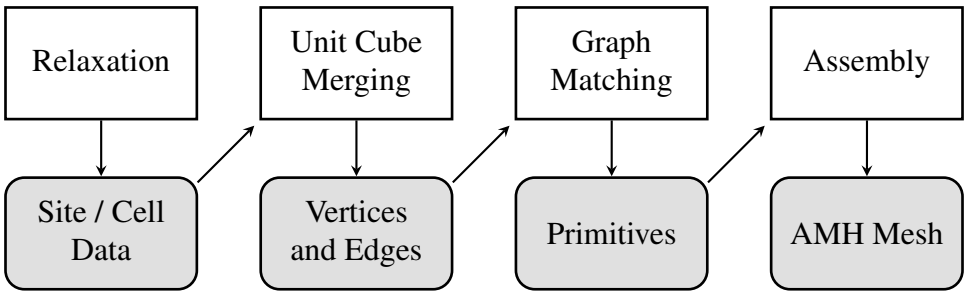


Figure 5.7: The post-relaxation domain: The extraction of vertices and edges from relaxed sites (Section 5.6.1) is followed by the collection and assembly (Section 5.6.2) of at-most-hexa primitives for the final mesh.

5.6.1 Stage One: Geometry

The geometric basis for the further steps is based on the materialization of all Voronoi cells with small cubes centered on their sites' position. This contrasts previous L_p-CVT based hex-dominant meshing concepts [LL10, BRM*14, SRUL16], which solely operate on the bi-graph of the relaxed diagram, by assembling hexahedral primitives from the Delaunay tetrahedralization. For our approach, each cube is scaled uniformly to approximate the extent of its corresponding cell and is rotated to the site's orientation. Boundary cells are ensured to grow such that the boundary faces align with the input hull or point cloud. This stage is illustrated in Figure 5.8 and as example in Figure 5.1 (center, left).

One can easily comprehend how two neighboring cubes should be connected: Take the quad of each cube that is facing the other cube and merge them into one. The eight *virtual* vertices of these two *virtual* quads shall become four *real* vertices of one *real* quad.

This task may sound fairly simple, but it is rather complex to determine robust connections geometrically. The left of Figure 5.8 shows relaxed cells and the right illustrates how uniform cubes are placed in this scenario. A simple snap-merge approach could succeed on very regular structures, but as soon as cells approximate curved surfaces, the distances between potential merge partners vary heavily due to keystone deformations of the cells. A merge criterion only based on geometric distance is therefore not very robust.

To approach this issue in the merging step, our algorithm incorporates topology information provided by the six nearest neighbors N_6 of each node. If there is a mutually unique link established between two cubes, they can be trivially interconnected. However, while the regularization in Section 5.5.4 largely improves the number of mutually unique relationships, they cannot be established everywhere. Some node might be considered as a neighbor to fewer or more than six other nodes. Those complicated cases will be considered after the trivial cases.

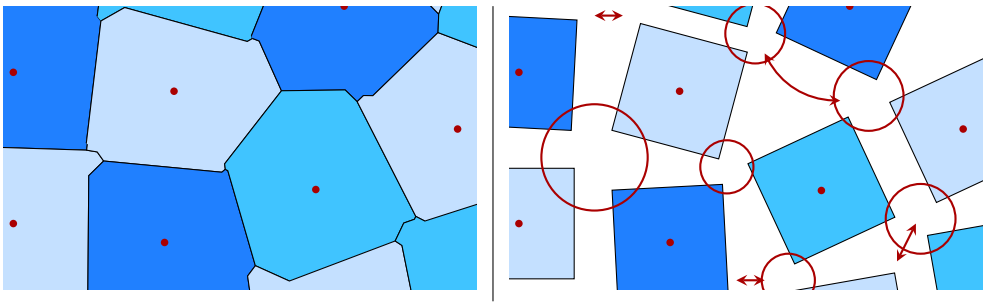


Figure 5.8: Cells (left) and materialized cubes (right). Due to keystoning, the distances between *virtual* vertices may vary significantly, thus are not a robust merge criterion.

Matching Scores In order to find the best suitable matches of the cube's quads to be merged, we first determine a score for each pair. The merging order is therefore determined by a global score-sorted priority queue over all possible faces that could fuse two cubes. For each face of a cube, the algorithm queries suitable faces of all six neighbors and computes a geometric score φ as formulated in Equation (5.5), expressing how well the two cubes match.

$$\varphi(U,V) = \left(2 - \frac{\arccos(\vec{n}_u \cdot \vec{n}_v)}{\pi}\right) \sum_{i \in (0,1,2,3)} ||u_i - v_i||_2 \tag{5.5}$$

where u_i are vertices of face U, v_i vertices of face V and $\vec{n}_u, \vec{n}_v$ are the face normals respectively. Assuming the best suitable permutation for the vertices on the other face has been determined, the scoring function computes the accumulated pairwise distance between face vertices scaled by an angular component based on the face normals. For normals ideally facing each other, the angular factor is 1, and it can grow up to 2 for face normals pointing in the same direction. Face pairs and corresponding neighbors are collected in the priority queue, sorted by ascending score values. Faces for which there is no inside neighbor are labeled as part of the outer hull of the final hex-mesh.

Vertex Merging With the face-merging order in place, we have to find a suitable way to merge vertices. In a completely regular scenario, the corners of eight cubes would make up one vertex for the final mesh. However, in an irregular arrangement, sometimes more or fewer than eight *virtual* vertices make up one *real* vertex. Positions of *real* vertices in the final hex-mesh are set to be the geometric centers of the associated sets of *virtual* vertices. By working off the priority queue, the information which *virtual* vertices are to be merged comes in serial form and has to be completely assembled before the vertices can actually be combined. Consequently, the merging process itself is split up into two steps: In the first run, all merge-relevant information is collected and the *virtual* vertices are accumulated into merge-sets. In the second run, all merge-sets compute the *real* vertex position for the final hex-mesh.

Processing the priority queue As mentioned before, the majority of all cubes can be connected in a straightforward way, but some may require extra care. Our implemented algorithm to process the priority queue follows a simple defensive strategy promoting only high-quality mesh output. Therefore, a bit-field is maintained, which keeps track of already merged and unmerged faces. If a face pair is up-next in the priority queue and one of the faces is already flagged as merged, the pair is simply skipped and left open.

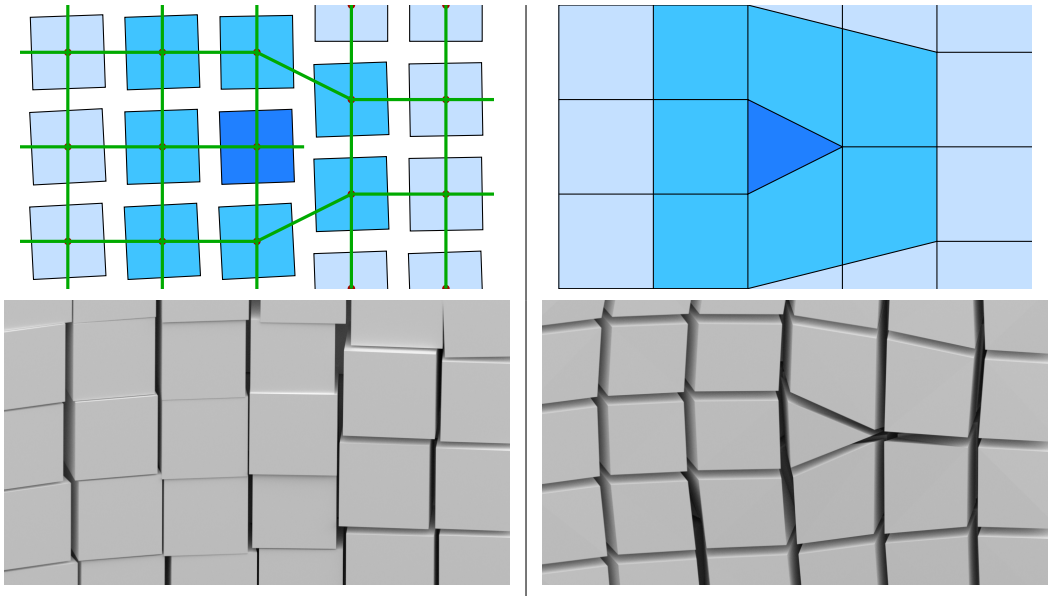


Figure 5.9: Despite the N_6 regularization, ambiguous scenarios as on the left may still occur. In the result on the right, a face is collapsed to an edge by automated matching.

Some of the skipped faces, e.g., as in the ambiguous assignment problem in Figure 5.9, are resolved implicitly by the vertex merging step. A cube's edge can collapse if two *virtual* vertices are in the same merge-set. If this occurs on two opposing edges of a quad-face, one side of a cube is collapsed to an edge.

Closing Cuts So-called *cuts* occur when there was a gap in the neighborhood topology at the time when matching scores were computed. These topological gaps are totally valid and fairly easy to fix by running another matching-score iteration before merging the vertices. If two cubes were not direct but indirect neighbors, it might happen that two of their *virtual* vertices will be together in the same two merge-sets. If those two merge-sets form an edge that connects open faces from these cubes, they will be considered as neighbors now and the faces can be merged as well. Figure 5.10 illustrates this process in theory. A very prominent result can be found in Figure 5.17 on the long rounded vertical corner of our *Fandisk* result in the rightmost image.

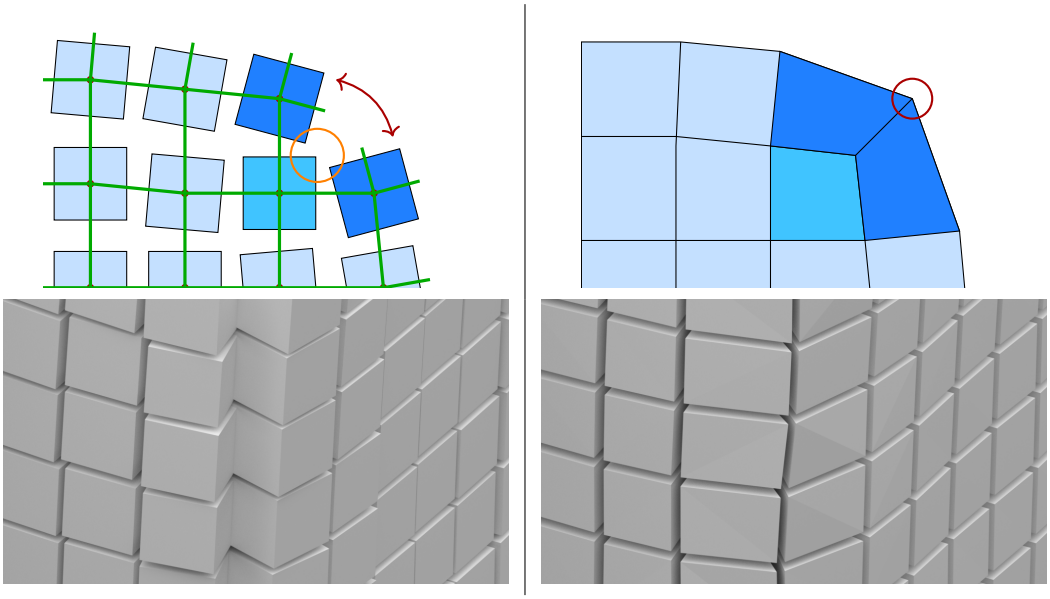


Figure 5.10: The dark-blue cubes have dangling nodes (green) in their N_6 neighborhood. Indirect merges (red) are possible over common edges (orange) at small enclosing angles.

After these steps, the positions of all *real* vertices are computed by averaging over all *virtual* vertices contained in their respective merge-sets. In some cases, the vertices generated by a merge can be a bit off from the anticipated surface. For example, the narrowing geometry of the *Jumpramp* in Figure 5.19 provokes such merges, which in combination with the concave 90° edge, can cause displaced vertices. However, a simple optimization step, pulling vertices onto the input surface (or a $\mathbf{S}_{pc}$ surface patch in case of point cloud input) would allow for an easy fix in such scenarios. We exemplify this later with our mesh assimilation technique, introduced in Chapter 7. The experimental results in Appendix 7.B show significantly improved HD and RMSE measures.

5.6.2 Stage Two: Topology

This is the point where actual mesh topology comes together. So far, the simple match-and-merge algorithm has only generated the final mesh vertices and associated edges derived from the merged cube structures. This edge-network already features trivial triangular and quadrangular faces, suitable for at-most-hexa primitives. However, this edge-complex is not yet entirely suitable for our objective as it also includes structures that can not be resolved with at-most-hexa primitives. We can identify such regions as spirals, shown in Figure 5.11, and resolve them by inserting additional edges. This can be done simultaneously to the identification of trivial tri- and quad-faces (I). On this basis, small state-machine programs, as shown in Figure 5.A.1, extract all possible at-most-hexa primitives (II) from the edge-network. The final mesh is then assembled (III) from a quality-oriented priority queue.

I Collect Faces

Specified at-most-hexa primitives are based on triangular and quadrangular faces found in the given edge-complex. Triangular faces are identified as loops of three adjacent edges or four edges for quadrangular faces, respectively. However, there are also cases where adjacent edge-paths do not form closed loops of length three or four. Topological miss-configurations, as shown in Figure 5.11, can be caused during the relaxation in narrowing geometry and can not be resolved using at-most-hexa primitives, thus would cause holes and missing primitives in the final mesh. Spiraling edge structures are identified and broken down, therefore, becoming tri- and quadrangulatable. In an iteration over all edges, a set of penta-loops is gathered, defined as five adjacent edges, not interconnected by any other existing edge. Spirals can always be broken down into one or more (overlapping) penta-loops, which can be easily split up by creating five new interior edges per loop. Regardless of being a prior existing or newly added edge, they are only featured in the final mesh as part of a fitting primitive. Furthermore, it is suitable to split very skewed or non-planar quad faces into two triangles by inserting a diagonal edge in some cases. For trapezoidal quads, we chose $\frac{\pi}{4}$ as the lower limit for corner angles and $\frac{\pi}{3}$ as the upper limit between opposing edges on trapezoidal quads. The shortest diagonal of such a quad is then added and further on treated equally to prior existing edges. Explicitly checking for dihedral angles is not necessary as these both measures already robustly identify disfigured quads.

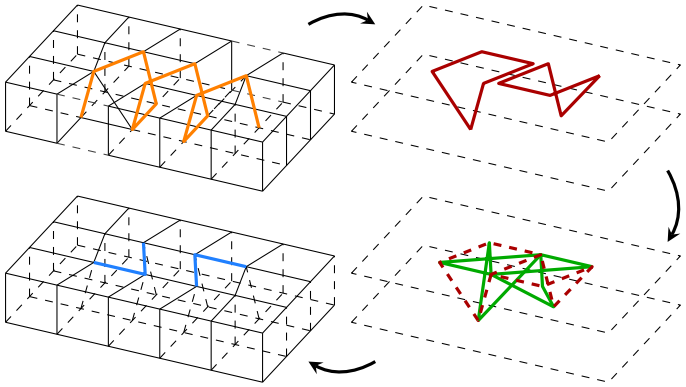


Figure 5.11: Spiral structures can not be represented with at-most-hexa primitives. Identified penta-loops are supplemented with additional edges, which are only used where needed in the final mesh.

II Graph Matching Algorithm

Based on the collected tri- and quad-faces, our algorithm is now able to generate the basic building blocks for the final mesh assembly, namely the at-most-hexa primitives. This poses a straightforward graph-matching task, but the greedy search's complexity escalates quickly if not approached with care. With insight into possible outcomes, one can specify a strict set of rules to prune the search tree drastically and avoid incredible amounts of redundancy early on.

The algorithm in Listing 5.1 iterates over the available faces, generating the individual primitive types. The outer loop is parallelized, such that all primitive types are processed at the same time. Pruning is achieved as the algorithm obeys the following rules:

- Every face f_i is considered as a possible starting point to assemble a primitive.
- Once a face f_i was a starting point, all primitives featuring f_i have been explored.
- Search paths from other starting points f_j, including f_i, result in redundant results.
- Therefore, used starting points f_i are marked and allow for an early termination of redundant search paths f_j.

The state-machine programs featured in Figure 5.A.1 are designed for as little branching as possible. Consequently, a triangular face is favorable over a quad as a primary face due to only three open edges, hence branching directions; except for the hexahedron obviously and the pyramid due to symmetry. This preselection of suitable tri/quad faces is also implemented in the following algorithm. Our face-based state-machines can be formulated with as little as only *one*, or at maximum *two*, possible assembly sequences. This contrasts common tet-based hex-assembly [MT00, LL10, BRM*14, SRUL16], where the hexahedron alone can be formulated in 10 different constellations, featuring 5, 6 or 7 tetrahedra plus sliver elements, as illustrated in Section 3.3.

```
1  primitives = {}
2  facesUsed = {}
3  for primType in [8,7,6,5,4,1]:
4      primitives[primType] = []
5      facesUsed[primType] = zeros(numFaces)
6      for fi, face in enumerate(faces):
7          if face is quad and primType in [8,4] \
8          or face is tri and primType in [7,6,5,1]:
9              newPrims = findPrims(fi, primType)
10             primitives[primType] += newPrims
11             facesUsed[primType][fi] = 1
```

Listing 5.1: Collecting primitives: Loop over all faces with specialized state-machine algorithms, as shown in Figure 5.A.1. The outer loop may be parallelized, for collecting the individual at-most-hexa primitive types simultaneously.

The algorithm in Listing 5.2 generates at-most-hexa primitives from a given starting face f_i, as schematically illustrated in Figure 5.A.1. While there are unfinished primitives in the `openPrims` list, the algorithm queries for adjacent, unused and type-suitable candidate faces f_j (line 7). New candidate faces explicitly qualify themselves by sharing one of the open edges in the unfinished primitive. A new primitive struct is generated by adding face f_j to the open primitive (line 8). If this leads to a complete primitive, it is added to the result set (line 10). If the new primitive is not yet complete but a valid state, it is added to the set of open primitives for the next round (line 13). If neither case is satisfied, the new primitive is an invalid state and rejected. For the same starting face f_i, the algorithm may return multiple primitives, i.e., two hexahedra sharing a common quad face, but never any duplicates.

```
def findPrims(fi, primType):
    donePrims = []
    openPrims = [[fi]]
    while len(openPrims):
        newOpenPrims = []
        for openPrim in openPrims:
            for fj in cFaces(openPrim, primType):
                newPrim = openPrim + [fj]
                if done(newPrim, primType):
                    donePrims.append(newPrim)
                    continue
                if valid(newPrim, primType):
                    newOpenPrims.append(newPrim)
        openPrims = newOpenPrims
    return donePrims
```

Listing 5.2: Building primitives: The search for all possible primitives of a given type including a specified starting face.

For the upcoming assembly routine, the at-most-hexa primitives are sorted by quality within their class. Minimum Scaled Jacobians (MSJ) are employed as an intuitive quality property. Not all primitives support this property as trivially as the hexahedron with 8 suitable vertices. However, as the primitives are derived from a hexahedron by collapsing edges, each primitive can be trivially mapped to a unit cube. Collapsed edges result in a Jacobian of 0 on affected vertices. Therefore, Jacobians are computed on non-hex primitives only where possible, namely on vertices connected to three edges.

III Assembly

With all available primitives at hand, we will now focus on the concept of assembling a full mesh. The upper histogram in Figure 5.12 shows the proportional amount of the different types from *all* collected primitives. As illustrated, each collection of primitives is internally sorted by quality (MSJ). With the following algorithm, the final mesh is assembled by incrementally adding the best suitable primitives to the existing set as exemplified in Figure 5.13. The relaxation produces a hex-dominant mesh, where the majority of hexahedra can be adopted on-the-fly to initialize the assembly's starting point in Listing 5.3: All available hexahedral primitives (type **8**) with a certain quality (MSJ > 0.5) are directly added to the `amhMesh` set. If there are conflicting hexahedra (i.e., partial overlaps) within this set by initialization, the lower quality primitives of conflict-pairs are removed until `amhMesh` is conflict-free. Faces used by two adjacent primitives are considered closed; faces used only once are collected in the `openFaces` set.

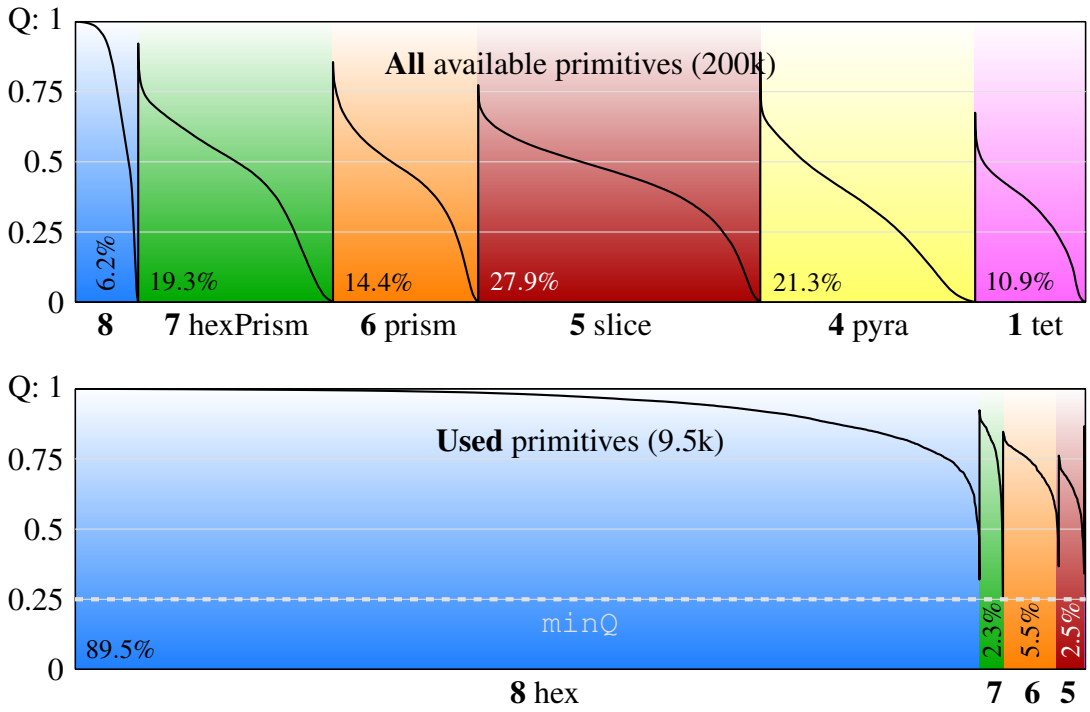


Figure 5.12: Quality histograms of primitives for the *Fandisk*, lex-sorted by `hexType` then quality (MSJ). Stats are shown before (top) and after the assembly (bottom). The final mesh only features high-quality (>`minQ`) primitives, primarily hexahedra.

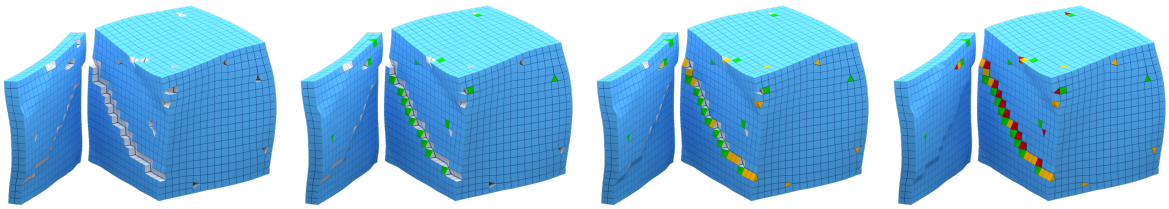


Figure 5.13: Progress (left to right) of the assembly routine on a cut open example of the *Twistcube*. This visualizes iterations of the `while` loop in Listing 5.3, lowering the `primType` each round (**8** hex > **7** hexPrism > **6** prism > **5** slice) to close open faces (white). The start `amhMesh` (left) is initialized with hex-only elements of high quality.

The algorithm's design ensures it prioritizes larger (favorably hexahedral) elements of high quality. Smaller primitives serve as a fallback solution, especially the tetrahedron, as a last resort to fill up the smallest gaps in the volume. Therefore, the criteria for primitives to be considered candidates start high and will be automatically lowered if there are no elements to be added with the current settings and reset if there was progress again. Each iteration of the assembly algorithm in Listing 5.3 consists of three phases:

• **First** (line 4-11) a preselection, where the main criteria for being considered in the next step are the `primType`, the minimum number of how many open faces a primitive should close `cLim` and a minimum quality threshold `minQ` which we set at 0.25. Suitable primitives are collected in the `newPrims` set and sorted (lexicographically) by their qualification criteria; first by their two integer keys (primitive type and the number of closeable faces), then the float quality measure: `primType > c > prim.Q`. Consequently, the first `newPrim` element of this sorted list has the maximum primitive type, closes the most open faces, and is of the highest quality. • **Second** (line 13-19), primitives are added but only if they are not in conflict (`primInConflict()`) with the existing mesh. Our conflict definition follows the relaxed interface-conformity constraints for hex-dominant meshes [YS03], prohibiting partial overlaps or inclusions, i.e., two prisms in a hexahedron. If it is safe, `newPrim` is added to the `amhMesh` set and the `openFaces` set is updated with a *symmetric difference* set operation ($\triangle$ in the pseudocode). Possible conflicts may arise for `newPrims` with each newly added primitive, while they still await their turn, queued in `newPrims`. Therefore, this check has to be performed individually and not for all elements in the preselection. • **Lastly** (line 21-26), the qualification constraints for the next addable primitives are either lowered or reset.

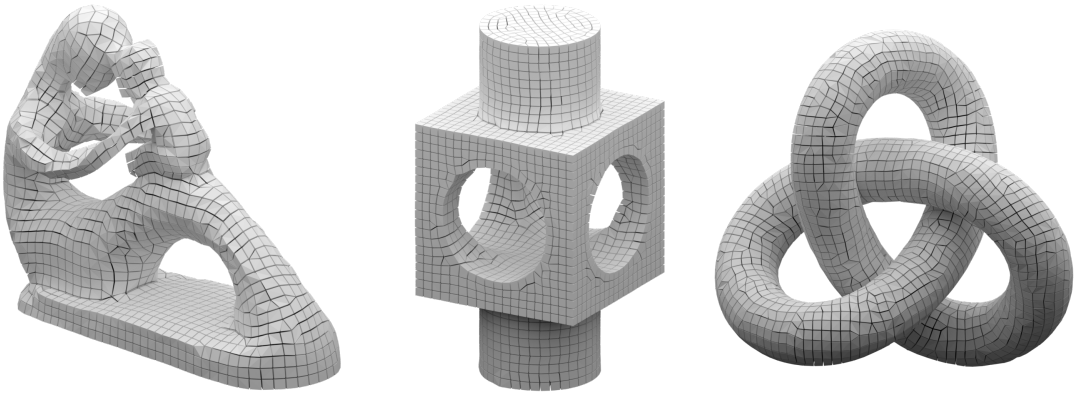
```
1  primType = 8
2  cLim = 4
3  while primType > 0:
4      newPrims = []
5      for prim in primitives:
6          prim.c = |openFaces ∩ prim|
7          if prim.t >= primType \
8          and prim.c >= cLim \
9          and prim.Q > minQ:
10             newPrims.append(prim)
11     newPrims = lexSort((newPrims, keys = [t,c,Q])
12
13     primsAdded = 0
14     for newPrim in newPrims:
15         if not primInConflict(newPrim):
16             amhMesh.append(newPrim)
17             openFaces = openFaces △ newPrim
18             primsAdded += 1
19         primitives.remove(newPrim)
20
21     if not primsAdded:
22         primType -= cLim < 4
23         cLim = max(2, cLim-1)
24     else:
25         primType = 8
26         cLim = 4
```

Listing 5.3: Assembly algorithm to construct the at-most-hexa mesh. `amhMesh` is initialized with non-conflicting high-quality hexahedra ($\geq$ `minQ`).

The lower histogram in Figure 5.12 gives the composition of the resulting at-most-hexa mesh after the assembly algorithm in Listing 5.3, most dominantly featuring hexahedral elements. However, the assembly routine is not necessarily as straightforward as Figure 5.13 suggests. The enclosing `while` loop is allowed to lower and reset the adding criteria multiple times. Further, the loop only may terminate when there was not a single tet (type **1**) left that would close up any more faces. Consequently, as the tetrahedron is the smallest possible primitive, the final mesh is completely filled up.

5.7 Experiments and Discussion

In this section, we discuss the capabilities and practicality of our proposed concept. We elaborate the assumptions to be made for our technique and compare our results to those of other procedures with visual examples as well as on numerical data.



#: 75.8% | V: 86.9% #: 87.8% | V: 93.9% #: 65.0% | V: 81.1%

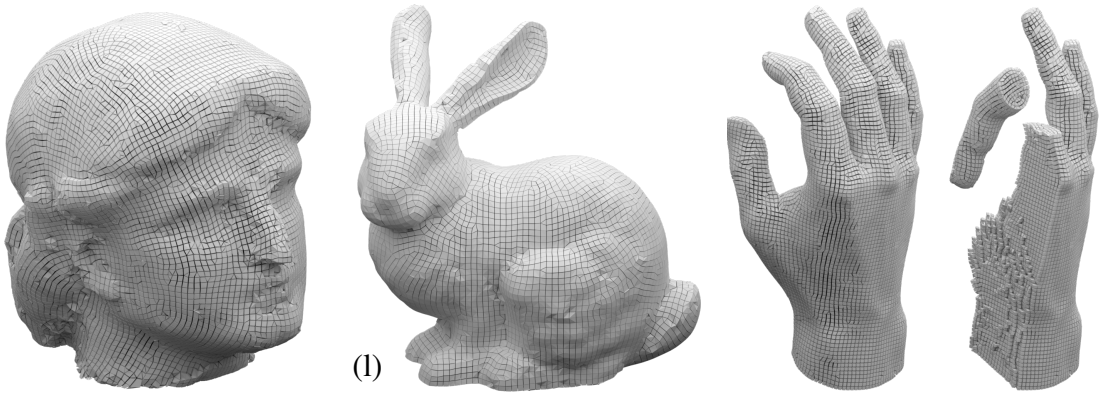
Figure 5.14: The complex geometry of higher genera is still a challenging task for some surface reconstruction techniques (Figure 4.23). Stats show the percentage of hexahedra in our results as: #: number | V: volume.

5.7.1 Requirements and Assumptions

Our method's input is very versatile: Surface meshes, hex-dominant volume meshes, or, as described in the dedicated Appendix 5.B, simple point clouds are sufficient as input.

Normal and orientation initialization is sampled from the input hull or point cloud. Surface orientation can be robustly extracted using winding numbers [BDS*18] for meshes, various strategies for point clouds [JBG19], or with our kNN graph, as described in Appendix 5.B.5. A dedicated frame field is not required as orientations implicitly align

during the relaxation. The method is robust with varying sampling density in the input, and the input boundary is not required to be closed (as exemplified with the *Bunny* and *Minerva* in Figure 5.15, which are open at the bottom). Manifolds of higher genera and complex geometries can also be meshed correctly, as shown with the examples of *Fertility*, the *Holeblock* or *Trefoil* in Figure 5.14. As it is common in volume meshing, objects like a Klein bottle or self-intersecting surfaces are out of our scope for now.



#: 77.6% | V: 88.9% (l) #: 76.7% | V: 88.2% #: 81.5% | V: 90.8%

Figure 5.15: *Minerva* [Mor09] features over 100k primitives, reconstructed from a point cloud of only 8k points. The *Bunny* [TL14] and the *Hand* were given as meshes and feature ∼48k primitives. Hexahedra stats show percentages as: #: number | V: volume.

5.7.2 Results

We tested our construction method on several 3D objects, featuring mechanical and organic shapes. For comparison purposes against competing hex-meshing approaches, our results comprise popular 3D test objects, the real 3D scan data of *Minerva* in Figure 5.15 and synthetic point cloud examples for our results of the *Cylinder* in Figure 5.22 and *Igea* in Figure 5.B.5. Table 5.1 lists numerical evaluations of our results compared to other approaches. Shown results and given numbers reflect the native outcome of our procedure without further optimization [LSVT15].

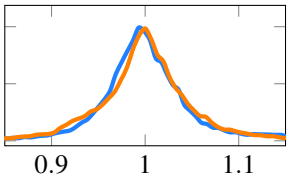
Comparisons & Quality In terms of mesh quality, one can compare our statistics in Table 5.1 to other hex-meshing approaches. Confirmed by the direct visual comparisons in Figures 5.16, 5.17 and 5.22, our results are on a par with state-of-the-art results.

As mentioned in Section 3.3, creating a valid tet-mesh [SJ08, Si15, HZG*18], or deriving a frame-field from this mesh poses a non-trivial problem [BRM*14, SVB17]. Therefore, it is difficult to evaluate a fair and meaningful comparison to related work since they

often focus on solving a very specific subtask and expect an explicitly tailored input for that, while ours generally works on simple meshes or point clouds. These drastically reduced assumptions on the input are a primary motivation in our work. State-of-the-art methods usually start from a tet-mesh and also at least require additional frame-fields [LLX*12, SRUL16, SVB17, LPP*20], compute one on their own [GJTP17], use mappings and parameterizations [GSZ11, LBK16, RSR*18, CAS*19], hand-crafted singularity graphs [LZC*18, CC19], dual-sheets [Tak19] or feature-edge selections [LPP*20]. These required inputs can be constructed in external and preliminary processes. As they are highly non-trivial, many of them are subject to their own lines of research: For example, the construction of a tet-mesh from a boundary tri-mesh [SJ08, Si15, HZG*18], a tri-mesh from a point cloud [BTS*17], a volumetric directions-field from boundary geometry [BRM*14, SVB17], or sharp-feature lines from a boundary mesh [MAR*20]. Despite the recent advancements in each of these fronts, we consider it advantageous to be able to bypass the need for these tasks.

Figure 5.12 reports quality measure histograms for the *Fandisk* model of all gathered primitives from the relaxation (top) and the used ones in the final mesh (bottom). Since the mesh extraction is based on a very relaxed state of aligned cells, the native hexahedra featured in the extracted mesh are hardly deformed at all. Further was the assembly algorithm designed to prioritize hexahedral primitives with a certain Minimum Scaled Jacobian quality. Consequently, our ASJs [PTS*08] also play in the same order of magnitude or even beat established state-of-the-art procedures which mostly produce results of ASJ$>$ 0.9. As expected, the MSJs never fall below our threshold `minQ` of 0.25.

The constrained regularized relaxation of Section 5.5.4 also pays out in quantitative terms: Proportions of hexahedral primitives in our at-most-hexa meshes ($>$ 80%) often exceed established hex-dominant procedures, although it is statistically beneficial to occupy volume with *one* large arbitrary polyhedron instead of *many* smaller at-most-hexa primitives. Our results also approach similar levels of feature alignment and homogeneous edge-flow as in the results of common mapping-based procedures [GSZ11].

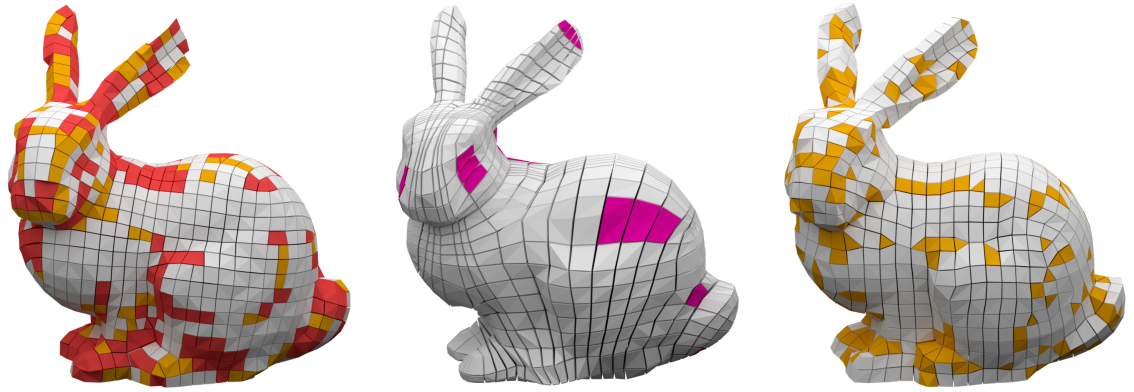


As the relaxation strives to maximize uniformity, equidistantly distributed sites generate primitives of approximately the same size. The plot on the left shows a histogram over edge lengths of the *Fandisk* model of Gao et al. compared to our result. The plot is zoomed in on the median peak, which was scaled to 1.

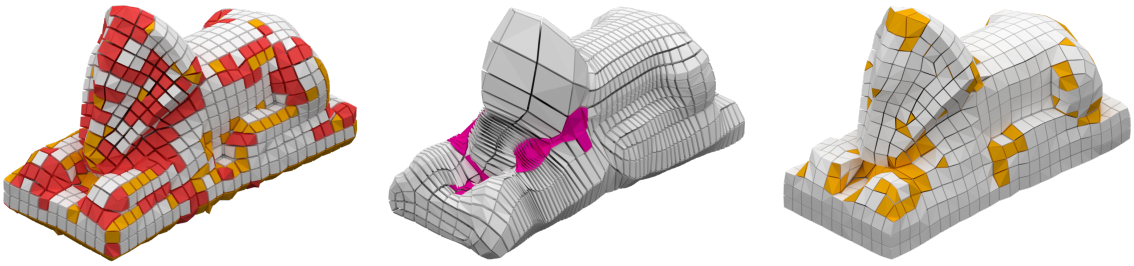
Compared to the *Jumpramp* or the *Twistcube*, organic models like *Minerva* or the *Bunny* in Figure 5.15 are obviously more challenging to be modeled with such inorganic mathematical shapes like hexahedra. Nevertheless, when comparing quantitative results of Table 5.1, the organic shapes don't fall back in comparison to CAD or mechanical parts.

		# prims	hex (%)	vol (%)	MSJ	ASJ	t
anc101	Ours	84 551	**93.30**	96.90	**0.261**	**0.983**	1590
	[LL10]	105 000	77.14	-	-	-	720
	[GSP19]*	188 886	-	-	0.094	0.865	46 366
aries	Ours	14 004	94.80	97.60	**0.255**	**0.987**	1255
	[GSP19]*	22 547	-	-	0.092	0.813	4568
bunny	Ours (l)	49 243	**76.70**	88.20	0.252	0.939	1639
	Ours (s)	3201	**71.40**	84.90	0.258	0.935	153
	[SRUL16]	-	60.57	88.61	-	**0.950**	469
	[RSR*18]	-	-	**92.20**	-	-	-
	[GJTP17]	2135	66.16	65.17	0.285	0.953	-
	[Tak19]*	2832	-	-	–0.771	0.749	-
	[GSP19]*	29 698	-	-	0.292	0.790	-
	[LPP*20]*	2172	-	-	**0.451**	0.911	112
cylinder	Ours (g)	1665	**93.40**	97.00	**0.373**	**0.973**	22
	Ours (n)	1671	85.80	93.10	0.298	0.964	16
	[SRUL16]	-	64.66	90.85	-	0.960	327
	[RSR*18]	-	-	**99.30**	-	-	-
fandisk	Ours	9523	**89.50**	**95.30**	0.303	0.973	780
	[SRUL16]	-	51.36	77.82	-	0.969	10
	[GJTP17]	7069	88.36	87.74	**0.668**	**0.986**	-
	[Tak19]*	1774	-	-	0.217	0.905	-
fertility	Ours	4997	**75.80**	**86.90**	**0.261**	0.946	645
	[SRUL16]	-	33.63	78.40	-	0.930	1121
	[GJTP17]	4769	72.04	72.56	–0.330	**0.960**	1429
hanger	Ours	12 706	91.10	95.80	0.256	**0.976**	1905
	[GSP19]*	26 918	-	-	0.155	0.828	1536
	[Tak19]*	1382	-	-	**0.333**	0.944	-
igea	Ours	27 015	76.80	**88.20**	**0.251**	0.941	1228
	[GJTP17]	12 936	**81.00**	81.00	–0.230	**0.970**	-
rod	Ours	7409	87.20	94.10	**0.355**	**0.968**	541
	[GSP19]*	26 918	-	-	0.155	0.828	1536
	[Tak19]*	600	-	-	0.221	0.763	-
sculpt	Ours	5247	77.10	89.10	0.251	**0.934**	667
	[GSP19]*	15 202	-	-	0.104	0.759	826
	[LPP*20]*	168	-	-	**0.806**	0.918	18
sphinx	Ours	2100	**81.50**	**90.90**	**0.312**	0.963	200
	[GJTP17]	2170	76.68	77.85	0.158	**0.971**	-
	[GSP19]*	45 348	-	-	0.182	0.814	-
	[LPP*20]*	3944	-	-	–0.803	0.808	672
Ours	hand	47 132	81.50	90.80	0.253	0.950	521
	holeblock	7491	87.80	93.90	0.250	0.970	175
	jumpramp	1460	97.30	98.70	0.776	0.994	9
	minerva	100 567	77.60	88.90	0.251	0.940	1235
	trefoil	8526	65.00	81.10	0.252	0.917	362
	twistcube (s)	1301	88.90	95.60	0.488	0.978	7
	twistcube (m)	5695	96.20	98.50	0.571	0.992	16
	twistcube (l)	15 841	96.20	98.40	0.316	0.990	76
	twistcube (t)	2836	21.40	41.30	0.125	0.801	-

Table 5.1: Statistical evaluation, comparing our results to numbers of hex-dominant (all-hex*) approaches. The total number of primitives is listed as well as the proportions of hexahedra and their volume. The quality metric of Scaled Jacobians (best close to 1) is given with **M**inimum and **A**verage. The *Cylinder* (Figure 5.22) is included as the native (n) and guided (g) version, the *Twistcube* (Figure 5.21) with three resolutions (s,m,l) and the tet-based example (t). The total time for shown results is included in seconds.



Gao et al. [GJTP17] Takayama [Tak19] Ours (s)

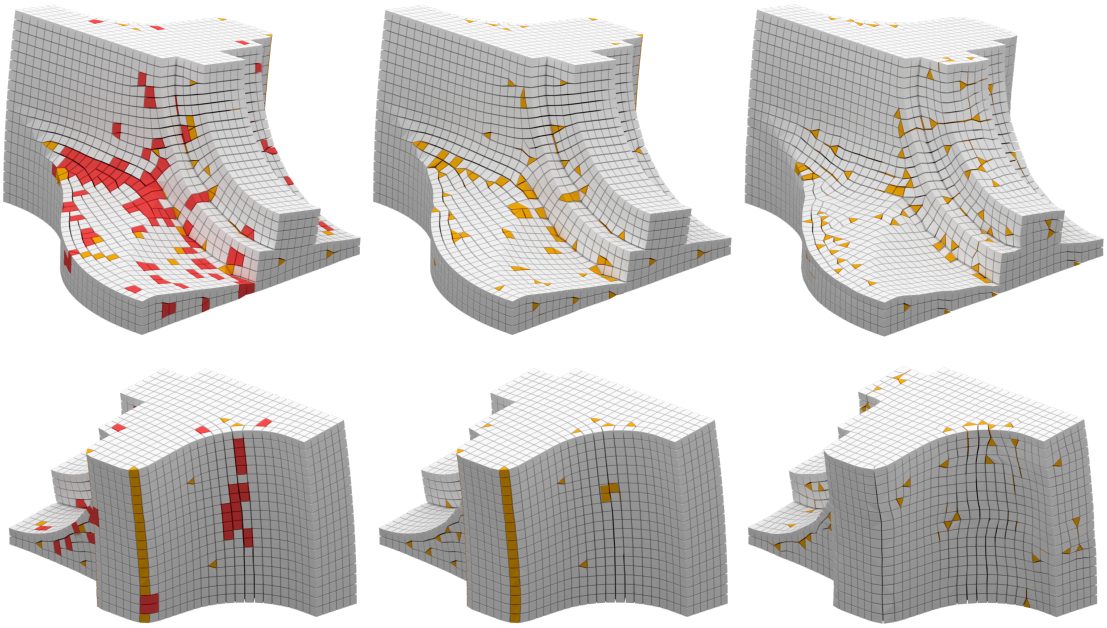


Gao et al. [GJTP17] Livesu et al. [LPP*20] Ours

Figure 5.16: Results of the *Bunny* and *Sphinx* with about the same resolution each. Colors encode general polyhedra (larger than hexa), smaller ones and inverted primitives.

Figure 5.16 shows a comparison that highlights the limitation of existing all-hex methods [Tak19, LPP*20], especially in term of reliance on complex-to-produce input. In these cases, irregular cell shapes, widely varying element sizes, and inverted elements are produced with the competing strategies. Livesu et al. attribute this failure case to a limitation of their procedure, which only performs well on suitable input frame-fields with evenly distributed singularities.

Figure 5.17 illustrates another direct comparison with a result of Gao et al. [GJTP17], highlighting all non-hexahedral elements. Whereas their mesh includes primitives with more than eight vertices and faces larger than four vertices, the maximum primitives in our mesh are pure hexahedra. The illustration further shows how our assembly routine can even improve on given hex-dominant meshes. Therefore, we skip the relaxation part to generate geometry, directly break down the input structure and reassemble an improved at-most-hexa mesh. For general hex-dominant meshes, this is also a step forward regarding conformality. Situations, common in hex-dominant meshes, where arbitrary non-planar polygonal faces separate cells, are now reduced to at maximum quad faces. Depending on the downstream application, cell volumes separated by quads may either be approached with bilinear interpolation or triangulated by a diagonal. *Sliver* elements



#: 88.36% | V: 87.74%
Gao et al. [GJTP17]

#: 90.6% | V: 96.1%
[GJTP17] + Ours

#: 89.5% | V: 95.3%
Ours

Figure 5.17: A direct comparison of results for the *Fandisk* object. Primitives with more than six faces (larger than hexahedra) are colored red and are not possible in our mesh. Primitives smaller than a hexahedron are highlighted in yellow. In the center column, we used the already hex-dominant result from the left as volume meshed input. Stats show the percentage of hexahedra as #: number | V: volume.

[SRUL16, RSR*18] would also allow for quads facing two triangles. However, while hex-dominant input is perfectly suitable for our routine, general meshed input like Delaunay tet-meshes are not. The rightmost example in Figure 5.21 shows a result based on a *TetWild* [HZG*18] input mesh. As the vertices and edges in the tet-mesh do not incorporate any flow or curvature information, the possibilities to extract nicely shaped primitives are limited. Nevertheless, by lowering the quality threshold `minQ` to 0.1, the assembly routine at least managed to recover 21.4% hexahedral elements.

Hull Deviation The hull-cell shrinkage illustrated on the right in Figure 5.B.4 allows for cells to approach the explicit mesh hull or implicit point cloud hull very closely. We measure this reconstruction error using the symmetric Hausdorff Distance, comparing our results' quad-dominant hull to ground truth meshes using 100k random samples. With the maximum diagonal of a rotated minimum bounding box around the object as reference, the worst-case surface deviations rarely exceed the 2% mark. These worst-case deviations usually arise from vertex merge operations on sharp corners as seen on

the bottom row of our *Fandisk* result in Figure 5.17 or the central corner of the *Jumpramp* in Figure 5.19. However, reconstruction errors like HD and RMSE can be easily minimized further in a post processing step, e.g., as shown in Appendix 7.B.

Comparison to L_p based Meshing Centroidal Voronoi Tessellations (CVT) under the L_p norm have been extensively studied by Lévy and Liu [LL10]. While their focus clearly lies on the formulation and derivation of L_p-CVTs, they also propose their use in quad- and hex-dominant meshing applications. Nevertheless, we can draw some clear distinctions between their approach and ours: Whereas our approach aims to materialize the hexahedral cells of the diagram itself, Lévy and Liu utilize the diagram's bi-graph for the geometry extraction. Similar to Sokolov et al. [SRUL16], the graph-matching approach of Meshkat and Talmor [MT00] is employed to assemble hex-like cells from the diagram's Delaunay tetrahedralization. As their mesh vertices are actually former cell centroids, they have to compensate for the resulting shrinkage on the most outer hull layer. This is not required in our approach where a cell itself corresponds to a mesh primitive, thus there is no gap between generated geometry and target hull. The concept of Lévy and Liu also does not allow for control over the mesh regularity, as their cell's positional alignment is solely based on the relaxation. In contrast to that, our internal graph structures can enforce certain favorable alignments during the relaxation, as introduced in Section 5.5.4 with the N_6 graph. Furthermore, our N_k graph provides the possibility to propagate and interpolate orientations throughout the diagram, thus allows for cells aligned to the input hull but also to each other. In the diagram of Lévy and Liu, individual cell orientations are queried from anisotropy matrices associated with outer hull faces, based on single nearest-neighbor connections. As a result, adjacent cells are not necessarily similarly oriented but solely depend on their closest connection to the outer hull. This effect is prominently visible in Figure 5.18 on the cut-open *anc101* model: In Lévy and Liu's L_p-CVT result, the rounded cap of the pin-cavity dominates large portions of the internal alignment. In our result, the small cavity only has a minor influence as the cells are primarily aligned with the dominant outer shape of the model.

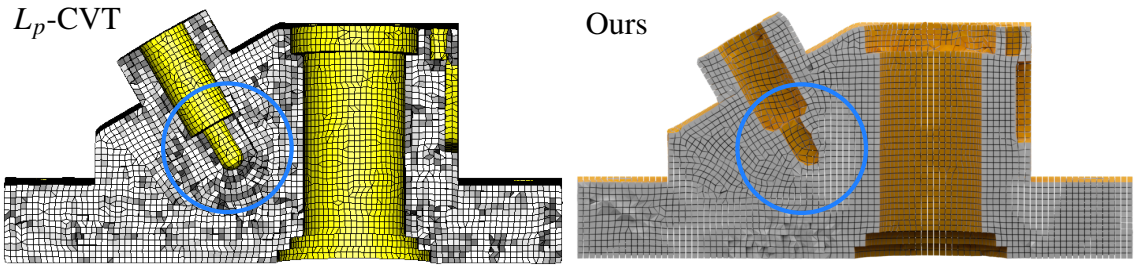


Figure 5.18: In the L_p-CVT mesh [LL10] the rounded cavity cap dominates the interior alignment; our cells are more aligned to the outer shape. We adopted the color scheme of Lévy and Liu with yellow hull faces and white for the exposed cut-open interior.

Regularity and Alignment The results presented in Figure 5.19 are prime examples of the synergy of hull samples and cell orientations during the relaxation with no frame-field given. On planar surfaces, as on the *Jumpramp*, only surface normals are determined robustly, whereas primary curvature directions are just random guesses. Since there was no consistent curvature field, the cell alignment of the flanks dominates during the relaxation and the orthogonally aligned orientations follow along, so the resulting arrangement is as regular as possible. Nevertheless, hex-like cells also intuitively align to more distinctively shaped geometry like the *Fandisk* in Figure 5.17, featuring creases, flat, angled, curved, and narrowing regions.

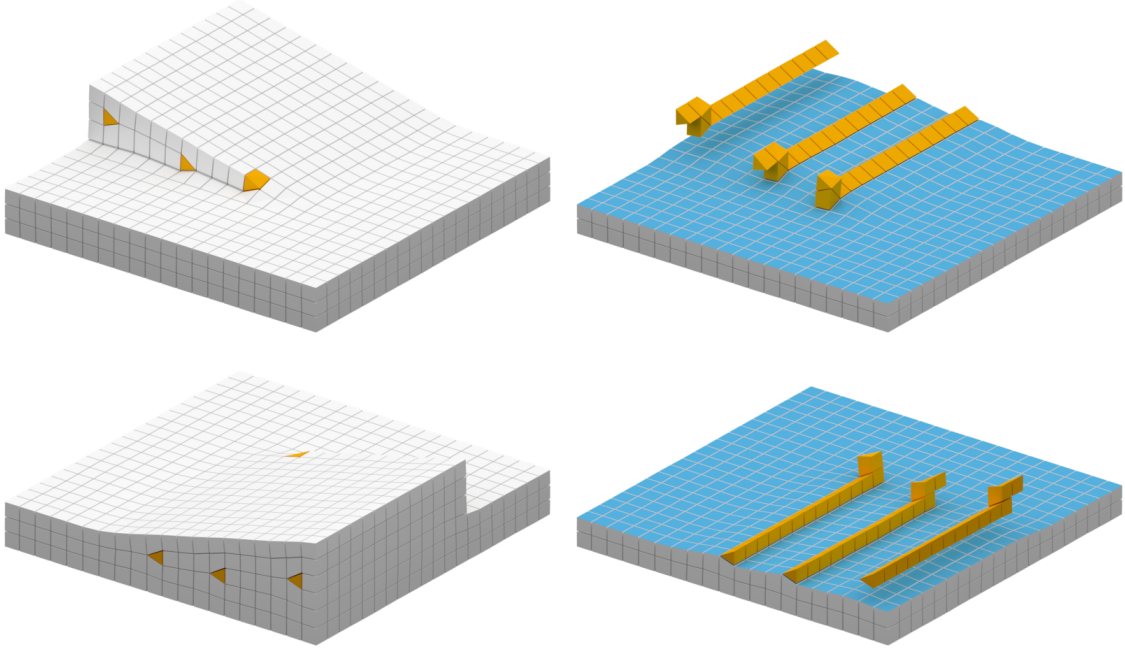


Figure 5.19: The *Jumpramp* is challenging for hex-meshing approaches based on regular parameterizations or integer mappings. Nevertheless, our relaxation is flexible enough to cope with the angled and narrowing geometry. Inner faces are shaded blue, non-hex primitives (all triangular prisms) are highlighted in yellow. Stats are #: 97.3% | V: 98.7%

The choice for initial site positions on an axis-aligned regular grid is well suited for objects that also feature axis-aligned parts like the *Jumpramp*, the *Holeblock* or the CAD models in Figure 5.23. Nevertheless, the relaxation is very flexible and able to faithfully approximate organic shapes without explicitly axis-aligned parts. But to be fair, even organic shapes like *Minerva* or *Fertility* are usually not given with an arbitrary rotation but are also often oriented for at least axis-aligned symmetry.

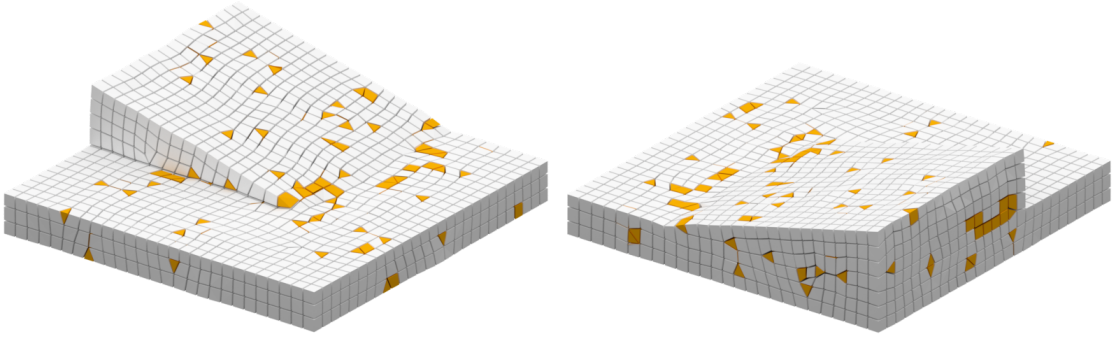


Figure 5.20: A forced-failure case where the initialization was rotated around 45° against the *Jumpramp*. Although the alignment is not optimal, the reconstruction still consists of 80.2% hexahedral primitives consuming 89.9% of the object's volume.

We've also experimented with forced-failure cases, for example leading to the result shown in Figure 5.20. It is based on a worst-case initialization, where the initial cell grid was rotated around 45° against the *Jumpramp*'s orientation. Although such cases can easily be avoided by automatically aligning the grid to dominant regions of the object, the relaxation still managed to overcome this misalignment gracefully and reconstructed all surface features accurately. While the quality of the edge flow is not as optimal as with the axis-aligned initialization (Figure 5.19), a portion of over 80% hexahedral primitives is still quite acceptable for a worst-case.

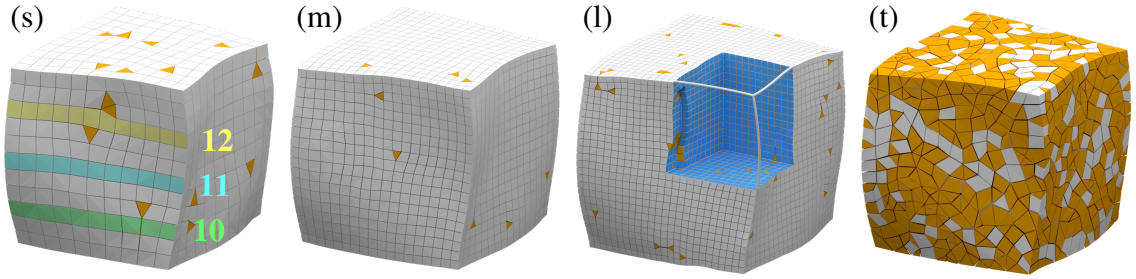
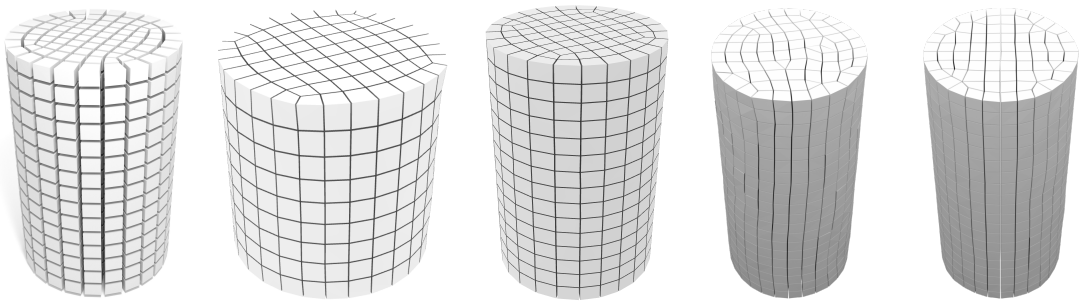


Figure 5.21: The *Twistcube* [JTPSH15] in three different resolutions (s), (m), and (l), the latter one with a cutout. The rightmost cube (t) is based on a Delaunay tetrahedralization [HZG*18] instead of our relaxed input.

The *Twistcube* in Figure 5.21 illustrates how the relaxation also maximizes isotropy on objects with non-axis-aligned surfaces: The curved point cloud hull acts as inside/outside separation on the initially regular grid cells. This rasterized initialization is resolved in the relaxation with a varying number of homogeneously shaped primitives instead of squashed and stretched ones.



[LZC*18] [CAS*19] [CC19] Ours (n) Ours (g)

Figure 5.22: Results of other recent work using tet-meshes, polycube mappings, and singularity graphs as input compared to ours based on hull points only. This also compares a native (n) outcome of our procedure with an experimentally guided (g) version.

The relaxation process also promotes homogeneity throughout the whole object. For example, the *Cylinder* in Figure 5.22 features equally sized primitives in its center as well as on the outer hull. Some hex-meshing algorithms tend to mimic proportions of the hull in deeper layers which leads to an unnecessarily high resolution towards the core of an object [LBK16, SRUL16, SVB17]. Cells are not explicitly bound to stay close to their initial position or neighborhood. Although objects like the *Cylinder* or the *Holeblock* (Figure 5.14) can't provide much vertical support, cells do not twist out of control and maintain a quite regular vertical alignment.

Guidance Due to the relaxation, the mesh's flow naturally aligns to object curvature, based on surface features alone. Nevertheless, it is quite easy to supplement this process with guiding structures inside the object to control the internal flow-field and orientations. For the *Cylinder* (g) in Figure 5.22 we extended $\mathbf{S}_{pc}$ with additional samples on two orthogonal planes, intersecting on the rotational axis of the object, similar to dual-sheet meshing [Tak19]. This pushes the anticipated regularity for simple geometric shapes even further by guiding the relaxation to obey symmetry or similar characteristics.

Performance Due to the decentralized graph structure (Section 5.5.2), the relaxation part of our pipeline can easily be parallelized. Timings were measured with an implementation in *CUDA*, run on a *GeForce GTX 1080Ti* graphics card. The performance of the relaxation heavily depends on the selected parameters, e.g., grid resolution and on the dimensions of the input structure. Included results were created using 150 relaxation iterations. The *Minerva* object, one of the larger reconstructions listed in Table 5.1, features about 225k Voronoi cells (including the space outside the object). One complete relaxation iteration includes: Computing Voronoi cells and recentering their sites, updating the N_6 and N_{26} neighborhood graphs and n-hop distances, updating and interpolating

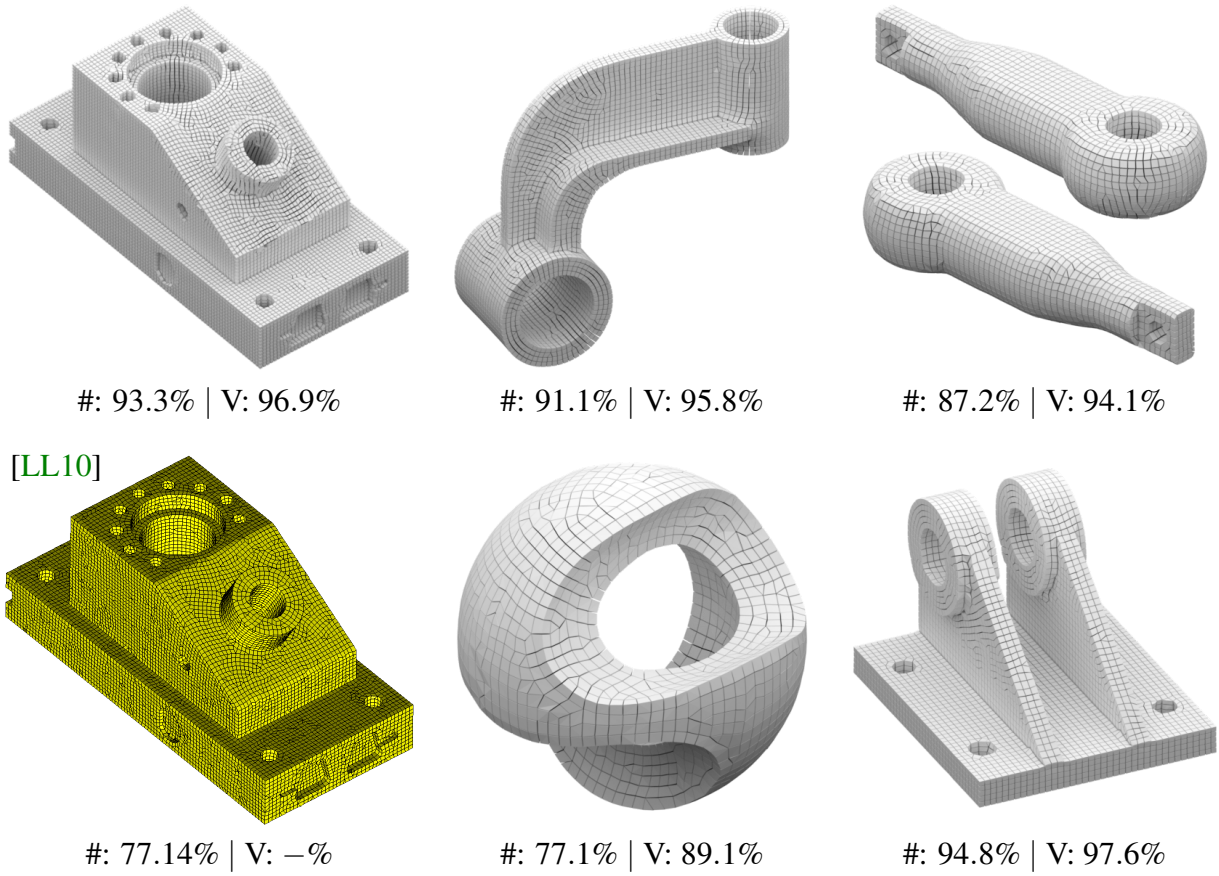


Figure 5.23: More results on CAD models. Numerical stats show the percentage of hexahedra as: #: number | V: volume.

separate individual positions and orientations, and combining them with the progress-dependent weighing function. With a neighborhood size of $k = 26$, one iteration for these 225k elements is done in 2.43s. For reconstructions of with a coarser resolution, e.g., the *Jumpramp* model, our implementation reaches about 18 iterations per second. The vertex merging steps as well as the graph-matching for geometry and topology extraction are also easily parallelizable and finish within a few seconds in a multithreaded Python implementation. The assembly iteration in Listing 5.3 is the only serial operation in our pipeline. Nevertheless, the straight ordering of input data (Figure 5.12) allows for an efficient implementation and also terminates within minutes.

An extensive comparison of *number-of-primitives vs. time* stats with a recent publication might be rather misleading due to the following reasons: Our relaxation currently (see Section 5.7.3) operates on all cells within the object's bounding box volume. Therefore, filigree objects (Figure 5.14) would do rather poorly compared to massive blocky objects (Figure 5.21). On the other hand, timings of other approaches with similar primitive

count usually also heavily depend on the task to be solved and given input conditions. Nevertheless, Table 5.1 lists measured timings for the included results.

Figure 5.12 lists about 200k gathered primitives of which only 9.5k were actually used in the final mesh. This brute-force approach seems kind of wasteful but still outperformed more advanced alternatives. An interleaved routine, alternating assembly steps with primitive collection queries *only where needed* caused overall too much overhead. The simplest method proved to be the fastest: Gather primitives in parallel and run the assembly on sorted data.

The approach for hex-dominant meshes proposed by Sokolov et al. [SRUL16] is also based on graph-matching, but instead of faces, their algorithm uses Delaunay tetrahedralizations to assemble primitives. However, with tetrahedra, the graph-search becomes much more complex as there exist multiple constellations of a varying number of tetrahedra to formulate one larger primitive. As shown in Section 3.3, there are 10 possible ways to construct a hexahedron using 5, 6, or 7 tetrahedra. In our graph-search, there is usually only 1, or at maximum 2, valid path option(s) per hex-type, using tri- and quad-faces as shown in Figure 5.A.1.

5.7.3 Outlook

Additional Guidance As experimentally introduced in Figure 5.22, the relaxation can be supplied with guidance from additional geometry. Possible scenarios in future research could be to explore the utilization of common and more explicit guiding structures. Dual-sheets, frame-fields, or singularity graphs are currently not required in our method but could improve result quality.

Non-uniform Cells Cells of non-uniform size and/or shape bear geometric challenges which have yet to be explored. However, solving them may pay out in the form of improved hex-mesh quality or adaptive meshing options. Formulating non-hex cells in the relaxation or cells scaled anti-proportional to the n-hop distance would allow for a more efficient and detail-focused evaluation.

Exclude far away Cells To further improve the relaxation performance, it could be beneficial to investigate mechanisms to exclude far-away outside cells at an early stage of the relaxation. A combined criterion with an n-hop distance > 1 would already exclude many cells from further computation, probably without too much impact on the closest relevant cells on the hull's inside.

5.8 Conclusion

In this work, we present a new and innovative way to construct at-most-hexa mesh structures. Further, the proposed procedure succeeded in bridging the gap between surface point clouds and volume hex-mesh reconstruction. The quantity and quality of hexahedral elements in our meshed results are improved compared to established state-of-the-art algorithms. We introduce at-most-hexa meshes, a novel class of hex-dominant meshes, where non-hex elements are linear combinations of hexahedra. This allows for trivial interpolation of scalar or vector signals within the volume and greatly simplifies the internal representation due to the suitability for indexed meshes. Contrasting common requirements for such tasks, arbitrary meshes or point clouds of various sizes and densities are sufficient as input. The resolution for the resulting mesh can be selected independently. The proposed approach consists of two parts, starting with a Lloyd relaxation that eventually reintroduces constrained regular structures. In contrast to previous L_p relaxation methods using Delaunay tetrahedralizations, our geometry extraction is based on the actual materialization of the relaxed Voronoi cells. At-most-hexa primitives are extracted with specialized state-machine programs, and the final mesh is then assembled from a quality-sorted priority queue. The interaction between fixed hull- or point cloud samples and relaxed cells generates a homogeneous orthogonal vector field for feature-aligned mesh structures. Therefore, the approach is suitable to reconstruct organic and curved objects with well-aligned hexahedral primitives as well as CAD-like models with flat surfaces, sharp edges, or angled geometry.

Chapter 5

Appendix

5.A At-Most-Hexa Primitives

The basic primitive of our at-most-hexa mesh is a simple hexahedron with 8 vertices and 6 quad faces. All other allowed primitives are listed in Figure 5.A.1 and can be derived from this base case by collapsing up to four edges.

State-Machine Programs The search for primitives in the set of quadrangular and triangular faces can be performed very efficiently, using specifically tailored state-machine algorithms for each type of primitive. Starting from the initialization state with only one face, adding a face is the only supported operation to transition from one state to the next. If a transition generates a valid state, the search path continues or is discarded otherwise. The illustration in Figure 5.A.1 shows the progress of the individual state-machines with color-coded faces and edges. The *primary face* is the starting position, providing the first set of *open edges*. *Secondary faces* may only be added on *open edges* of the *primary face*. *Tertiary faces* are determined indirectly by a specific set of open edges, e.g., in the penultimate state of the hexahedron, the last four open edges unambiguously specify the last quad face. Since only faces in a direct adjacent neighborhood (fix size) are considered for a state transition, the search's complexity is in $\mathcal{O}(n)$.

Interexchange Format At-most-hexa primitives can be formulated as linear combinations of the 8 vertices of a hexahedron. We use this property to store our results as simple indexed *mesh* files, commonly used for hex-meshes. Therefore, corner vertex indices are duplicated as listed in Figure 5.2 with 8-vertex-index sets for the different primitives, respectively. Out in the wild, this format could easily be read by dedicated hex-meshing applications. However, some internal routines may struggle with collapsed edges or faces: *Graphite* [LA] is able to load our meshes but may not be able to robustly differentiate between interior and hull faces. Nevertheless, the *HexaLab* [BTP*19], an online visualizer/analyzer that is designed for pure-hexa meshes, can be used to examine our results, which are provided in the supplemental material.

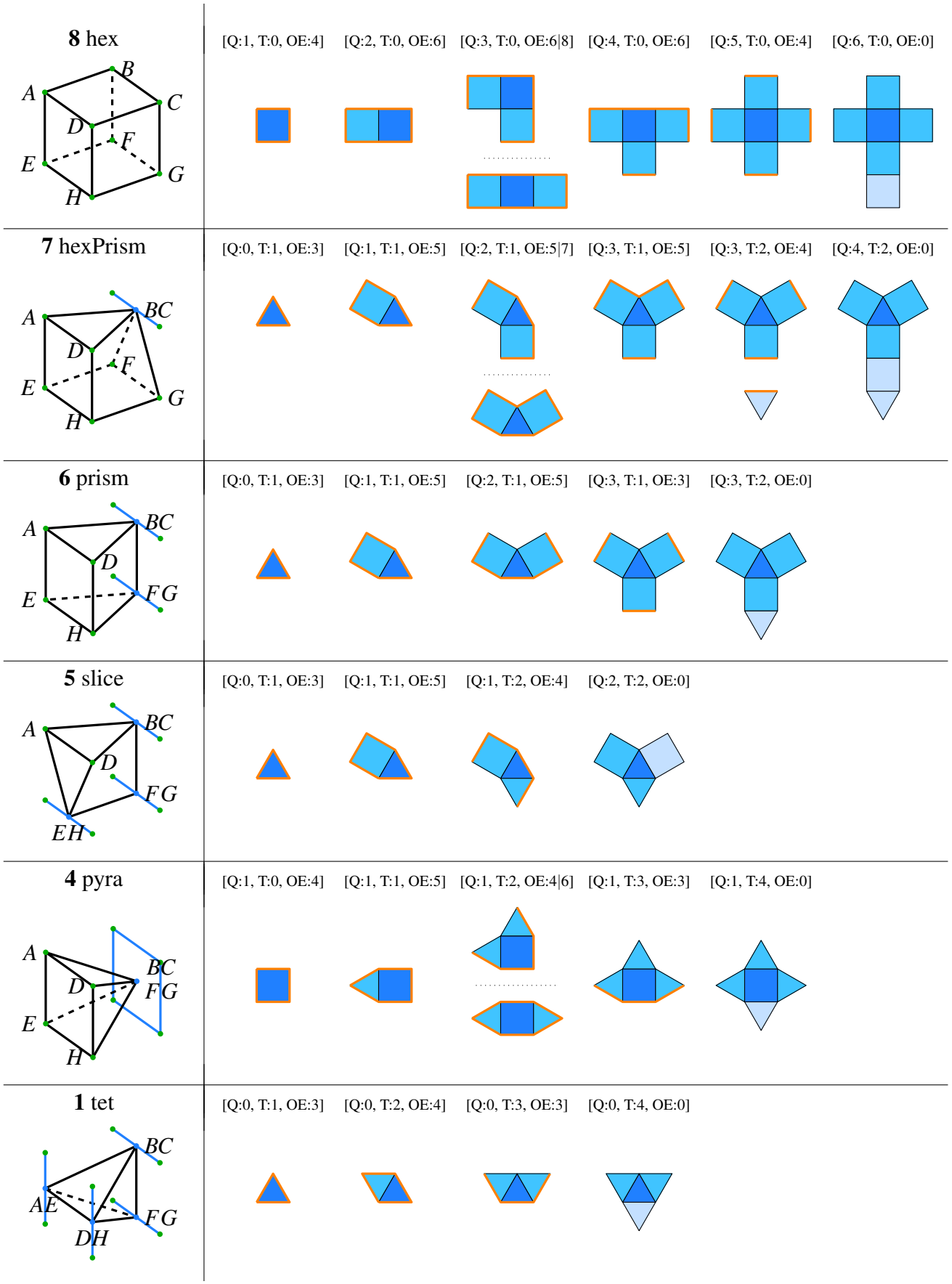


Figure 5.A.1: State-machine programs for the different (used) at-most-hexa primitives, with an initialization state of only one face on the left and completed primitives on the right. Valid states are encoded in triples: Number of **Q**uads / **T**ris and the number of **O**pen **E**dges.
Colors highlight open edges, primary, secondary and tertiary faces.

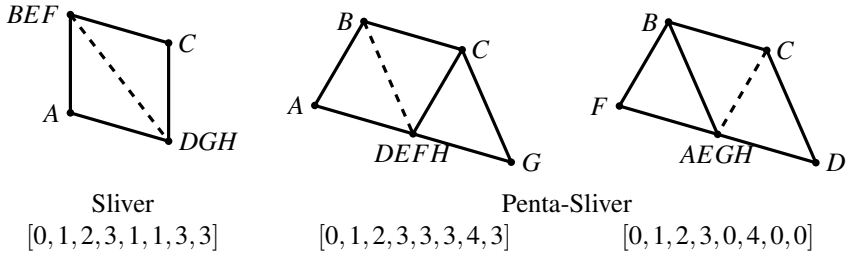


Figure 5.A.2: At-most-hexa conform versions of quad- and penta-sliver primitives.

Conformal Slivers As introduced in Section 2.3.4, flat sliver elements with no volume can be inserted between primitives of a hex-dominant mesh. This provides fitting opponents to internal faces, which would otherwise falsely be considered as *open*, thus part of the outer hull. However, the concepts illustrated in Figure 2.3 and 2.4, squashing a tetrahedron and triangular prism, have to be adapted to be conformal with our at-most-hexa mesh. While there exist equivalent variations of this problem, we employ the formulations shown in Figure 5.A.2 for a quad-sliver and two (mirrored) penta-slivers. These formulations allow being encoded as at-most-hexa primitives in such a way that hex-mesh applications like *HexaLab* [BTP*19] do not consider the collapsed faces as open.

5.B Point Cloud Input

In addition to supporting closed manifold input, we also propose a simple extension to our method, so that it solely requires point clouds as input. This feature is elaborated analogously to Section 5.5.

5.B.1 Motivation

While surface reconstruction techniques can easily start from point cloud samples directly, established hex-meshing algorithms require complex input like parameterizations, mappings, other volume or surface meshes. The goal of any signal processing pipeline should be to avoid the concatenation of lossy processing steps and the accumulation of errors. Even for well-researched procedures, it is always important to study alternative ways – if only to prove a point. So if we can go from point samples to a surface mesh and from a surface mesh to a volume mesh - why not skip this surface precomputation step and generate the volume mesh directly from point cloud samples? Nevertheless, a surface mesh is indirectly generated anyway as the outer hull of the volume mesh. Therefore, we propose an approach to bridge this gap and produce high-quality hex-dominant meshes and watertight surfaces solely from point cloud surface samples.

5.B.2 Site Population

The whole set of sites in the Voronoi diagram consists of two disjunct subsets $\mathbf{S} = \mathbf{S}_{pc} \cup \mathbf{S}_v$, where $\mathbf{S}_v$ is initialized as before. Additionally, to represent the surface, each 3D point cloud sample creates one additional site, forming the second set of sites $\mathbf{S}_{pc}$. Sites in $\mathbf{S}_{pc}$ have a fixed location – the one of the point cloud sample – and a fixed normal derived from the local neighborhood (Section 2.2.2). Both properties do not change during the optimization. The sites $\mathbf{S}_{pc}$ serve two purposes: They separate inside from outside cells, making sure that the created boundary is precisely aligned to the implicit hull. Further, they enforce that the orientation of all other sites $\mathbf{S}_v$ will adapt to the surface features.

5.B.3 Point Cloud Hull

In the Voronoi diagram, both sites $\mathbf{S}_v$ and $\mathbf{S}_{pc}$ are considered to spawn cells. Whereas $\mathbf{S}_v$ cells are uniformly shaped with the Chebyshev metric following the sites' orientation and therefore will approximate cuboids; for $\mathbf{S}_{pc}$ cells, an anisotropic transformed norm is applied: These cells are compressed along the direction of their normal and stretched along the direction of their tangent and bitangent as illustrated in Figure 5.B.3 (with factors 0.5 and 1.5). Even sparsely sampled point clouds provide a sufficiently dense hull to separate sites floating around the in- and outside of the object [Hau01]. Throughout the relaxation, the height of the $\mathbf{S}_{pc}$ cells is slowly reduced to 0 to ensure that the hull is not covering any volume (Figure 5.B.4, right). Furthermore, sites $\mathbf{S}_{pc}$ are not allowed to move during the relaxation. This way sites $\mathbf{S}_v$ can move up to the outer hull from the in- and outside of the object but will not pass the hull. Sites starting somewhere on this hull will be pushed either in or out of the object, within only a few iterations.

5.B.4 *k*NN-Graph

There are actually no modifications required for the *k*NN-graphs to support the point cloud input. The N_{26} is allowed to include neighbors from $\mathbf{S}$, therefore natively interconnecting $\mathbf{S}_{pc}$ and $\mathbf{S}_v$ sites. As before, N_6 only operates on $\mathbf{S}_v$.

5.B.5 Inter- and Extrapolation

Analogously to Section 5.5.3, all sites are initialized with orientations that are sampled and then propagated from the hull. However, in this case, based on the point cloud as described in Section 2.2.2. In/Out labels are propagated after the relaxation.

Orientation Initialization Instead of sampling the hull individually, nodes of level $d_i = 1$ query their orientation from the closest point cloud neighbors ($d_i = 0$), which are naturally included in their k nearest neighbor list. A $\mathbf{S}_v$ may query multiple $\mathbf{S}_{pc}$ nodes and a $\mathbf{S}_{pc}$ node may be shared by multiple $\mathbf{S}_v$ nodes as illustrated in Figure 5.B.3 (left).

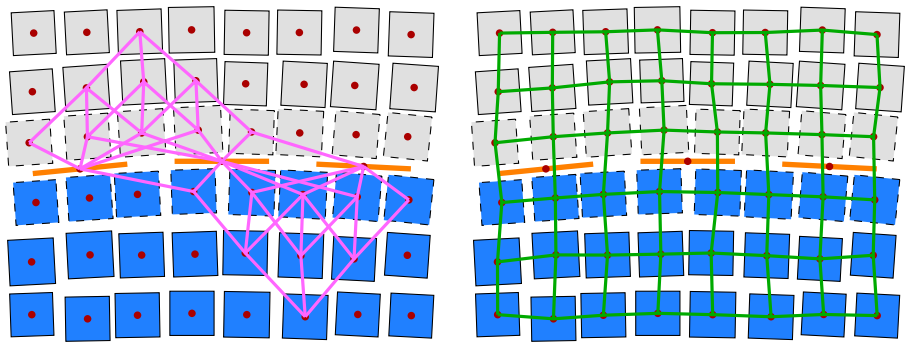


Figure 5.B.3: Nodes with $d_i = 1$ (dashed) determine their in/out label from the dot product of direction $\vec{v}_i$ and normal $\vec{n}_i$. Nodes of $d_i > 1$ derive their label from neighboring nodes closer to the surface as illustrated with the graph in magenta. The right side shows the N_6 graph (green), transcending the outer hull, so that adjacent cells on the in- and outside are able to align nevertheless.

Inside/Outside Determining in/out labels is no longer as trivial as with a hull given. Although algorithms specialize in winding numbers for point clouds, we can simply query the information at hand, using the neighborhood graph and basic geometry. First, this labeling operation is simplified by focusing only on nodes of level $d_i = 1$, which are close to the hull. All other nodes with $1 < d_i$ easily derive their state from already labeled nodes in their direct neighborhood. In/out labels for nodes with $d_i = 1$ depend on their normals pointing *towards* or *away* from hull nodes as shown in Figure 5.B.3. Vector $\vec{v}_i$ in Equation (5.B.1) is the normalized average direction from node s_i to its direct neighbors $s_j \in \mathbf{S}_{pc}$ with $j \in N_{26}(i)$. As formulated in Equation (5.B.2), the in/out label l_i derives from the sign of the dot product between direction vector $\vec{v}_i$ and normal $\vec{n}_i$.

$$\vec{v}_i = \frac{1}{|N_{26}(i) \cap \mathbf{S}_{pc}|} \sum_{j \in N_{26}(i) \cap \mathbf{S}_{pc}} \frac{s_j - s_i}{||s_j - s_i||_2} \tag{5.B.1}$$

$$l_i = \begin{cases} \text{in} & \text{if } \vec{v}_i \cdot \vec{n}_i > 0 \\ \text{out} & \text{else} \end{cases} \tag{5.B.2}$$

Near sharp edges, this procedure may cause false-positives. The scenario illustrated in Figure 5.B.4 (left) shows a case where the majority of direct point cloud neighbors is *just around the corner*. Therefore, the derived normal $\vec{n}_i$ and averaged direction $\vec{v}_i$ point into the same direction. This is a perfectly valid behavior, but one can also easily fix such outliers by querying the labels of their N_6 neighborhood as illustrated in Figure 5.B.4 (center): If a node is labeled as *inside* and has less than three direct neighbors, which are also marked as inside, its label is switched to *outside*.

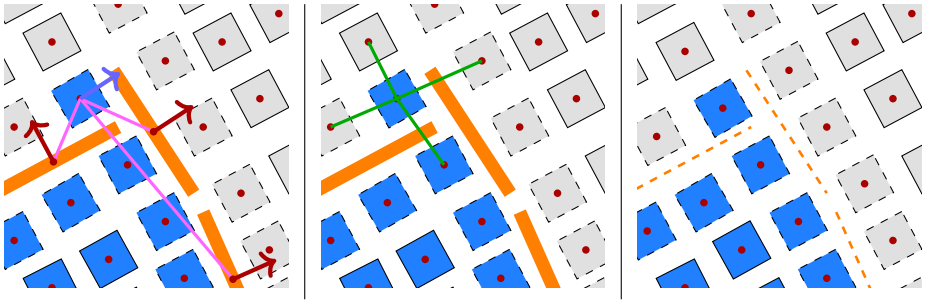


Figure 5.B.4: Left: A node identifies as **in** because its primary direction is in accord with the majority of its hull neighbors. Center: A majority vote of the N_6 neighborhood easily fixes false-positives. Right: $\mathbf{S}_{pc}$ cells converge to 0 height at the end of the relaxation, so the hull no longer covers any volume.

Compared to winding numbers [BDS*18], our approach trivially blends in with the rest of the graph operations and does not add any further computation steps. Since we limit our labeling to points that are closest to the oriented hull and propagate results for farther points, our results are practically error-free.

Bring the Noise Due to our focus on simple and minimal input, we never used more than 10k randomly sampled points for included results. The quality of real 3D scans is usually higher than what we actually aimed for: Modern mid-range 3D scanners are capable of producing high-density results with millions of points arranged in nice regular patterns. Still, robustness to bad input is always important for reconstruction tasks. To explore the limits of our procedure, we artificially perturbed input samples with noise vectors (sphere samples with limited random magnitude). Figure 5.B.5 illustrates results, using different magnitudes of noise which are given in percent of the length of the object's bounding-box diagonal. Normals and orientations were computed *from* the noisy point clouds to simulate really unreliable data. Furthermore, we did not apply any topology reconstruction or remeshing steps to these results. Even with a certain amount of noise, our method is able to concisely reconstruct the object with only minor flaws on the hull. Obviously, enough noise can also provoke failures, as shown on the right in Figure 5.B.5. With noise magnitudes larger than the cell size, the hull becomes very vague: The in/out labels of the outermost layer can no longer be determined with certainty, which is why there are loose hexahedra floating around. However, already from the second layer on inwards, the mesh is fine and follows a smooth orientation field.

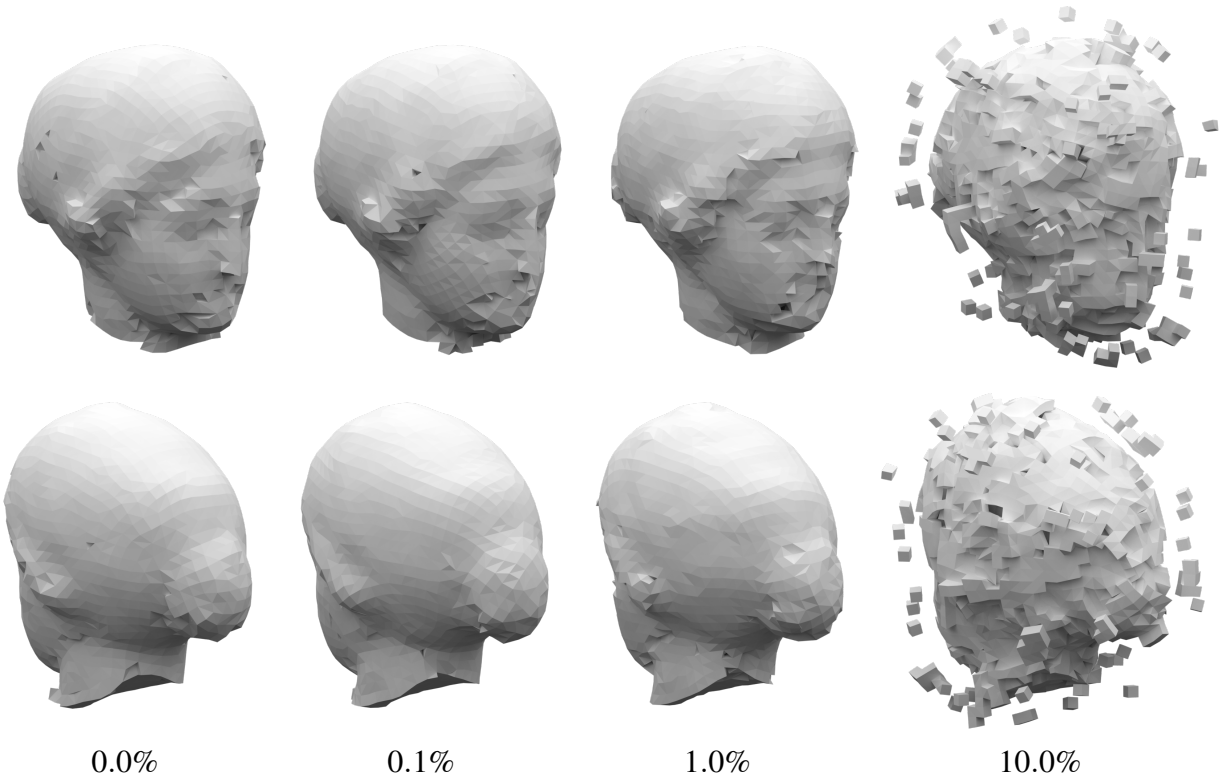


Figure 5.B.5: Reconstructions of the *Igea* artifact from 5k points, perturbed with random noise vectors. Noise magnitude levels are given in percent of the length of the bounding-box's diagonal. Raw output of the relaxation and merging, no at-most-hexa topology.

5.B.6 Relaxation

The Lloyd relaxation process with point cloud input is equivalent to the one introduced in Section 5.5.3 with one exception: As there is no hull separating interior and exterior cells, the sites $\mathbf{S}_{pc}$ are not altered during the relaxation. They remain on their initial position during the whole process and also maintain their normal direction. However, tangent and bitangent are free to be updated according to the frame-field that is generated during the relaxation. By keeping their normals fixed, the hull barrier is guaranteed to stay intact, but the cells can align with dominant surface features.

Limitations Voronoi cells relax under the assumption to approximate a cubical shape with uniform edge lengths. Cells that are enclosed within geometry, thinner than the average relaxed edge length, are in a very unrelaxed state and might be squeezed outside the object. Nevertheless, with point cloud samples placed sufficiently dense, it becomes very unlikely for cells to be able to pass the outer hull and even thin geometry can be represented, e.g., with only one layer of hexahedral cells.

Chapter 6

Tetrahedra of Varying Density and their Applications

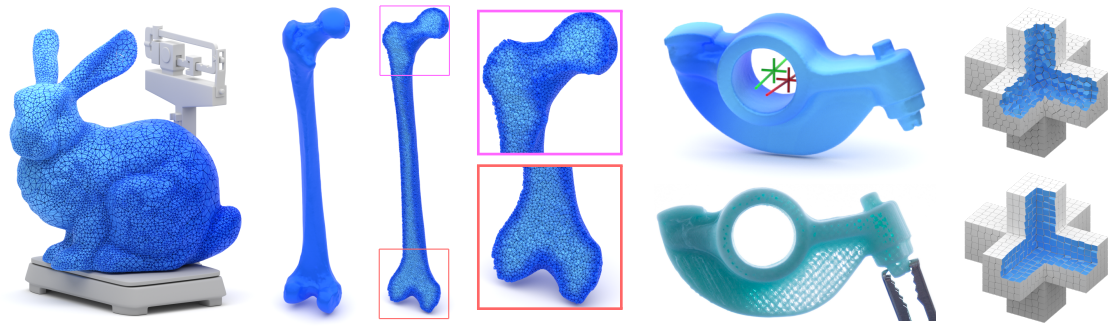


Figure 6.1: Exact mass properties for objects under varying density enable a number of applications: approximating non-linear density, optimizing density fields for specified mass attributes and the analytic evaluation of energy terms in $L_{2|\infty}$ Lloyd relaxations.

6.1 Abstract

We propose concepts to utilize basic mathematical principles for computing the exact mass properties of objects with varying densities. For objects given as 3D triangle meshes, the method is analytically accurate and, at the same time, faster than any established approximation method. Our concept is based on tetrahedra as underlying primitives, which allows for the object's actual mesh surface to be incorporated in the computation. The density within a tetrahedron is allowed to vary linearly, i.e., arbitrary density fields can be approximated by specifying the density at all vertices of a tetrahedral mesh. Involved integrals are formulated in closed-form and can be evaluated by simple, easily parallelized, vector-matrix multiplications. The ability to compute exact masses and centroids for objects of varying density enables novel or more precise solutions to several interesting problems: besides the accurate analysis of objects under given density fields, this includes the synthesis of parameterized density functions for the make-it-stand challenge or manufacturing of objects with controlled rotational inertia. In addition, based on

the tetrahedralization of Voronoi cells, we introduce a precise method to solve $L_{2|\infty}$ Lloyd relaxations by exact integration of the Chebyshev norm. In the context of additive manufacturing research, objects of varying density are a prominent topic. However, current state-of-the-art algorithms are still based on voxelizations, which produce rather crude approximations of masses and mass centers of 3D objects. Many existing frameworks will benefit by replacing approximations with fast and exact calculations.

6.2 Introduction

Whereas the other proposed techniques in this thesis are focused on the reconstruction of point clouds and mesh generation in general, this chapter explores novel applications methods of a particular volumetric mesh class, i.e., tetrahedral meshes, especially with respect to the remaining challenges of Chapter 5 on solving L_∞ Lloyd relaxations. Mass properties of bounded mesh objects can be computed accurately using the divergence theorem as long as the object is of constant density. However, when it comes to estimating masses or computing gravitational centers for varying-density objects, discretized approximations are used almost everywhere. Often their accuracy is not even questioned. Surprisingly, it is neither a very complex problem nor bound to numerical trade-offs between computation cost and accuracy. As used in finite element methods (FEM) [RR96, RNV07], our techniques are based on tetrahedra as the underlying geometry primitive. Density fields are accurately represented by specifying the density at the four vertices. The analytically derived closed-form integrals for mass properties are expressed with simple matrix-vector multiplications. In Section 6.3 we derive the concept, demonstrate its versatile applicability in Section 6.4 and conclude in Section 6.5 with numerical comparisons and a quantitative evaluation.

6.2.1 Contributions

This chapter briefly reviews the mathematical basis, allowing for accurate solutions of mass property integrals of objects with varying densities. It is **simple** as computations for tetrahedra with constant density easily extend towards varying density. It is **precise** due to the use of a tet-mesh itself as input, therefore, avoiding aliasing bias from discretization, as it is common in state-of-the-art applications using axis-aligned voxelizations. It is **fast** as computations can be implemented as matrix-vector multiplications, suitable for vectorized or GPU execution. It is **versatile** because the framework is not limited to density and easily applies to integrate other linear properties over a volume.

Our application framework, introduced in Section 6.4, can be summarized with the following main contributions: • **Arbitrary Objects:** The solutions, known for tetrahedra, can be straightforwardly generalized to arbitrary polyhedra. We can accurately determine mass properties for any tetrahedralized input object in specified density fields. • **Opti-**

mizing Density: We can also invert the problem and optimize a parameterized density field for an object and given mass properties. • **Approaching non-linearity:** Arbitrary non-linear functions can be approximated in a *Taylor*-like piecewise linear fashion, limited in accuracy only by the tetrahedralization's resolution. • **Additive manufacturing:** We eventually propose combining the former two contributions to 3D-print objects of non-linear density with optimized mass properties for balance or rotation-aligned inertia momentum. • **Expressing energy functions:** As the method is not bound to only physical characteristics, we extend a Lloyd relaxation procedure based on L_2 Voronoi cells: Using the concept to replace a cell's energy integral allows for an accurate closed-form solution for minimizing an L_∞ objective function.

6.2.2 Related Work

Over the last few years, 3D printing not only attracted the do-it-yourself hobbyist community but also gained popularity in various industrial applications. Nowadays, additive manufacturing processes go way beyond stacked layers of plastic and support a wide range of multiple or mixed materials, even including metals. Its widespread use, for example, in medical applications [YDS*18] or the automotive industry [LBA*17], keep this a relevant field of research with great potential.

Hence the general interest of the computer graphics community for analyzing and processing 3D geometry, research in this field also spawned state-of-the-art algorithms aiming at 3D manufacturing. The procedure introduced by Prévost et al. [PWLSH13] allows 3D models to be balanced in a specific position by shifting the object's center of gravity over a safe area on which the object eventually stands. Optimized weight distribution is achieved by carving out the object's interior and deformations of the hull, if necessary. Advancements of this technique optimize objects to have rotation-symmetric weight distributions and allow them to spin like toy-tops [BWBSH14]. Multistable balancing states are accomplished by using movable masses [PBJSH16]. However, established techniques for mass property optimization are still based on approximations of the actual volume and mass distribution using quantized voxelizations. The approach by Musialski et al. [MAB*15] utilizes offset surfaces for shape and mass property optimization but also relies on binary material distribution. Even publications specialized on varying density distributions for manufacturing [WAWS17, KWW19] discretize their density field with marching cubes [LC87] or octrees [Mea82] combined with dithering techniques.

Known methods for computing exact mass properties of polyhedral bodies [Mir96] are restricted to constant density. Like ours, they are based on integrals over the volume and surface of an object. A later revision [Ebe02] made the concept feasible for implementation. The varying density of polyhedral bodies, however, was first studied in the field of geophysics and concluded with a focus on gravitational fields [Han99, D'U14] but not general mass properties. Our approach for computing accurate masses and mass centers

under varying densities relies on a tetrahedral decomposition of the input object. *TetGen* [Si15] is a tetrahedralization tool for polyhedral manifolds based on a Voronoi/Delaunay tessellation. Most recently, *TetWild* [HZG*18] introduced another fast and robust way to tetrahedralize any given 3D triangle soup, providing many adjustable parameters to the user. Our results for examples and applications are based on the outputs of both tools but often specifically on *TetGen* since it can preserve the original input surface. However, any other tet-meshing pipeline will generate equally suitable input as well.

As mentioned with *TetGen*, tetrahedralization is closely related to Voronoi- and Delaunay graphs. In our Section 6.4.5 we introduce a novel approach on Lloyd relaxations (based on Voronoi tessellations) using the L_∞ norm, as they are utilized in Section 5.5 for the generation of at-most-hexa meshes. Ray et al. proposed a concept for very fast meshless Voronoi [RSLL18] and restricted power diagrams [BAR*21] on the GPU, however, both only in the common L_2 space. Nevertheless, it definitely is a promising task to explore combinations of their diagrams and our take on L_∞ relaxations.

6.3 Concept

Our goal is to approach mass properties for polyhedral manifolds of varying densities with analytical tools. In order to compute the mass, the center of mass or other related quantities in a field of varying density that is bounded by a triangle mesh, we use closed-form solutions on its tetrahedralization. These kinds of approaches are admittedly standard in FEM but so far rarely have been used in computer graphics. Therefore, this section briefly summarizes the essentials and introduces the geometric concept that allows for computing these quantities exactly for linearly varying density fields inside a tetrahedron. Detailed derivations of the resulting equations are featured in Appendix 6.A.

6.3.1 Problem Statement

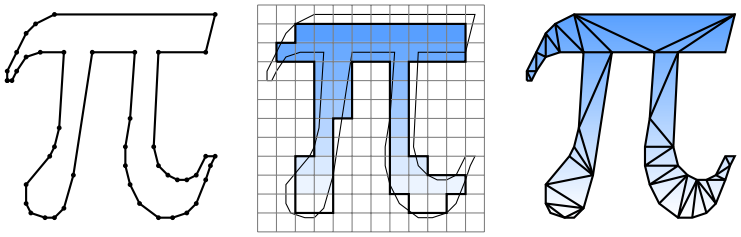
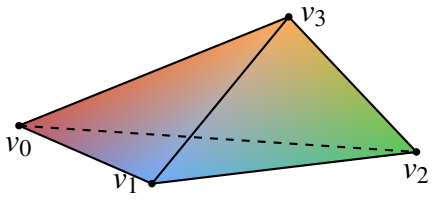


Figure 6.2: 2D example: The common solution to determine mass properties under varying density is to approximate the input with a voxelization. With tetrahedra, the accurate representation on the right can be used to produce precise analytic results.

Mesh data structures are the straightforward and, therefore, most common way to store and represent 3D manifolds. Under constant density, mass properties like the center of mass or an inertia momentum can be easily computed directly on a mesh using the divergence theorem. With varying density, however, a volume bounded by arbitrarily shaped polyhedra cannot be directly integrated. A common fallback solution is to approximate the object shape by decomposition into feasible volume entities, usually voxels. However, as illustrated with the 2D example in Figure 6.2, the success and precision of the approximation will always be limited by the chosen resolution (for space and values). Even hierarchical concepts can only reduce sampling artifacts but not avoid them entirely. In contrast, our solution for the computation under varying density is based on the simple idea of an alternative volume representation, namely, the tetrahedron. As illustrated on the right in Figure 6.2 with a trivially triangulated 2D shape, every 3D shape with a polygonal surface can be decomposed using tetrahedra. With tetrahedra, the mesh's actual shape can be used in all computations and, therefore, corresponding results are free of discretization and aliasing bias.

6.3.2 Geometry Integration



$$T = \begin{pmatrix} v_0 & v_1 & v_2 & v_3 \end{pmatrix}^T = \begin{pmatrix} x_0 & y_0 & z_0 & d_0 \\ x_1 & y_1 & z_1 & d_1 \\ x_2 & y_2 & z_2 & d_2 \\ x_3 & y_3 & z_3 & d_3 \end{pmatrix}$$

Figure 6.3: A general tetrahedron T with varying density, defined by its four vertices v_i. In addition to their geometrical dimensions $(x, y, z) \in \mathbb{R}^3$ each vertex is attributed with a fourth dimension d, encoding density.

Computing mass properties for the general tetrahedron T, specified in Figure 6.3, is trivial for constant density: For example, with $d_i = 1$ the mass is equal to the tetrahedron's volume, and the center of mass is equal to its centroid. However, as the density attributes at each vertex can be individually specified, expressing a linear density field inside the tetrahedron, the computation of mass and mass center changes. Instead, as in FEM [RR96, RNV07], we utilize a simple basis case in a linear density field, for which the integration is solved analytically. A linear combination of four base cases (one per vertex) already gives the desired properties for a general tetrahedron.

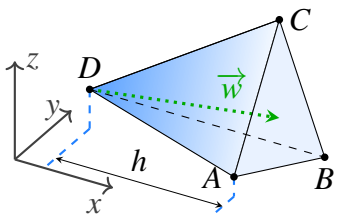


Figure 6.4: The basis case for an *integrable* tetrahedron with $d_D = 1$ and $d_{A,B,C} = 0$. The density gradient over the extent of h is visualized with fading blue. $\overrightarrow{w}$ indicates the center of mass vector.

6.3.3 Mass Properties for Arbitrary Tetrahedra

Mass Due to linearity, the mass for a tetrahedron with four different density values at its vertices can be simply expressed as the mean of these values times the volume, formulated in Equation (6.1) and derived in Appendix 6.A, Equation (6.A.2).

$$\mathbf{M}_T = \mathbf{V}_T \frac{d_A + d_B + d_C + d_D}{4} \tag{6.1}$$

Center of Mass As expressed in Equation (6.2), the mass center computes as the weighted sum of vertex positions and their normalized density values. An extended derivation for the combination of the four base cases to this general form is included in Appendix 6.A, Equation (6.A.4).

$$\mathbf{C}_T = \frac{1}{5}\left(A + B + C + D + \frac{A \cdot d_A + B \cdot d_B + C \cdot d_C + D \cdot d_D}{d_A + d_B + d_C + d_D}\right) \tag{6.2}$$

6.4 Application

With the presented approach, one can now calculate the mass and other mass properties of tet-meshes with arbitrary density fields efficiently and exactly. The power of the approach will be demonstrated in three different application scenarios: First, as a fast and accurate replacement for widespread voxel-based approximations of arbitrary objects' mass properties. Further, it can be used to optimize the density distribution inside an object to obtain a given center of mass or a stable rotation axis, potentially even with non-linear density fields. By introducing a closed-form solution, our concept even allows us to formulate an objective function in a volumetric Lloyd relaxation process, which was so far not analytically feasible.

6.4.1 Mass Properties of Arbitrary Objects

With the techniques introduced in Section 6.3, to compute mass and center of mass for general tetrahedra, we can generalize this concept further and approach arbitrary polyhedral manifolds: Objects are partitioned into tetrahedra, mass properties are determined

individually and results eventually recombined. Any tetrahedral mesh is suitable as input for our method; if the model is not already available as tetrahedral mesh, it can easily be generated using freely available tools like *TetGen* [Si15] or *TetWild* [HZG*18]. Appendix 6.D proves the concept to be invariant of the actual tetrahedralization.

$$\begin{aligned} \mathbf{M_O} &= \sum_{T_i \in \mathbf{O}} \mathbf{M}_{T_i} \\ \mathbf{C_O} &= \frac{1}{\mathbf{M_O}} \sum_{T_i \in \mathbf{O}} \mathbf{M}_{T_i} \mathbf{C}_{T_i} \end{aligned} \tag{6.3}$$

For an object $\mathbf{O}$ and a given density-field, one can now compute the accurate mass $\mathbf{M}_{T_i}$ and center of mass $\mathbf{C}_{T_i}$ for all tetrahedra $T_i \in \mathbf{O}$ using Equation (6.1) and Equation (6.2), respectively. These calculations can be executed very efficiently, using fast matrix-vector multiplications. As formulated in Equation (6.3), the object's overall mass is obtained by simple summation and the center of mass as mass-weighted dot-product. Further, one may extend the derivation, as described by Tonon [Ton04] for the inertia tensor $\Theta_{\mathbf{T_i}}$ of a general tetrahedron with specified density values of the individual vertices. Following the rules for rigid bodies and the parallel axis theorem, one can derive other mass properties like the moment of inertia as formulated with the inertia tensor $\Theta_{\mathbf{O}}$ in Equation (6.4), where $\mathbf{I}_3$ is the 3×3 identity matrix and $\otimes$ the outer product.

$$\begin{aligned} \hat{\mathbf{C}}_{T_i} &= \mathbf{C}_{T_i} - \mathbf{C_O} \\ \Theta_{\mathbf{O}} &= \sum_{T_i \in \mathbf{O}} \Theta_{\mathbf{T_i}} + \mathbf{M}_{T_i} \left(|\hat{\mathbf{C}}_{T_i}|^2 \mathbf{I}_3 - \hat{\mathbf{C}}_{T_i} \otimes \hat{\mathbf{C}}_{T_i} \right) \end{aligned} \tag{6.4}$$

6.4.2 Optimizing Density Fields

Now that an object's center of mass can be determined for a given density field, one can invert the problem and fit a density field to an object where the mass properties are given. As an exemplary use case, we approached the make-it-stand challenge described by Prévost et al. [PWLSH13] to balance objects in a given pose. The center of mass has to be within certain boundaries of a projected surface polygon on which the object is supposed to be balanced. However, a solution to this problem is limited by the following constraints: **i)** Negative mass is reasonable only in theoretical fields of physics, so we limit our model to the realm of positive density for now. **ii)** Zero density is a special case that can be modeled with our concept, e.g., with $d_i = 0$. **iii)** The shape of an object together with constraints (**i**) and (**ii**) will put some limits on the achievable location of an object's center of mass, i.e., it simply cannot pass a certain point.

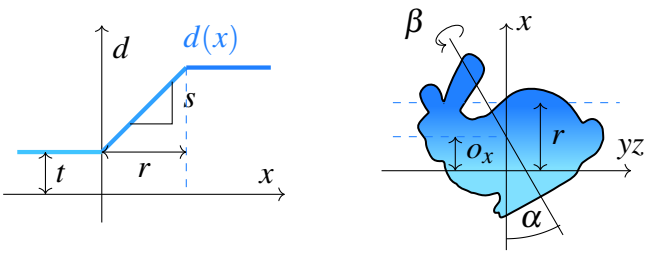
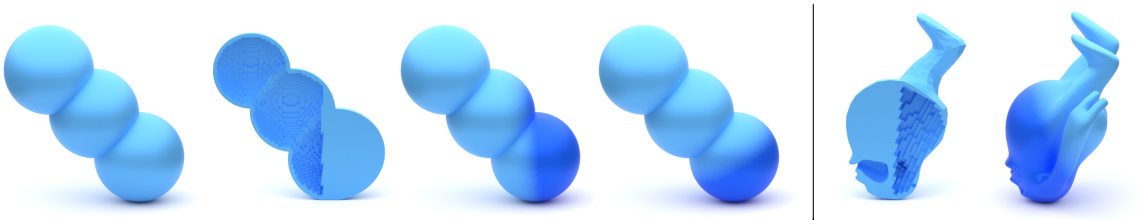


Figure 6.5: A simplified linear density field for optimizing the center of mass. Left: The parameterization of a density field $d(x)$ with parameters r, s and t. Right: Embedding of the object, with angles α, β and offset o_x.

Rather than optimizing the per-vertex density directly, let's first consider a simplified density field as illustrated in Figure 6.5. We utilize two planes, that separate volumes of constant minimum $d_{\min}$ and maximum density $d_{\max}$ respectively, sandwiching a slice of volume of width r with linearly growing density $\in [d_{\min}, d_{\max}]$. To simplify many computation steps we fix the density field to be axis aligned, i.e., the planes are parallel to the yz plane. As accommodation for this fixed orientation of the field, the optimization needs to rotate and to translate the object accordingly instead. To realize the arbitrary location of the bisection-planes in the density function, we have to prepare our input mesh by intersecting some of the tetrahedra, see Appendix 6.B for details. The energy to be minimized is formulated as the Euclidean distance $E = |\mathcal{C} - \mathbf{C}_O|$ between a target point $\mathcal{C}$ and the object's current center of mass $\mathbf{C}_O$ when embedded in the density field, obviously with respect to the object's rotated and translated state. The following six parameters are determined by the optimization:

α, β	angles for tilt and rotation of the object
o_x	object center offset on the x-axis
r	width of the density range
s	steepness of the density gradient
t	constant density offset

The optimization energy is smooth and in some sense probably differentiable but deriving gradients is left for future work. In our experiments we used Powell's method [Pow64] to minimize the objective function.



Unstable Input | Prévost et al. | Ours (balanced) | Ours (standing) | Prévost et al. | Ours

Figure 6.6: An unbalanced input of three spheres and a figure which is supposed to stand upside down. Prévost et al. [PWLSH13] managed to balance the objects by deforming them, carving out the interior and shifting the center of mass over a defined safe region. Our *balanced* version of the spheres can also be balanced on a small flattened face. The *standing* version will roll into this position on its own due to its low center of gravity. Varying density is sufficient to balance the objects, and deformation is not required. As a reference for Table 6.1, the spheres are scaled to have a diameter of 1, and *MrHumpty* has a hand-to-hand width of 2.

Results Figure 6.6 compares our balanced objects to cross-sections of Prévost et al [PWLSH13]. Their proposed method found a solution to make the three spheres stand, by carving out the voxelized interior and deforming the object. To move the mass center of the *Spheres* into the balance region, the top sphere is shrunk and the bottom sphere enlarged. For the *MrHumpty* figure to stand upside down, the belly is enlarged and half the interior carved out to compensate for the off-axis legs. In our results the objects remain untouched as they are only embedded in an optimized density field. As our output geometry incorporates the input, error measures like the HD or RMSE are simply 0.

Our first experiment meets the same conditions as Prévost et al. where the center of gravity only has to be on the central vertical axis of the bottom sphere so that the object is in balance. For our next experiment, we chose the center of mass to be located centered in the bottom sphere, 10% of the sphere's radius below its horizontal equator-line. Due to this low center of gravity, the *standing* spheres would roll into this position on their own.

	α	o_x	r	s
Spheres (b)	−1.666 670	−0.278 929	0.404 357	13.418 110
Spheres (s)	−0.513 042	−0.453 811	0.565 776	26.173 806
MrHumpty	1.104 747	0.244 073	0.562 282	2.614 978

Table 6.1: The density field parameterizations as determined by the optimization for balancing the objects shown in Figure 6.6 with $\beta = 0$ and $t = 1$.

Our optimization managed to define density fields for which the object's center of mass is exactly on the specified axis or target point, respectively. The results have regions of constant minimum and maximum density with a tilted and shifted gradient between them. Due to the symmetry of the objects, the angle β is zero. To approach somewhat reasonable manufacturing limits, we set $t = 1$ ($\hat{=} d_{\text{min}}$). The other found parameters are given in Table 6.1. Results of this comparison should be seen as a theoretical proof of concept, as this relatively unconstrained optimization leads to exceptionally high values for the gradient steepness s. Density differences of this multitude are currently probably not feasible with single-material manufacturing techniques. Additive multi-material techniques, on the other hand, could only approximate such gradients with dithering.

6.4.3 Optimizing Non-Linear Fields

Section 6.3.3 introduced our concept for density fields with a generalization to define geometry-independent density values per vertex. This allows for the approximation of arbitrary non-linear fields, as illustrated with the examples in Figure 6.7: The *Bunny* is embedded in a spherical sinusoidal density function, and the density in the *Femur* decreases from surface to core with a Gaussian slope. Tet-mesh vertices become 3D sampling positions for the 3D density field. However, gradients within each tetrahedron are still linear. Nevertheless, this piecewise linear *Taylor* approximation of a non-linear field is C_0 continuous everywhere (C_∞ within a tetrahedron). The accuracy of this representation is only limited by the resolution of the tetrahedral mesh, which can be specified in standard tetrahedralization tools.

$$d(v) = \sin(|p - v| \cdot k) + 1 \tag{6.5}$$

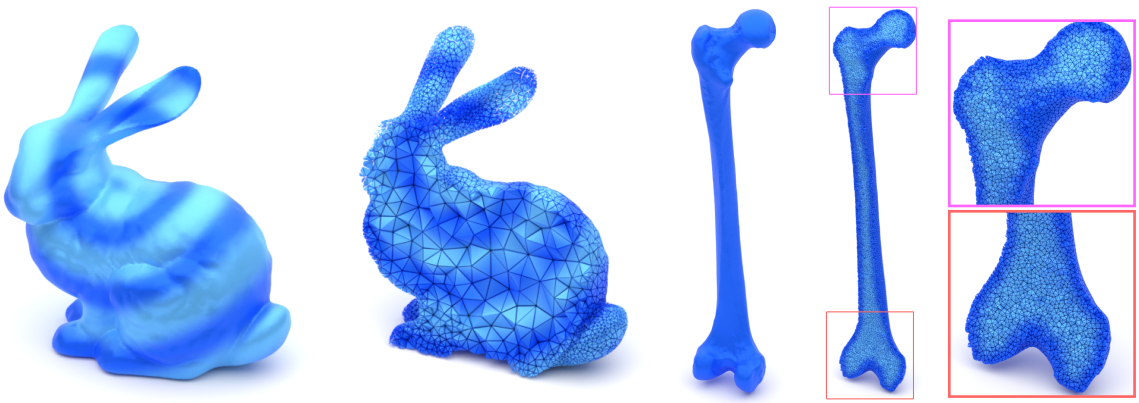


Figure 6.7: Examples of non-linear density fields, sampled at vertex positions. Tetrahedralizations were created with *TetWild* on default settings for the *Bunny* and with a smaller edge-length for the *Femur*.

Advanced applications specifying more than a single center of mass may require density fields, more sophisticated than linear gradients. An approach related to the make-it-stand concept proposed the challenge to make objects spinnable [BWBSH14] by moving mass centers to a specified rotation axis. This is not only a desirable criterion for toy tops or yo-yos but is also of great value in any mechanical process involving rotating movements to reduce the wear and tear of involved components. The engineering of such mechanical components often comes with tight constraints on available space and does not allow for arbitrary placement of counterweights. Figure 6.8 illustrates an example with the *Rockerarm* object, which is to be mounted on rotary bearings. With constant density, the native center of gravity (red) and inertia tensor are off-axis due to the obvious asymmetry of the object. For this object, the optimized density field results in the center of mass located on the rotation axis along with a parallel principle inertia momentum axis. The density field is parameterized with the non-linear density function $d(v)$ (Equation (6.5)), where v is a tet-mesh vertex, p a 3D coordinate and k a scalar factor.

Results Optimizing for a specific center of mass, as in Section 6.4.2, is not trivial but possible, dependent on given constraints. Additionally fitting a principle inertia momentum axis can be challenging as the density distribution for a specific center of mass may be in conflict with the optimal density for the momentum axis. A field parameterization with more degrees of freedom than ours (Equation (6.5)) might be more suitable for optimization but unreasonable for practical results. Our results are shown in Figure 6.8 with an optimized center of mass (green). The density parameters are:

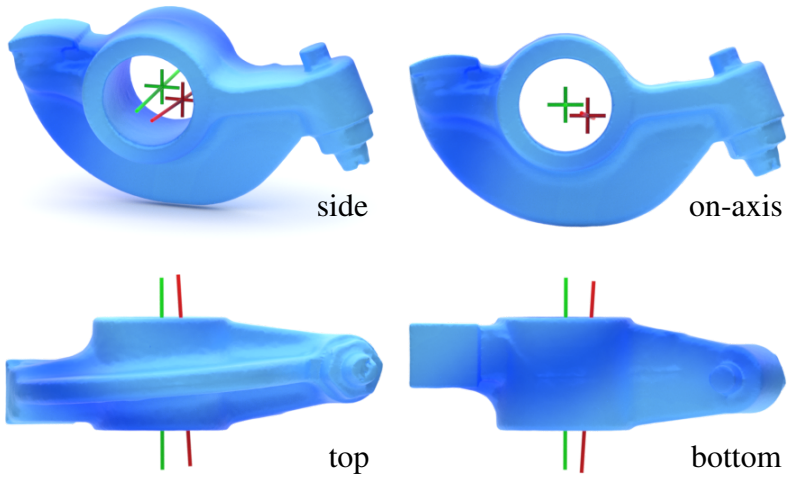


Figure 6.8: The *Rockerarm* is a prominent example of an asymmetric object with a rotary mount. Due to imbalance, the native center of mass is not located on the rotation axis. With optimized density, our center of mass is located on the rotation axis, and the principle inertia momentum axis is parallel to the rotation axis.

$$p = \begin{pmatrix} 0.509475 \\ -0.699066 \\ 1.328176 \end{pmatrix} \text{ and } k = 7.853375$$

The principle inertia momentum axis was met with accuracy of $< 1°$, the center of mass is actually precisely located on the specified target rotation axis.

6.4.4 3D Printing of Varying Density

For some objects, we demonstrate both synthetic results as well as 3D printouts. One has to mention that 3D printing hardware for varying density is still in an early development state [Onl20, LFF*20, HKD*20], and the range of available densities is limited. On recent *Prusa* FDM printers, however, it is possible to alter the extrusion rate while printing. The first step is to obtain the G-code for an object with a regular infill pattern. In order to approximate the optimized density field, we modify the line thickness to vary along with printed segments by accordingly adjusting the relative extrusion rate in the G-code slice by slice. The default infill pattern density should be set to 50%. Then, a modulated extrusion rate, ranging from 0-200%, can approximate the used sinusoidal density function, formally expressed in Equation (6.5).

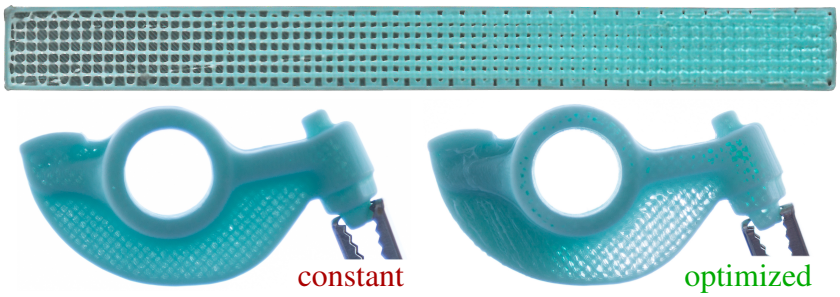


Figure 6.9: Modifying the extrusion rate allows for printing density gradients. The bar on top is a cross-section of an object 10×10×100 mm in size, scanned with a photocopier. Pictures of the *Rockerarms* (62.9×33×17.6 mm) were taken in front of a light source to highlight the different density distributions.

Figure 6.9 shows a 3D printed example of a simple bar with increasing density. The varying amount of printed material alters the translucency of the object. The 3D printed *Rockeram* of Figure 6.8 with optimized density for an on-axis center of mass and a parallel principle inertia momentum axis leads to significantly smoother spinning.

6.4.5 Lloyd Relaxation with the L_∞ norm

The last proposed application scenario makes use of the closed-form integral solution of measures on volumes of varying density to approach Lloyd relaxations under non-standard norms (Section 2.4.1). All calculations are done on the actual Voronoi cells, avoiding voxelization, which reduces artifacts and speeds up the computation.

Section 2.4 provides a detailed introduction to Lloyd's algorithm [Llo82]. However, to briefly review the main characteristics: It is an iterative optimization procedure, that is proven to converge to CVTs under the L_2 norm [DEJ06]. The iteration alternates two steps: **I.** Compute a Voronoi diagram for a given set of points. **II.** Reposition each point to the centroid of its Voronoi cell. This can also be formulated as an optimization task, minimizing the diagram's global energy function. Since the native L_2 cells are all convex, the computation of new centroids is quite simple. However, in many meshing applications, L_p or even L_∞ are more desirable [LL10], due to their more rectangular or cubical cell shapes. For L_p norms ($p > 2$), the Voronoi cells are no longer always convex, and the diagram becomes very impractical to handle or even generate since there is (to the best of our knowledge) no software library that is able to compute L_p or L_∞ Voronoi tessellations. In meshing applications [LL10, SRUL16] the diagram itself is actually not relevant, but only the site positions are of interest [RSLL18].

We propose a way to compute Lloyd relaxed site positions with the Chebyshev L_∞ norm: First, cell geometry and topology are borrowed from an L_2 tessellated diagram, which comes with the convenience of convex-only cell shapes. Then our method computes a cell's mass, reinterpreted as the energy which is to be minimized by a new cell center.

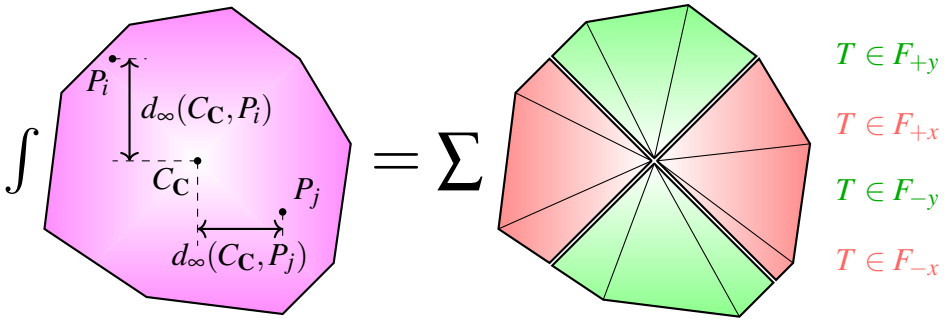


Figure 6.10: 2D visualization of the equivalent energy terms of Equation (6.7) with an integral over the *Chebyshev* distances of all points in a cell (left) or the sum over its tet-fragments with density gradients. (right)

The Energy Term For the goal to minimize the L_∞ energy within a cell, let us briefly recapitulate how the *Chebyshev* distance d_∞ is defined, also see Section 2.4.1. As formulated in Equation (6.6), the distance between two points p and q is the maximum of their absolute differences over all dimensions, in the 3D case x, y and z.

$$d_\infty(p,q) = \max_{k\in[x,y,z]} |q_k - p_k| \tag{6.6}$$

$$\begin{aligned} E_{\mathbf{C}} &= \int_{P\in\mathbf{C}} d_\infty(C_{\mathbf{C}},P) \\ &= \sum_{k\in[\pm x,\pm y,\pm z]} \sum_{T\in F_k} \mathbf{M}_T^k \end{aligned} \tag{6.7}$$

Equation (6.7) formulates the energy $E_{\mathbf{C}}$ of a cell $\mathbf{C}$ as the total *Chebyshev* distance of all points $P \in \mathbf{C}$ with respect to the cell's centroid $C_{\mathbf{C}}$. However, there are infinitely many points $P \in \mathbf{C}$, so the energy function can only be evaluated with a nontrivial integral over the cell volume. As illustrated in Figure 6.10 (in 2D), this integral becomes feasible as a finite sum of analytical solutions. To achieve this, a cell is split into six fragments F_k ($k \in [\pm x, \pm y, \pm z]$), as illustrated in Figure 6.11. This effectively separates all points P within the cell with respect to their maximum difference-dimension (*Chebyshev*). Due to the separation into the six fragments, the $d_\infty(C_{\mathbf{C}},P)$ distance dimension conveniently coincides with the corresponding geometric dimension k, i.e., the distance linearly increases along one of the coordinate axes. The six cell fragments are tetrahedralized using a trivial triangulation of their hull faces and the cell center itself. The inner sum in Equation (6.7) accumulates masses $\mathbf{M}_T^k$ of all tetrahedra T in a fragment F_k as defined in Equation (6.1). The outer sum accumulates the density- (or *Chebyshev* distance-) weighted volumes of the six fragments, resulting as the cell's L_∞ energy. The cell center is finally repositioned to minimize the computed L_∞ energy using the *L-BFGS-B* algorithm [BLNZ95, ZBLN97] for bound constrained minimization.

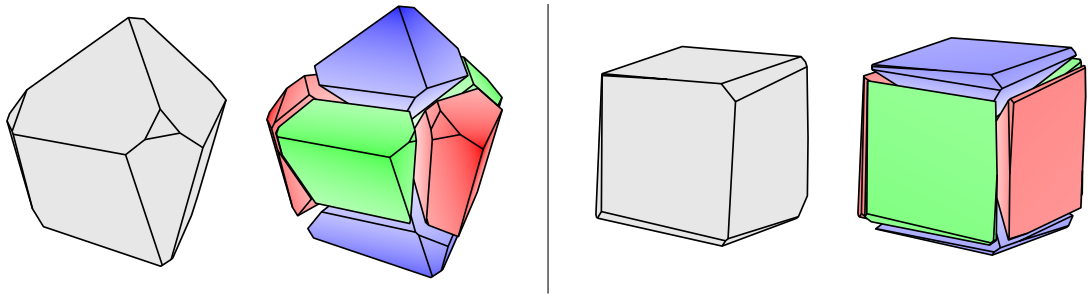


Figure 6.11: Voronoi cells under the L_∞ norm before (left) and after the relaxation (right). Saturation visualizes the L_∞ energy, increasing in each dimension. Centroids in cubical cells minimize this energy.

Results Although cell energies are only optimized on an individual basis, the relaxation iteration also leads to a global decrease of the diagram's energy, analogously to the L_2 case. With our reformulation of the objective function, the second part of the Lloyd relaxation (repositioning of cell centers) becomes feasible for the L_∞ norm. The initialization of each iteration is still based on a computable L_2 Voronoi tessellation, which

turned out to be sufficient as the relaxation still converges. Considering the shape of an L_2 Voronoi cell while optimizing the centers for the L_∞ energy, this optimization is not a complete L_∞ relaxation but a convenient alternative. If an L_∞ tessellation was available, the relaxation would probably converge even faster and would also allow for individually oriented cells. Nevertheless, considering the alternatives, like labeling underlying high-resolution voxel grids, it is an improvement in terms of both accuracy and performance.

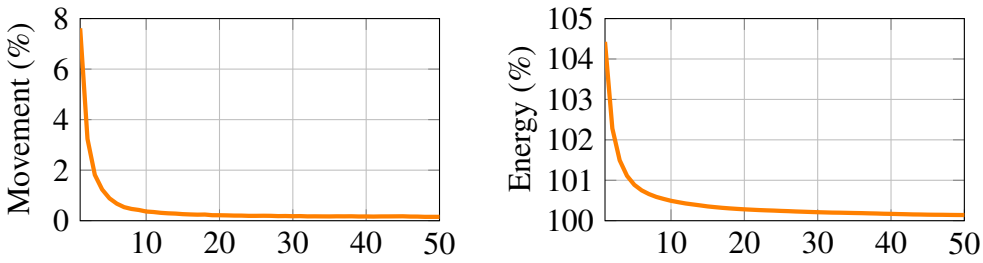


Figure 6.12: L_∞ relaxation plots of the cube from Figure 6.13b over 50 iterations. Left: Average movement of all cell centers in one iteration, given in percent of the optimal cell's diagonal. Right: Average L_∞ energy of all cells, given in percent of an optimal cell's L_∞ energy, which is why the result converges to 100%.

The plots in Figure 6.12 show convergence results of the cube (Figure 6.13b) over the course of 50 Lloyd relaxation steps. The *Movement* plot shows the average distance traveled by all sites (cell centroids) in the Voronoi diagram during each relaxation step. This distance is given in percent of an optimal cubical cell's diagonal. The average cell energy (in the *Energy* plot) computes for each cell as illustrated in Figure 6.11 by separating a cell into six fragments and accumulating the density-weighted volume. It is expressed in the percent of an optimal cubical cell's L_∞ energy. Therefore, the convergence towards 100% indicates that our cells approach the anticipated optimal cubical cell shape.

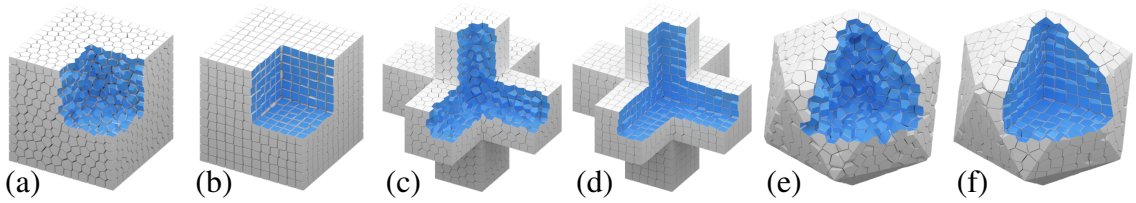


Figure 6.13: Results after 50 Lloyd relaxations: A unit-cube on the left and two Clipped-Voronoi-Diagrams [YWLL13] in the middle and on the right (Cutouts in blue). Examples (a),(c) and (e) used the L_2 norm to reposition their sites in each step, the examples (b),(d) and (f) show results using the L_∞ norm, generating close to cube-like cells. Despite different relaxation norms, all results are visualized as L_2 tessellations.

6.5 Discussion

This section presents the results of proposed application scenarios for our concept. Benefits over traditional methods in terms of performance and accuracy are quantitatively discussed with numerical results. After an outlook to potential extensions and future work, we conclude with a roundup of our main contributions.

6.5.1 Results

Input			*TetGen*		Ours (tet)	Octree (depth 4)				Octree (depth 8)			
Object	$\#v_{\text{in}}$	$\#f_{\text{in}}$	$\#t$	time (s)	time (s)	$\#n_4$	time (s)	ε_C (%)	ε_M (%)	$\#n_8$	time (s)	ε_C (%)	ε_M (%)
buddha	99370	198736	364531	3.0942	0.1982	2047	5.8286	0.4362	0.8117	730647	637.780	0.0027	0.0432
casting	5096	10224	15827	0.1200	0.0064	1719	1.3123	4.1531	12.1164	911551	480.849	0.7120	0.9831
fandisk	2877	5750	8556	0.0678	0.0035	1505	0.9744	1.0598	0.5505	433740	217.282	0.0329	0.0872
fertility	241607	483226	848832	7.9802	0.4155	1854	10.3933	0.3387	1.3620	695297	996.629	0.0118	0.1219
maxplanck	49132	98260	164562	1.3608	0.0790	1759	4.9667	0.1952	0.0457	561597	446.385	0.0225	0.0765
nefertiti	1009118	2018232	3301743	32.8039	1.7506	1677	56.4184	0.4852	0.6365	570914	2756.926	0.0010	0.0085
rockerarm	10044	20088	31131	0.2560	0.0119	2148	1.7444	0.4042	3.1069	818778	469.041	0.0728	0.1308
trefoil	10240	20480	37473	0.3160	0.0244	2221	1.9554	0.0952	0.4600	825107	448.259	0.0008	0.1352

Table 6.2: A practical performance comparison to octree approximations, where our method generated the reference mass properties. Input objects are listed with their number of vertices and faces. They are uniformly scaled to a unit-height of 1 with density linearly increasing from bottom ($d_{\text{min}} = 0$) to top ($d_{\text{max}} = 1$). Tet-inputs are listed with number of tets, time to generate with *TetGen* using the `-Y` option. Octree results are from two different depths, listing the number of nodes, the computation time (build + traversal) and two error measures (Equation (6.8)).

As mentioned in Section 6.2.2, voxelizations or octrees are currently the most common method to approximate mass properties for objects in fields of varying density. Table 6.2 documents comparisons of our exact tetrahedron-based method to octrees of different depths. Our results provide the ground truth reference, to which the octree approximations are compared. Timings for the octrees include the build-up phase and traversal to compute the results. The most demanding parts in the build-up are in/out-tests to decide if a cell is to be split again. To be comparable, we included the time to create the tet-mesh inputs for our method from basic surface meshes. Delaunay tetrahedralizations are computed with *TetGen* [Si15], using the `-Y` option, which preserves the source mesh so that our method and the octree have the exact same input. Both octree and our method are implemented in *Python* using vectorized *NumPy* arrays where possible for optimized efficiency. Although our method is well suited to be implemented in parallelized GPU code, all timings are measured on a single CPU core. The measurements show that, not only compared to the very deep but also for the small octree of only 4 levels, our method is multitudes faster, even including the input tetrahedralization.

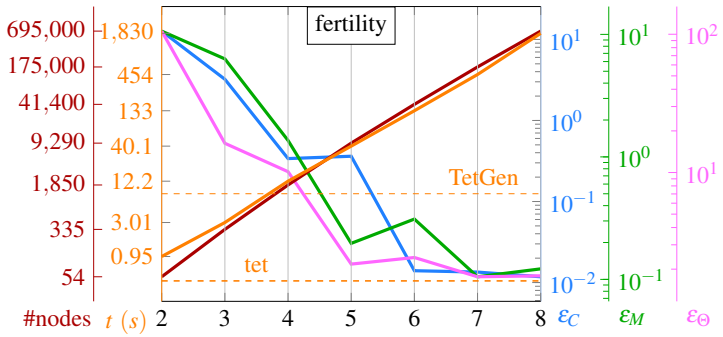


Figure 6.14: By increasing the depth of an octree (# levels on the x-axis), the approximations converge against our analytic results. The plot shows the time in seconds to build and traverse the tree, the number of nodes in the tree, the mass center error ε_C, the mass error ε_M and inertia tensor error ε_Θ. Dashed lines show timings for our computation based on tets and the time it took for TetGen [Si15] to tetrahedralize the input.

Timings and performance aside, the probably most valuable takeaway is the analytical accuracy of the results. Our method establishes actual ground truth results for mass properties under varying densities. Figure 6.14 plots the mass-property-errors of octree approximations (Figure 6.15) converging against our results, as we increase their depth and therefore accuracy. Featured errors of mass ε_M, the center of mass ε_C and the inertia tensor ε_Θ are specified in Equation (6.8).

$$\begin{aligned} \varepsilon_C &= |C_{\text{octree}} - C_{\text{tet}}| \\ \varepsilon_M &= \frac{|M_{\text{octree}} - M_{\text{tet}}|}{M_{\text{tet}}} \\ \varepsilon_\Theta &= \sum_{k \in [x,y,z]} \frac{|\Theta^k_{\text{octree}} - \Theta^k_{\text{tet}}|}{|\Theta^k_{\text{tet}}|} \end{aligned} \tag{6.8}$$

In theory, a voxel grid of infinite resolution or an octree of infinite depth would give correct and unbiased mass property results. We use this capacity to show that octree results converge against our analytic results by increasing their depth and accuracy. The error-plots do not converge monotonically due to aliasing artifacts. For some lower levels, the approximations are just more accurate by chance. More data and plots of all the objects featured in Table 6.2 are included in Appendix 6.C.

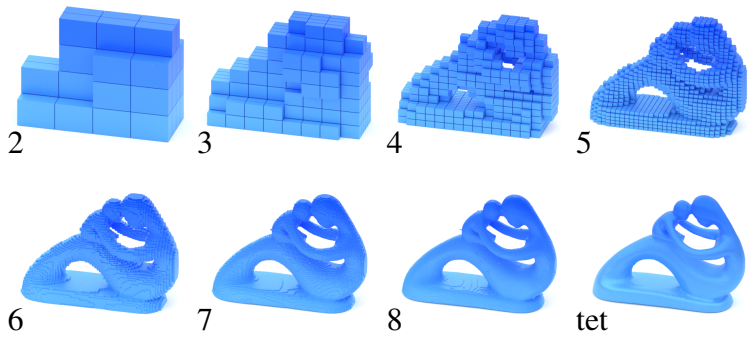


Figure 6.15: Visualization of the octrees used in Figure 6.14. With increasing depth (2-8) of the tree, the approximations for mass properties converge against our analytic results, based on the real triangular boundary mesh.

6.5.2 Conclusion

We propose novel application scenarios for object's mass properties under varying densities. Easy to use analytical solutions make approximations obsolete, which are still common in recent state-of-the-art applications [PWLSH13, BWBSH14, PBJSH16, WAWS17, KWW19]. Our concept is fast, lightweight, easy to implement and suitable for vectorized or parallelized frameworks. We demonstrate possible use cases where our method can be utilized straightforward: Masses, mass centers and inertia tensors of arbitrary manifolds in given density fields are computed accurately. We formulate an optimization to determine a parameterized density field for an object and specified mass properties like a center of gravity or inertia tensor. Our proposed modification of the Lloyd relaxation is a novel $L_{2|\infty}$ hybrid that allows us to imitate real L_∞ relaxations, which is a leap forward compared to the existing approximative alternatives. While our approach may find direct application in established research topics as meshing, spatial tessellation, and simulation [HZG*18, RSLL18], we also see great potential in the scientific field of additive manufacturing and hope to inspire many further research [WAWS17, YDS*18, KWW19].

Chapter 6

Appendix

6.A Derivations

This appendix completes the derivations of the closed-from equations for mass and center of mass calculations used in Section 6.3.

$$\begin{aligned} M_T(D) &= \int_0^{h_D} \overline{ABC} \cdot \left(\frac{r}{h_D}\right)^2 \cdot \left(1 - \frac{r}{h_D}\right) dr \\ &= \overline{ABC} \int_0^{h_D} \frac{r^2}{h_D^2} \cdot \left(1 - \frac{r}{h_D}\right) dr \\ &= \overline{ABC} \cdot \frac{h_D}{12} = \underbrace{\overline{ABC} \cdot \frac{h_D}{3}}_{\mathbf{V}_T} \cdot \frac{1}{4} \end{aligned} \tag{6.A.1}$$

Figure 6.4 illustrates an exemplary basis case with density $d_D = 1$ and 0 at the three other vertices. Since the density is normalized, it is not surprising that the density integral over the volume in Equation (6.A.1) results in a quarter of the tetrahedron's volume $\mathbf{V}_T$. Thus, when combined for a general tetrahedron embedded in an arbitrary density field, the mass $\mathbf{M}_T$ computes as the tetrahedron volume times the mean of all four density values, as formulated in Equation (6.A.2).

$$\begin{aligned} \mathbf{M}_T &= d_A \cdot M_T(A) + d_B \cdot M_T(B) + d_C \cdot M_T(C) + d_D \cdot M_T(D) \\ &= d_A \cdot \mathbf{V}_T \cdot \frac{1}{4} + d_B \cdot \mathbf{V}_T \cdot \frac{1}{4} + d_C \cdot \mathbf{V}_T \cdot \frac{1}{4} + d_D \cdot \mathbf{V}_T \cdot \frac{1}{4} \\ &= \mathbf{V}_T \frac{d_A + d_B + d_C + d_D}{4} \end{aligned} \tag{6.A.2}$$

Equation (6.A.3) formulates the volume integration of the mass center $C_T(B)$ for the shown base case, using a scaled center vector $\overrightarrow{w}$.

$$\begin{aligned}
C_T(D) &= D + \frac{1}{M_T(D)} \int_0^{h_D} \overline{ABC}(r) \cdot \left(1 - \frac{r}{h_D}\right) \cdot \vec{w}(r)\, dr \\
&= D + \frac{1}{M_T(D)} \int_0^{h_D} \overline{ABC} \cdot \left(\frac{r}{h_D}\right)^2 \cdot \left(1 - \frac{r}{h_D}\right) \cdot \vec{w} \cdot \frac{r}{h_D}\, dr \\
&= D + \frac{1}{\overline{ABC} \cdot \frac{h_D}{12}} \overline{ABC} \cdot \vec{w} \int_0^{h_D} \left(\frac{r}{h_D}\right)^3 \cdot \left(1 - \frac{r}{h_D}\right) dr \\
&= D + \vec{w} \cdot \frac{12}{h_D} \cdot \frac{h_D}{20} = D + \vec{w} \cdot \frac{3}{5} \\
&= D + \left(\frac{A+B+C}{3} - D\right) \cdot \frac{3}{5} \\
&= \frac{1}{5}(A+B+C+D+D)
\end{aligned} \tag{6.A.3}$$

The center of mass $\mathbf{C}_T$ in Equation (6.A.4) for the general tetrahedron computes as the combination of the individual base case mass centers, weighted and normalized by the individual density values at the four vertices, respectively.

$$\begin{aligned}
\mathbf{C}_T &= \frac{d_A \cdot B_T(A) + d_B \cdot B_T(B) + d_C \cdot B_T(C) + d_D \cdot B_T(D)}{d_A + d_B + d_C + d_D} \\
&= \begin{pmatrix} d_A \\ d_B \\ d_C \\ d_D \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{5} \cdot (A+A+B+C+D) \\ \frac{1}{5} \cdot (A+B+B+C+D) \\ \frac{1}{5} \cdot (A+B+C+C+D) \\ \frac{1}{5} \cdot (A+B+C+D+D) \end{pmatrix} \cdot \frac{1}{d_A + d_B + d_C + d_D} \\
&= \begin{pmatrix} d_A \\ d_B \\ d_C \\ d_D \end{pmatrix} \cdot \left(\begin{pmatrix} A+B+C+D \\ A+B+C+D \\ A+B+C+D \\ A+B+C+D \end{pmatrix} + \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \right) \cdot \frac{1}{5 \cdot \sum d_i} \\
&= \left(\sum d_i \cdot (A+B+C+D) + \begin{pmatrix} d_A \\ d_B \\ d_C \\ d_D \end{pmatrix} \cdot \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \right) \cdot \frac{1}{5 \cdot \sum d_i} \\
&= \frac{1}{5} \left(A+B+C+D + \frac{A \cdot d_A + B \cdot d_B + C \cdot d_C + D \cdot d_D}{d_A + d_B + d_C + d_D} \right)
\end{aligned} \tag{6.A.4}$$

6.B Tet-Split

To model the density function described in Section 6.4.2, the object is bisected using split-planes. As the input is already tetrahedralized, there might be tetrahedra that are only partially in one or the other region. This potentially violates the constraints of Section 6.4.2, for example, if a tetrahedron would cross the 0-density limit. Consequently, such tetrahedra with vertices on both sides of a split-plane have to be cut. Figure 6.B.1 illustrates the four possible cases, how a plane may intersect a tetrahedron. The splits result in geometry that is always further tetrahedralizable, hence triangular prisms, quad-based pyramids or trivial tetrahedra. Affected tetrahedra in the input are easily identified by checking if they match one of these cases. After the cut, corresponding tetrahedra are simply replaced by the subdivided geometry.

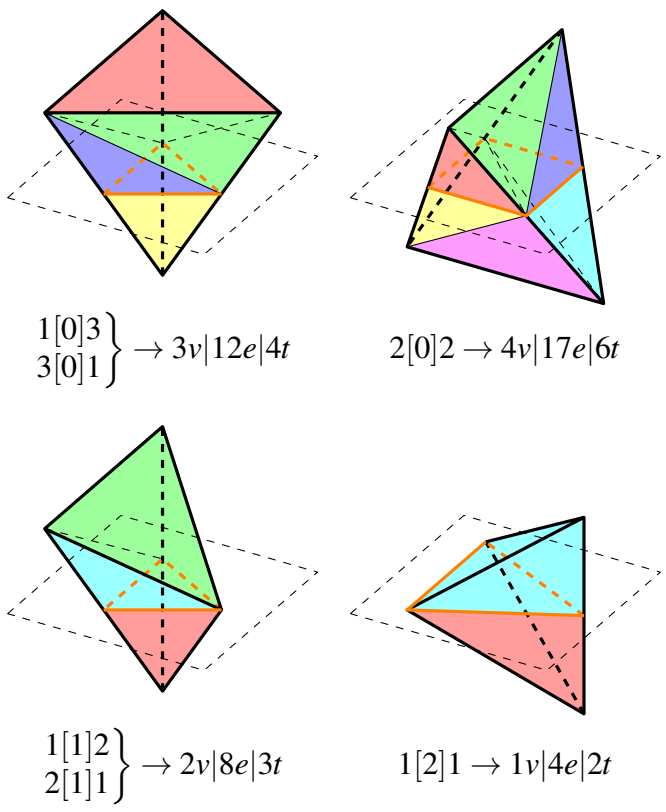


Figure 6.B.1: Up to symmetry or ration, there are four cases, how tetrahedra are split by a plane. The resulting geometry is again tetrahedralizable. The left-hand numbers account for vertices separated by or lying within the plane (in brackets). Right-hand numbers list the number of *new* vertices, edges and tetrahedra created by the split.

6.C Practical Comparisons

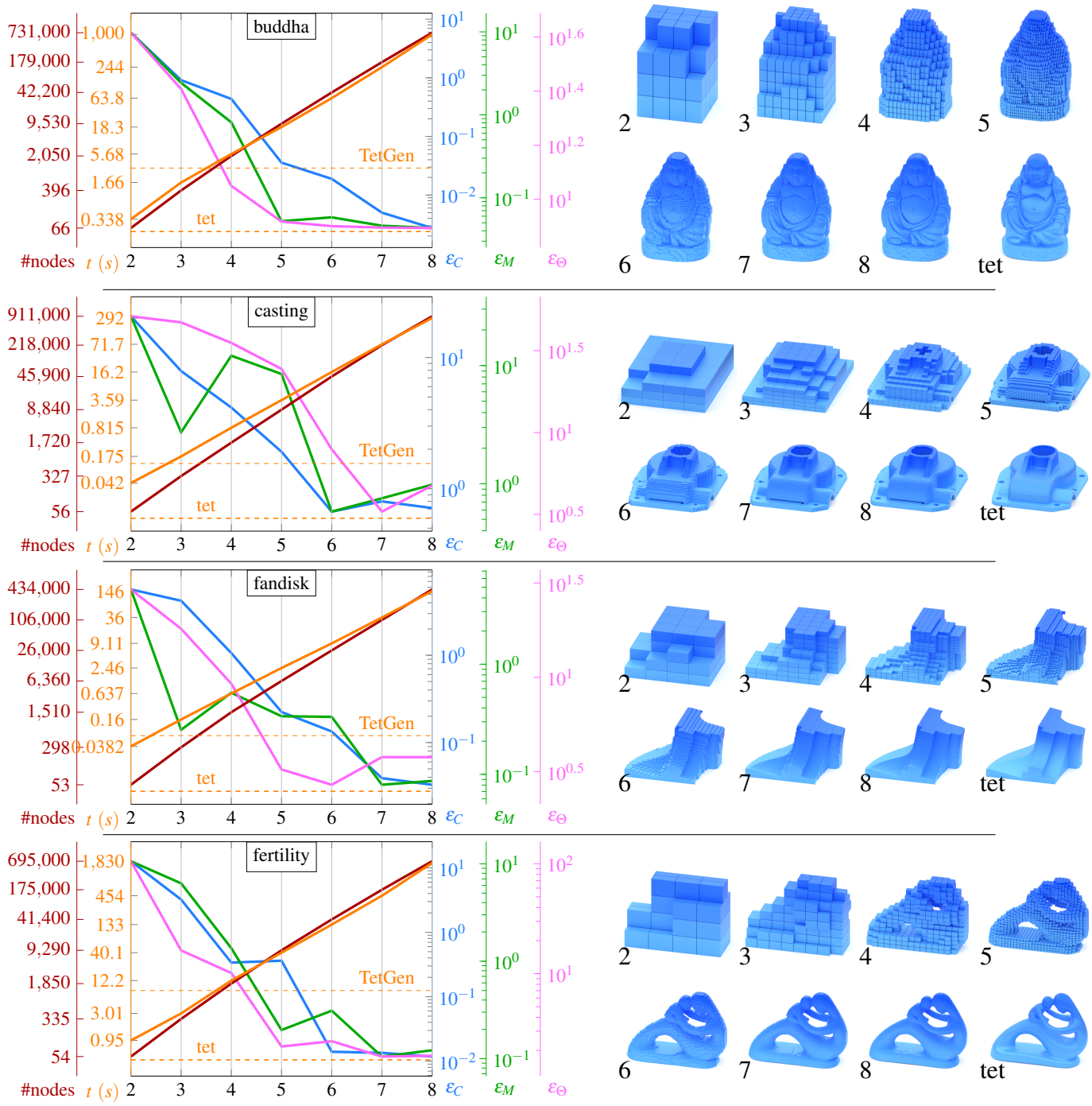


Figure 6.C.2: First half of the objects listed in Table 6.2. Octree approximations converge against our analytic results. Tree depths (2-8) are shown on the plots' x-axis and on each visualization. The plots shows the time in seconds to build and traverse the tree, the number of nodes in the tree, the mass center error ε_C in percent, the relative mass error ε_M and inertia tensor error ε_Θ. Also included are timings for our computation based on tets and the time it took for TetGen [Si15] to tetrahedralize the input. Error measures are defined in Equation (6.8).

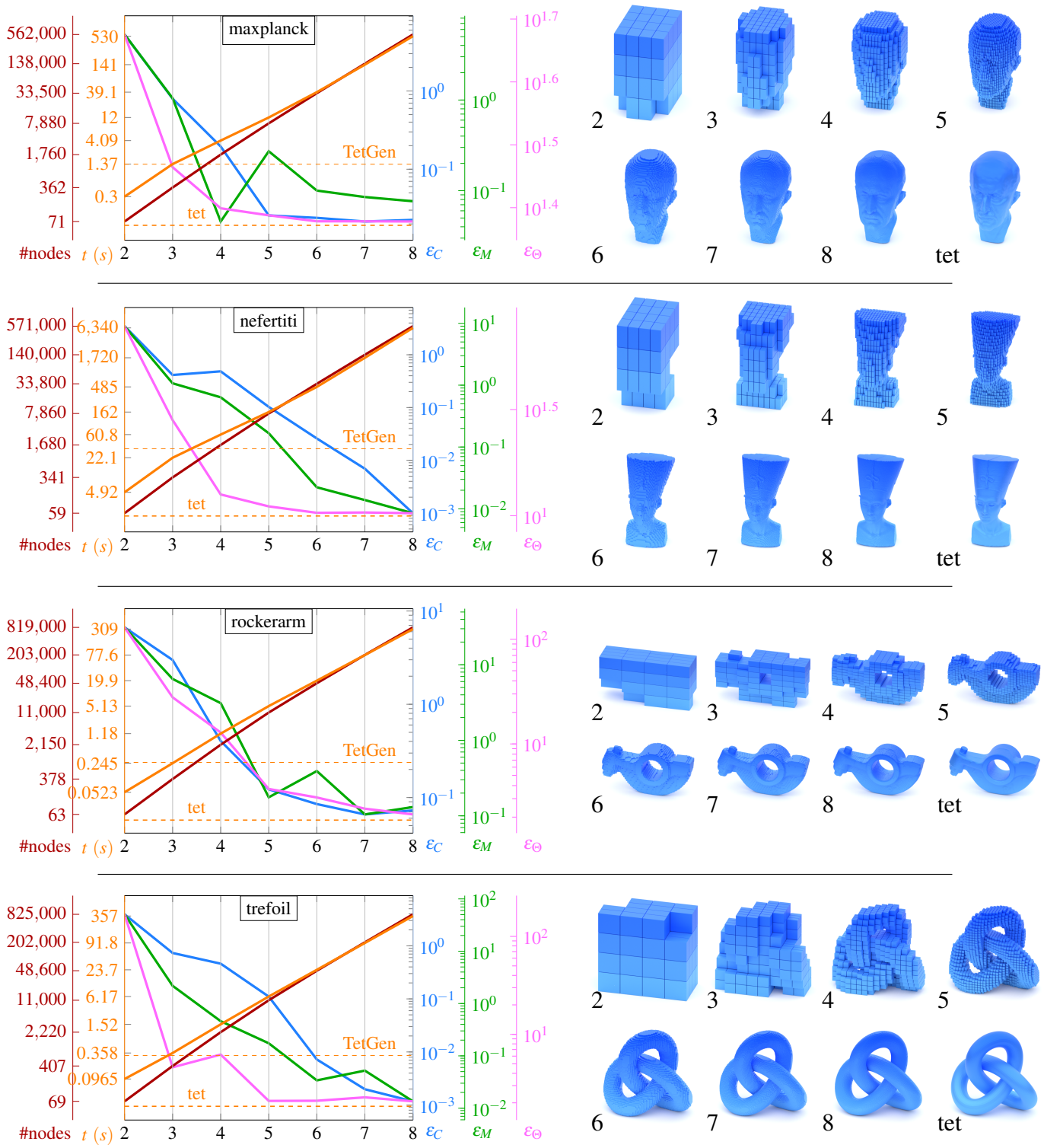


Figure 6.C.3: Second half of the objects listed in Table 6.2. See Figure 6.C.2 for more details.

6.D Proof of Concept

This appendix aims to demonstrate the validity of our approach, especially proving mass property computations of polyhedra to be invariant of the used tetrahedralization. Therefore, we assemble the simple scenario shown in Figure 6.D.4: An axis-aligned box is (w.l.o.g.) centered on the x-axis with the $\langle AEHD \rangle$ quad face parallel to the yz-plane and the $\langle BFGC \rangle$ quad at a distance h to the origin at x_0. The density gradually increases over the x dimension dependent on the density function $d(x) = s \cdot x + t$. Box-related equations are denoted with an overset box-symbol $\square$, the tetrahedra equivalents are marked with a triangle-symbol $\triangle$.

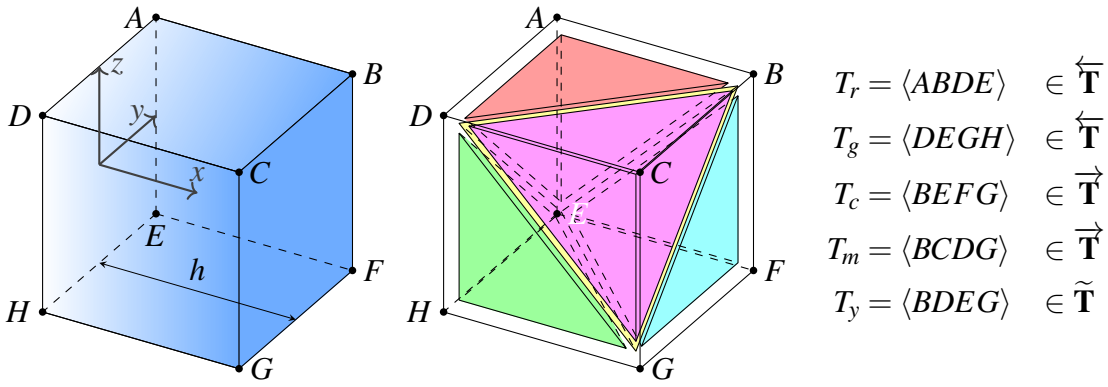


Figure 6.D.4: To prove our concept, we construct an axis-aligned box and simple but general tetrahedralization. The five tetrahedra are assigned to the three integrable base-classes introduced in Figure 6.D.5. Colors **r**ed, **g**reen, **c**yan, **m**agenta and **y**ellow correspond to the indices used in upcoming equations to refer to the individual tetrahedra.

6.D.1 Setup

For the constructed box scenario, it is fairly easy to determine its mass and mass center via integration as formulated in Equation (6.D.5) and Equation (6.D.6), respectively. The cross-section-area of the box is formally expressed as a function $\Phi_B(x)$ over the integration domain, which is constant and, for simplicity, assumed to be 1.

$$\begin{aligned} \overset{\square}{\mathbf{M}}_B &= \int_{x_0}^{x_0+h} \Phi_B(x) \cdot (sx+t)\, dx \\ &= \int_{x_0}^{x_0+h} sx+t\, dx \\ &= \frac{h}{2}(hs+2(sx_0+t)) \end{aligned} \tag{6.D.5}$$

$$
\begin{aligned}
\overset{\square}{\mathbf{C}}_B &= \frac{1}{\mathbf{M}_B}\int_{x_0}^{x_0+h} \Phi_B(x)\cdot(sx+t)\cdot\overrightarrow{w}\cdot\left(\frac{x-x_0}{h}\right)dx \\
&= \overrightarrow{w}\frac{1}{\frac{h}{2}(hs+2(sx_0+t))}\int_{x_0}^{x_0+h}(sx+t)\left(\frac{x-x_0}{h}\right)dx \\
&= \overrightarrow{w}\frac{2}{h(hs+2(sx_0+t))}\frac{h}{6}(2hs+3(sx_0+t)) \\
&= \overrightarrow{w}\frac{2hs+3(sx_0+t)}{3hs+6(sx_0+t)}
\end{aligned}
\tag{6.D.6}
$$

6.D.2 Mass

To demonstrate equality of our approach to the box-solution, we first gather all the tetrahedra-related components. Equation (6.D.7) lists the independent masses of the tetrahedra, using the assumption of the box's cross-section to have an area of 1. The volume of the individual tetrahedra in this configuration compute as $\mathbf{V}_{rgcm}=\frac{1}{6}$ and $\mathbf{V}_y=\frac{1}{3}$. They scale linearly if the box elongates along the x-axis, thus include the factor h. Further, we utilize the simplifications that $d_0 = sx_0+t$ and $d_h = s(x_0+h)+t$.

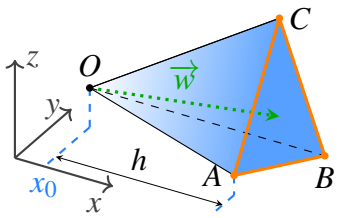
$$
\begin{aligned}
\mathbf{M}_r &= \mathbf{V}_r\frac{d_A+d_B+d_D+d_E}{4} &&= \frac{h}{24}(3d_0+d_h) \\
\mathbf{M}_g &= \mathbf{V}_g\frac{d_D+d_E+d_G+d_H}{4} &&= \frac{h}{24}(3d_0+d_h) \\
\mathbf{M}_c &= \mathbf{V}_c\frac{d_B+d_E+d_F+d_G}{4} &&= \frac{h}{24}(d_0+3d_h) \\
\mathbf{M}_m &= \mathbf{V}_m\frac{d_B+d_C+d_D+d_G}{4} &&= \frac{h}{24}(d_0+3d_h) \\
\mathbf{M}_y &= \mathbf{V}_y\frac{d_B+d_D+d_E+d_G}{4} &&= \frac{h}{24}4(d_0+d_h)
\end{aligned}
\tag{6.D.7}
$$

$$
\begin{aligned}
\overset{\triangle}{\mathbf{M}}_B &= \mathbf{M}_r+\mathbf{M}_g+\mathbf{M}_c+\mathbf{M}_m+\mathbf{M}_y \\
&= \frac{h}{24}\left(2(3d_0+d_h)+2(d_0+3d_h)+4(d_0+d_h)\right) \\
&= \frac{h}{24}\left(12d_0+12d_h\right) \\
&= \frac{h}{2}\left(sx_0+t+s(x_0+h)+t\right) \\
&= \frac{h}{2}\left(hs+2(sx_0+t)\right) = \overset{\square}{\mathbf{M}}_B \qquad \blacksquare
\end{aligned}
\tag{6.D.8}
$$

Masses of the individual tetrahedra (Equation (6.D.7)) summed up (Equation (6.D.8)) prove equality to the integrated box-mass (Equation (6.D.5)).

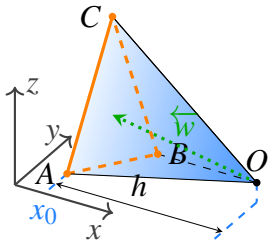
6.D.3 Center of Mass

In Equation (6.D.6) the center of mass integral results as a scaled vector $\overrightarrow{w}$, parallel to the x-axis and scaled dependent on the density function. However, the center of mass for a general tetrahedron is formulated in Equation (6.2) solely based on its vertices. To eventually express the equality of the box- and tetrahedralized solutions, we first reformulate the center of mass for tetrahedra in a similar $\overrightarrow{w}$ vector-dependent way. Therefore, we establish the three basic integrable tetrahedra cases shown in Figure 6.D.5.



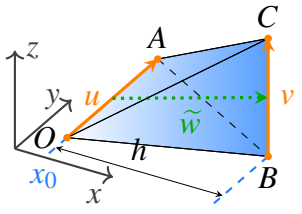
forward $\overrightarrow{\mathbf{T}}$

$$\overrightarrow{\mathbf{C}}\overrightarrow{\mathbf{M}} = O\overrightarrow{\mathbf{M}} + \int_{x_0}^{x_0+h} \overline{ABC}\,\tau^2 \cdot \overrightarrow{w}\,\tau \cdot (s\cdot x + t)\,dx$$
$$= O\overrightarrow{\mathbf{M}} + 3\overrightarrow{w} \cdot \overline{ABC}\frac{h}{3} \cdot \frac{4hs + 5(sx_0 + t)}{20}$$



backward $\overleftarrow{\mathbf{T}}$

$$\overleftarrow{\mathbf{C}}\overleftarrow{\mathbf{M}} = O\overleftarrow{\mathbf{M}} + \int_{x_0}^{x_0+h} \overline{ABC}\,(1-\tau)^2 \cdot \overleftarrow{w}\,(1-\tau) \cdot (s\cdot x + t)\,dx$$
$$= O\overleftarrow{\mathbf{M}} + 3\overleftarrow{w} \cdot \overline{ABC}\frac{h}{3} \cdot \frac{hs + 5(sx_0 + t)}{20}$$



edge $\widetilde{\mathbf{T}}$

$$\widetilde{\mathbf{C}}\widetilde{\mathbf{M}} = \left(O + \frac{u}{2}\right)\widetilde{\mathbf{M}} + \int_{x_0}^{x_0+h} |(1-\tau)\,u \times \tau v| \cdot \widetilde{w}\tau \cdot (s\cdot x + t)\,dx$$
$$= \left(O + \frac{u}{2}\right)\widetilde{\mathbf{M}} + \widetilde{w} \cdot |u \times v|\frac{h}{6} \cdot \frac{3hs + 5(sx_0 + t)}{10}$$

Figure 6.D.5: The three basis classes of *integrable* tetrahedra, exemplary with the x-axis as the integration domain. The density gradient over the extent of h is visualized in blue. Faces and edges orthogonal to the integration dimension, are highlighted in orange. For them the density is constant. The green arrows indicate the center of mass vectors $\overrightarrow{w}$, $\overleftarrow{w}$ and $\widetilde{w}$. Equations are formulated using $\tau = \frac{x-x_0}{h}$.

$$
\begin{aligned}
\mathbf{C}_r\mathbf{M}_r &= B\mathbf{M}_r + 3\overleftarrow{w}_r \cdot \overline{ADE}\frac{h}{3}\frac{hs+5(sx_0+t)}{20} \\
&= B\mathbf{M}_r + 3\overleftarrow{w}_r \cdot \frac{h}{120}(hs+5(sx_0+t)) \\
\mathbf{C}_g\mathbf{M}_g &= G\mathbf{M}_g + 3\overleftarrow{w}_g \cdot \overline{DHE}\frac{h}{3}\frac{hs+5(sx_0+t)}{20} \\
&= G\mathbf{M}_g + 3\overleftarrow{w}_g \cdot \frac{h}{120}(hs+5(sx_0+t)) \\
\mathbf{C}_c\mathbf{M}_c &= E\mathbf{M}_c + 3\overrightarrow{w}_c \cdot \overline{BFG}\frac{h}{3}\frac{4hs+5(sx_0+t)}{20} \\
&= E\mathbf{M}_c + 3\overrightarrow{w}_c \cdot \frac{h}{120}(4hs+5(sx_0+t)) \\
\mathbf{C}_m\mathbf{M}_m &= D\mathbf{M}_m + 3\overrightarrow{w}_m \cdot \overline{BCG}\frac{h}{3}\frac{4hs+5(sx_0+t)}{20} \\
&= D\mathbf{M}_m + 3\overrightarrow{w}_m \cdot \frac{h}{120}(4hs+5(sx_0+t)) \\
\mathbf{C}_y\mathbf{M}_y &= \frac{D+E}{2}\mathbf{M}_y + \widetilde{w}_y \cdot |u\times v|\frac{h}{6}\frac{3hs+5(sx_0+t)}{10} \\
&= \frac{D+E}{2}\mathbf{M}_y + \widetilde{w}_y \cdot \frac{4h}{120}(3hs+5(sx_0+t))
\end{aligned}
\tag{6.D.9}
$$

We approach the center of mass analogously to the mass itself and first establish the component-wise results for the individual tetrahedra in Equation (6.D.9), however, using the vector based formulations expressed in Figure 6.D.5.

$$
\begin{aligned}
\overset{\triangle}{\mathbf{C}}_B &= \frac{1}{\overset{\triangle}{\mathbf{M}}_B}\left(\mathbf{C}_r\mathbf{M}_r + \mathbf{C}_g\mathbf{M}_g + \mathbf{C}_c\mathbf{M}_c + \mathbf{C}_m\mathbf{M}_m + \mathbf{C}_y\mathbf{M}_y\right) \\
&= \frac{1}{\overset{\triangle}{\mathbf{M}}_B}\begin{pmatrix} & \mathbf{M}_{rg}(B+G) + (3\overleftarrow{w}_r + 3\overleftarrow{w}_g)\cdot\frac{h}{120}(hs+5d_0) \\ + & \mathbf{M}_{cm}(E+D) + (3\overrightarrow{w}_c + 3\overrightarrow{w}_m)\cdot\frac{h}{120}(4hs+5d_0) \\ + & \frac{D+E}{2}\mathbf{M}_y + \widetilde{w}_y\cdot\frac{4h}{120}(3hs+5d_0) \end{pmatrix} \\
&= \frac{1}{\overset{\triangle}{\mathbf{M}}_B}\begin{pmatrix} & \mathbf{M}_{rg}\cdot 2\overrightarrow{w} - 6\overrightarrow{w}\cdot\frac{h}{120}(hs+5d_0) \\ + & \mathbf{M}_{cm}0 + 6\overrightarrow{w}\cdot\frac{h}{120}(4hs+5d_0) \\ + & \frac{0}{2}\mathbf{M}_y + \overrightarrow{w}\cdot\frac{4h}{120}(3hs+5d_0) \end{pmatrix} \\
&= \frac{1}{\overset{\triangle}{\mathbf{M}}_B}\overrightarrow{w}\begin{pmatrix} & 2\mathbf{M}_{rg} - 6\frac{h}{120}(hs+5d_0) \\ + & 6\frac{h}{120}(4hs+5d_0) \\ + & 4\frac{h}{120}(3hs+5d_0) \end{pmatrix} \\
&= \frac{1}{\overset{\triangle}{\mathbf{M}}_B}\overrightarrow{w}\begin{pmatrix} & 5\frac{h}{60}(3d_0+d_h) - 3\frac{h}{60}(hs+5d_0) \\ + & 3\frac{h}{60}(4hs+5d_0) \\ + & 2\frac{h}{60}(3hs+5d_0) \end{pmatrix} \\
&= \overrightarrow{w}\frac{1}{\frac{h}{24}(12d_0+12d_h)}\frac{h}{60}(25d_0+5d_h+15hs) \\
&= \overrightarrow{w}\frac{25d_0+5d_h+15hs}{30d_0+30d_h} \\
&= \overrightarrow{w}\frac{25(sx_0+t)+5(s(x_0+h)+t)+15hs}{30(sx_0+t)+30(s(x_0+h)+t)} \\
&= \overrightarrow{w}\frac{2hs+3(sx_0+t)}{3hs+6(sx_0+t)} = \overset{\square}{\mathbf{C}}_B \qquad \blacksquare
\end{aligned}
\tag{6.D.10}
$$

In Equation (6.D.10) we sum up individual results from Equation (6.D.9) and compact the term. To do so, we utilize the properties of $\overleftarrow{\mathbf{M}}_r = \overleftarrow{\mathbf{M}}_g$ and $\overrightarrow{\mathbf{M}}_c = \overrightarrow{\mathbf{M}}_m$, respectively. Furthermore, we can combine mass center w-vectors and express them with the axis-aligned box-vector $\overrightarrow{w}$. The solution again proves equality to the box's mass center derived in Equation (6.D.6).

Chapter 7

Be Water my Friend: Mesh Assimilation

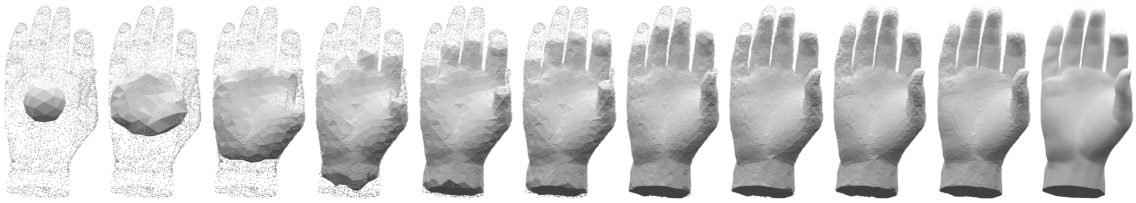


Figure 7.1: Progress of our method: Starting with a low-poly icosphere as init mesh, the surface grows within the target point cloud. If first approximates the rough shape and further progresses by assimilating and refining to reconstruct fine detail.

7.1 Abstract

Inspired by the ability of water to assimilate any shape, if being poured into it, regardless if flat, round, sharp or pointy, we present a novel, high-quality meshing method. Our algorithm creates a triangulated mesh, which automatically refines where necessary and accurately aligns to any target, given as mesh, point cloud or volumetric function. The technique is, furthermore, also suitable to improve upon existing reconstructions, such as the quad-meshes from Chapter 4 or the quad-dominant hulls from Chapter 5. Our core optimization iterates over steps for mesh uniformity, point cloud projection and mesh topology corrections, always guaranteeing mesh integrity and ε-close surface reconstructions. In contrast to similar approaches, our simple algorithm operates on an individual vertex basis. This allows for automated and seamless transitions between the optimization phases for rough shape approximation and fine detail reconstruction. Therefore, our proposed algorithm equals established techniques in terms of accuracy and robustness but supersedes them in terms of simplicity and better feature reconstruction, all controlled by a single parameter, the intended edge-length. Due to the overall increased versatility of input scenarios and assimilation robustness, our technique furthermore generalizes multiple established approaches such as ballooning or shrink wrapping.

7.2 Introduction

Without the need for mappings, parameterizations or specifically trained models, our approach presents a simple yet powerful extension in the field of geometric surface reconstruction techniques. We demonstrate how our algorithm can top similar state-of-the-art approaches; e.g., when reconstructing fine details and sharp feature edges in terms of mesh uniformity or versatility of supported input data. Our method is based on the concept of water being quite formless and able to assimilate any shape if being poured into it. Starting from a small initial triangle mesh, the object can grow, shrink and locally refine to assimilate a target shape with controllable precision. Vertices of the initial mesh extend rather autonomously along surface normals until they reach the target hull space. The expanding surface mesh is supplemented with suitably interpolated vertices where it is stretched the most. In contrast to the surface tension of water, the inter-vertex energy optimization promotes mesh uniformity but without enforcing smoothness priors, often found in other ballooning concepts. Once the vertices are close enough to the target hull, they individually transition from mesh growth mode to a projection scheme. Due to the bilateral weighting concept of the projection, vertices are effectively pulled into intersecting tangent planes of the hull, thus allowing for accurate reconstructions of small detail and sharp edges. The optimization of vertices is an iteration process, but individual for each vertex and can therefore be executed on the GPU as a massively parallelized operation. Per design, the algorithm assimilates to shape rather than a strict type of input data, which can be given as mesh, point cloud, signed distance or other volumetric function. With its frugal input requirements and minimal parameter space, our method simplifies the rather complex nature of other established techniques; it is situated somewhere between the exhaustively studied fields of shape approximation [CSAD04], surface remeshing [AUGA08] and point cloud reconstruction [BTS*17].

7.2.1 Contributions

Our approach fulfills all criteria which one would ask of a state-of-the-art reconstruction technique: Guarantee for closed manifold meshes, the ability to cope with varying input density, robustness to noise and outliers as well as to missing data and control over higher genus topology. We can summarize our main contributions: • **Mesh uniformity** is achieved as our mesh unravels with an inter-vertex energy, striving to equalize distances between vertices. • **Adaptive resolution** is realized by locally adapting the target-edge-length to the provided sample density. • **Sharp features** are reconstructed, as the second vertex optimization step is designed to minimize projected distances to surface tangents, thus the vertices snap on close-by edges and corners. This is where most state-of-the-art methods still lack precision. • **Arbitrary initialization** meshes are possible, as demonstrated with various examples. • **Versatile input** data is supported, as our method is not bound to point clouds and can easily handle meshes as well as volume data and signed distance fields, currently explored in neural surface representations [MON*19, PFS*19].

7.2.2 Related Work

While the ideas of ballooning or shrink wrapping are not exactly new [DQ01], we are yet to see its full potential. In some aspects, our method follows similar principles like the concept of Competing Fronts (CF) [SLS*06], which is also a coarse-to-fine point cloud reconstruction technique, using growing mesh geometry. But instead of whole mesh fronts, which get frozen at some point, our method operates on an individual vertex basis so that the mesh remains flexible during the optimization. This allows for smoothing out irregularities without the need for remeshing in every other iteration. Our hull projection scheme furthermore resolves the issue of reconstructing creases and sharp feature edges, an open problem for CF. Unfortunately, a direct comparison was not possible as there is no reference implementation or result data available. Nevertheless, the concept gained some recent attention with the concept of Cooperative Evolutions (CE) [LL20], combining two mesh fronts enclosing on the point cloud from the in- and outside in parallel. A first draft utilizing this idea in a learning-based approach was proposed with Point2Mesh (P2M) [HMGCO20]. A thorough analysis of the actual distinctions of our approach to other ballooning concepts is featured in our discussion in Section 7.5.2.

As previously introduced in Section 3.2, the Screened Poisson Surface Reconstruction (SPSR) [KBH06, KH13, KH19, KCRH20] is probably one of the most commonly used reconstruction methods but conceptually quite different from ours. In direct comparison, however, SPSR tends to produce rather smoothed out results, whereas ours generally appear sharper and capture finer details, even at lower resolution. Due to the octree-nature of the SPSR approach, the resulting triangulation is quite inhomogeneous, while ours approaches both a uniform distribution and Delaunay-like connectivity. Scale Space Meshing (SSM) [DMSL11] really takes advantage of modern high precision 3D scanners and is able to faithfully reconstruct even very fine details. However, as our comparison shows, sharp edges are not captured very well. The marching cubes algorithm [LC87] is often used in a medical context to reconstruct surfaces from volumetric data or to remesh existing surfaces [Ble20]. Our algorithm is able to handle both as it can be used on signed distance fields (SDF) as well as on existing meshes with the advantage of producing more evenly distributed vertices and triangle faces. Both SPSR and Marching Cubes are able to recover surfaces but fall short in terms of triangle mesh homogeneity, a key contribution of our approach. Similar deficiencies can be attributed to α-shape- [EM94], Voronoi-based- [ABK98, AB99, ACK01, BL17] or the ball-pivoting-algorithm (BPA) [BMR*99]. For these algorithms, the achievable mesh geometry-, and sometimes topology-, quality crucially depend on the density and uniformity of the given input points and sometimes require prior outlier-filtering or noise-smoothing steps. Our approach is quite invariant to these artifacts, as it is able to cope with noise and can either ignore or incorporate varying sample density. Most recently, learning-based approaches also joined the reconstruction game [MON*19, PFS*19] but often require specifically trained models on an explicit object class. This is no longer a requirement with P2M, which reconstructs point

clouds via shrink-wrapping and basically learns from the input itself. But their results also lack precision in corners and sharp object features. Section 4.6.3 already addressed quad-mesh reconstruction such as Instant Field Aligned Meshes (IFAM) [JTPSH15] or Online Surface Reconstruction (OSR) [STJ*17] in a comparison to our quad-meshing technique. Both external procedures produce nice feature-aligned mesh structures but often struggle with low or varying sample density, as demonstrated in comparisons to our results. Normal information is a quite common requirement for reconstruction techniques, except for our quad-meshing method, which operates on unoriented samples. The proposed assimilation technique also takes full advantage of the implicit tangent space when provided with normals to increase reconstruction accuracy.

7.3 Method

As illustrated in Figure 7.1, the goal of our algorithm is to assimilate to a target shape, starting from a simple initialization mesh. Therefore, the basis mesh first approximates the rough shape of the input, and finer details are recovered as the optimization continues. This assimilation process follows a simple optimization scheme, iterating over the following five steps, performed in one iteration cycle:

I Subdivide the mesh where necessary

II Equalize inter-vertex distances

III Mesh expansion and hull projection

IV Fix mesh irregularities

V Add tunnels to increase genus (optional)

Termination criteria for the optimization can be specified, either as a target-edge-length l_t for the final mesh or an ε-closeness threshold to the target shape. Each operation is self-contained, meaning the surface remains a closed manifold and valid mesh at all times. Related coarse-to-fine approaches [SLS*06, LL20, HMGCO20] separate the *mesh development* and *final fitting* phase in their procedure or interleave the assimilation progress with remeshing operations. However, the strength of our method comes with the mixed design of the assimilation (III): Simple mesh growth seamlessly transitions into a projection scheme on an individual vertex basis.

Our method is first introduced with a focus on point cloud reconstruction. The adaptation to other input scenarios is later addressed in Section 7.4. In the following we associate *vertices* v_i with the growing mesh structure and *samples* $\dot{s}$ with the input point cloud. Variables related to samples are denoted with an over-set dot. Further, we observe local 1-ring neighborhoods around vertices: For a vertex v_i, adjacent mesh neighbors are grouped in N_i where $k_i = |N_i|$. The average edge length around vertex v_i is formally expressed in Equation (7.1).

$$l_i = \frac{1}{k_i} \sum_{v_j \in N_i} |v_i - v_j| \tag{7.1}$$

Vertex normals n_i are derived from the face normals in this 1-ring neighborhood as described in Section 2.2.1: The normalized portions of corner angles in the triangle fan are used to weigh their normals' contribution, respectively. Further, each vertex v_i is associated with a set S_i of 10 closest point cloud samples. This is cheap to initialize with a brute-forced search as the initial mesh only has very few vertices. To update the existing sets or create new ones for added vertices (I) we then only query the sample sets in the vertices' 1-ring neighborhoods for potential new nearest neighbors. Further, each sample is interpreted as a splat-disc on the implicit hull, represented by a normal $\dot{n}$ and a radius $\dot{r}$. We compute the radius of a sample as half the median distance to its 10 closest point cloud neighbors. While all our results are generated using this simple approach, more advanced splat methods [WK04, CGAY13] may be suitable as well.

I Refinement

The triangle mesh is constantly refined with targeted $\sqrt{3}$-subdivision operations [Kob00]. A vertex qualifies for refinement when its average length l_i of incident edges is larger than the specified target-edge-length l_t. Or, if ε-closeness is specified as the optimization criterion, simply the vertices with the largest l_i get subsequently refined. As a result, faces around a vertex are replaced as shown in Figure 7.2: For each triangle in this 1-ring fan, a new vertex is created along with three new triangles. The new vertices are the interpolated center points of the fan triangles, interpreted as curved point-normal (PN) triangles [VPBM01] using vertex normals. This allows us to maintain local curvature and the subdivided geometry smoothly integrates with the surrounding mesh as the outer 1-ring loop edges are not split. This operation increases the valence of the 1-ring neighbor vertices by 1 and may create very obtuse or pointy triangles. Nevertheless, both issues are fixed within the same cycle by upcoming steps (II, IV).

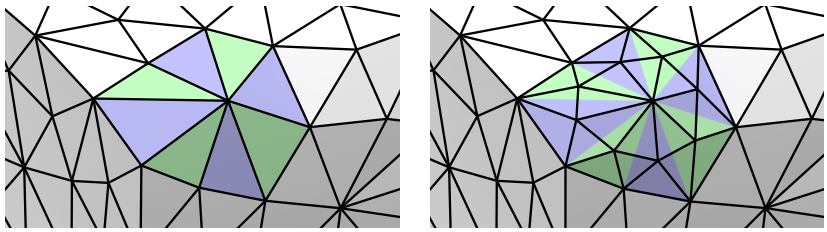


Figure 7.2: Mesh refinement: The 1-ring triangle fan around a vertex is $\sqrt{3}$-subdivided. New vertex positions are interpolated as PN triangle centers to maintain local curvature.

II Equalization

In order to obtain the most regular mesh structures with evenly distributed vertices, we employ a simple geometric optimization: Every vertex v_i strives to individually equalize the distances to its direct 1-ring neighbors $v_j \in N_i$, eventually resembling a Delaunay-like triangulation. Equation (7.2) formulates the update vectors, where $\vec{v}_{ij} = v_j - v_i$ and l_i is the average length of edges incident on v_i.

$$\vec{e}_i = \left(\frac{1}{k_i} \sum_{v_j \in N_i} v_j - l_i \frac{\vec{v}_{ij}}{|\vec{v}_{ij}|} \right) - v_i \tag{7.2}$$

These geometric increments mainly smooth out dense clusters created by step I and can be applied with a weighted update on vertex positions as $v_i := v_i + \lambda_e \vec{e}_i$. Thus, λ_e allows adjusting the progression speed for mesh homogenization. We achieved the best results with a soft update, using $\lambda_e = 0.1$.

III Assimilation

This step aims to pull vertices to the surface and snap vertices on edges and corners. Once our mesh is close enough to the point cloud hull, a suitable projection technique is applied. However, when starting from a generic base mesh, e.g., a simple sphere inside of, or enclosing, the point cloud, it first has to grow or shrink into a rough approximation of the target shape so that the projection may take over later. Whereas this is how it is usually done, we propose a mixed procedure, adaptive to each individual vertex. This step modifies the mesh's geometry by moving vertices with individual update vectors. These vectors $\vec{a}_i$ may embody surface projection (III.A) or mesh growth (III.B). The term in Equation (7.4) decides which one is to choose, dependent on an individual vertex v_i and its weights $\dot{w}_{ij}$, summed up in Equation (7.3).

$$\omega_i = \sum_{j \in S_i} \dot{w}_{ij} \tag{7.3}$$

$$\vec{a}_i = \begin{cases} \dfrac{1}{\omega_i} \displaystyle\sum_{j \in S_i} \dot{w}_{ij} \dot{\vec{v}}_{ij} & \text{III.A if } \omega_i > \varepsilon \\ n_i \cdot \dfrac{o_i l_i}{4} & \text{III.B else} \end{cases} \tag{7.4}$$

III.A Projection

For valid weights $\dot{w}_{ij}$, gathered from the point cloud samples associated with vertex v_i, the update vector $\vec{a}_i$ is the weighted sum of v_i's projection vectors $\dot{\vec{v}}_{ij}$ into the samples' tangent planes (Figure 7.3, green). Projected points compute as $\dot{v}_{ij} = v_i + \dot{\vec{v}}_{ij}$ with vectors

$\dot{\vec{v}}_{ij} = \dot{n}_j \cdot (\dot{s}_j - v_i)\dot{n}_j$ where $\dot{n}_j$ is the normal of point cloud sample $\dot{s}_j$. The bilateral weights $\dot{w}_{ij}$ combine a distance and an angular component. This gives us the advantage of only pulling qualified vertices into edges and corners while their direct neighbors reside in flat regions nearby. As meaningful distance measures we use the closest points (Figure 7.3, blue) on the individual splat-discs respectively, as formulated in Equation (7.5), where $\dot{r}_j$ is the radius of sample $\dot{s}_j$. Further, the actual distance weight is the result of a Gaussian, centered on $\mu = 0$ with $\sigma = \frac{l_i}{3}$. The angular component computes as the dot-product of vertex and sample normal, clipped to values ≥ 0. As formulated in Equation (7.6), $\dot{w}_{ij}$ is simply the product of both components.

$$\dot{d}_{ij} = \begin{cases} \left| v_i - \dot{s}_j + \frac{\dot{v}_{ij} - \dot{s}_j}{|\dot{v}_{ij} - \dot{s}_j|} \cdot \dot{r}_j \right| & \text{if } |\dot{v}_{ij} - \dot{s}_j| > \dot{r}_j \\ \left| \dot{\vec{v}}_{ij} \right| & \text{else} \end{cases} \tag{7.5}$$

$$\dot{w}_{ij} = \exp\left(-\frac{9\dot{d}_{ij}^2}{2l_i^2} \right) \cdot \max\left(0, n_i \cdot \dot{n}_j\right) \tag{7.6}$$

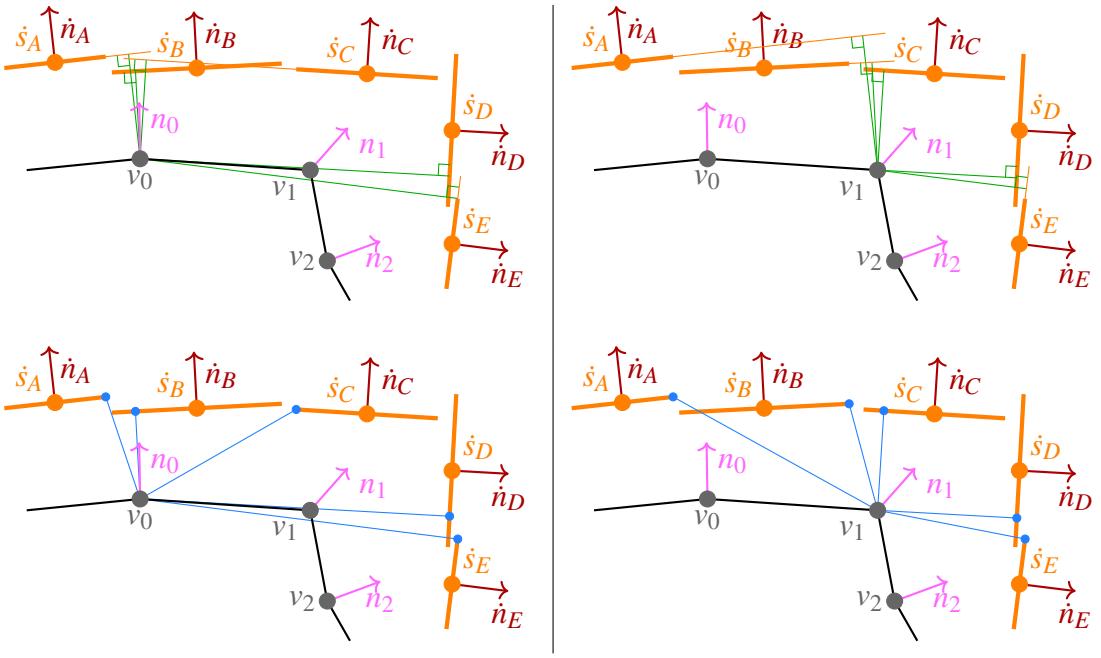


Figure 7.3: Point cloud samples $\dot{s}_i$ are shown with their splat discs and normals $\dot{n}_i$. The example shows projections for vertices v_0 (left) and v_1 (right) as well as closest points on splat discs, respectively. Vertices moving along the projection direction, weighted by distance and the angle between vertex normal and sample normal, and are pulled into the intersection of tangent planes, thus allow for better edge preservation.

Figure 7.3 visualizes the strength of our proposed concept. The contribution of the force that pulls vertices into tangent planes scales down with increasing distance from its originator. However, the multiplication with the angular component is what really makes the difference when it comes to recovering more surface detail.: Vertex v_0 lies on a flat part of the mesh, the contribution of samples *just around the corner* ($\dot{s}_D$, $\dot{s}_E$) is decreased significantly or even canceled out completely. For a vertex residing in a curved neighborhood or on an edge like v_1, the attraction of samples from both sides of the implicit edge ($\dot{s}_B$, $\dot{s}_C$ & $\dot{s}_D$, $\dot{s}_E$) effectively pull it evenly into the intersection of their tangent planes. Even if the splat-discs themselves do not intersect, this is guaranteed because only the projection directions contribute to the update vectors.

III.B Growth

If the weights summed up in ω_i are too small, the vertex v_i is not yet sufficiently close to the point cloud hull. Alternatively, the displacement vector $\vec{v}_i$ is derived from the vertices' normal n_i, and scaled by $\frac{l_i}{4}$. This very robustly ties the geometric increment of a vertex to the scale of its local mesh neighborhood. Further, the normal is oriented with a value $o_i = \pm 1$, which computes as the combined majority sign of all dot-products between normal n_i and projection vectors $\dot{\vec{v}}_{ij}$. This allows for mesh expansion as shown in Figure 7.1, when the mesh is within the target shape, but also for shrinkage as in Figure 7.6 (top), with the mesh enclosing the target shape. Further robustness is exemplified in Figure 7.A.2 (top) with an out-of-target initialization mesh, which first gets drawn into the target and then starts to grow. Again, vertices are displaced using a differential update $v_i := v_i + \lambda_a \vec{a}_i$, where λ_a allows to adjust the mesh's assimilation speed. However, as the differential increment is by design also linked to dimensions of the local mesh neighborhood, a large λ_a could provoke single vertices to grow out too fast. A moderate growth rate of $\lambda_a = 0.1$, allows step II to level out length discrepancies before the next assimilation update, thus promotes a more even and homogeneous mesh expansion.

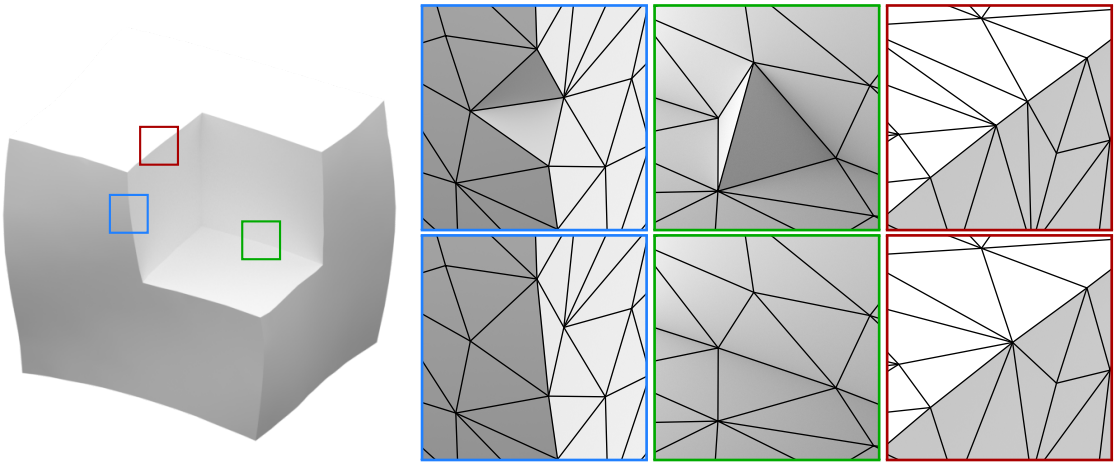


Figure 7.4: Improvement operations: The concave and convex edges (standing out in contrast to their neighbors) are fixed by rotation. A short edge is collapsed to one vertex.

IV Improvements

For retaining a mesh of high quality, unfavorable edge constellations are detected, and fixed [DQ01] on-the-fly: Concave or convex edges, which stand out in contrast to their direct triangle opposites, are fixed with a simple rotation. Figure 7.4 shows examples of these cases and how they are resolved. Edges shorter than a certain length (e.g., $\frac{l_t}{10}$) are collapsed, merging two vertices into one. This might occur after the edge was rotated or if two neighboring vertices are drawn onto the same implicit edge and get too close to each other. As this step proactively prevents local foldovers, our results do not feature manifold violation artifacts.

V Object Topology Adaptation

In the most basic scenario, the mesh assimilation is initialized on a simple spherical mesh with a target of the same genus 0. In order to support reconstructions of objects with higher genus, our algorithm supports incorporating tunneling operation steps, automatically adapting the mesh's topology. Analogously to *Freestyle* [SCC11], this step prevents opposing fronts from intersecting each other. Simple proximity tests on moving vertices trigger the replacement of two opposing 1-ring neighborhoods around close vertices or triangles with an intra-mesh tunnel. However, as the initial mesh might be quite arbitrary, tunnels can also resolve inter-mesh situations as shown in Figure 7.21 (center) and Figure 7.A.2 (bottom), when initialized with separated geometry. As shown in Figure 7.5, the operation is not bound to surface orientation and may also form inverted tunnels, resulting in the correct genus nevertheless. Another example is shown in Figure 7.A.1 where multiple intra-mesh tunnels are required to accommodate the target's genus. This step is optional and may be omitted if the target object's genus is known to be 0 beforehand.

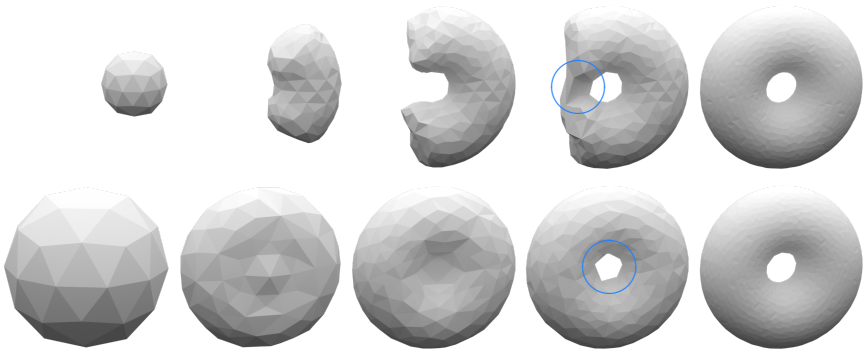


Figure 7.5: Adapting the topology. Tunneling operations are invariant of the surface orientation and automatically apply for *ballooning* (top) and *shrink wrapping* (bottom).

7.4 Generalized Input

To be able to mesh more than just point clouds by assimilation, we actually only have to adapt step III of the optimization. This is demonstrated with meshed input, signed distance fields and volumetric data.

7.4.1 Remeshing

For the purpose of remeshing existing objects, e.g., to ensure hole-free reconstruction or uniform mesh quality is quite trivial to handle, as we can simply use actual surface triangles instead of splat discs. Distances, directions and weighting follow straightforward. Two examples using a meshed input as a target are shown in Figure 7.6. On top, the sphere shrink-wraps around the target shape and refines for a smooth and curved surface. For the bottom case, the target-edge-length is increased, which leads to an incremental collapse of short edges, and the mesh assimilates the coarse icosphere again.

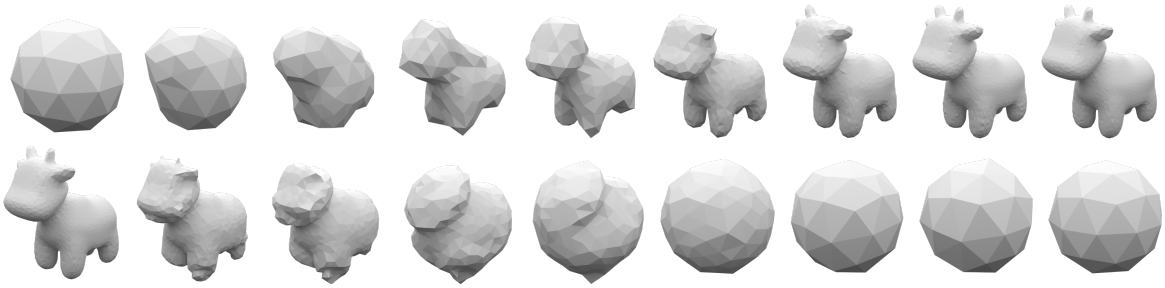


Figure 7.6: Top: Our algorithm assimilates the coarse icosphere to the target quad mesh [Cra20]. Bottom: Swapped input/target lead to a coarsening; with a larger target-edge-length, short edges incrementally collapse. Progress is left to right in both examples.

7.4.2 Signed Distance Fields

Our mesh construction also easily supports SDF input. Here one only has to replace the orientation o_i in Equation (7.4) with a signed-distance. It is sufficient to sample the field once for each vertex position, acquiring single scalar values respectively, as the vertex normal is used as movement direction. The update vectors $\vec{a}_i$ are always in mesh-growth mode (III.B), since there is no trivial tangent space given, as with splat discs. This has the drawback that there is no surface projection and, therefore, vertices are not automatically drawn to edges or corners. Nevertheless, smooth and curved surfaces are still reconstructed very accurately: Update vectors (Equation (7.4)) are automatically scaled down to 0 length as the signed-distance decreases, and therefore, the mesh closely approaches the implicit target hull. Figure 7.7 illustrates surface reconstructions of a target shape, given as a signed distance field.

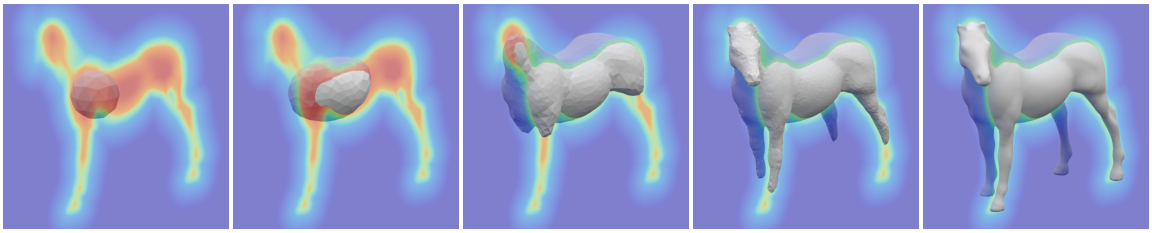


Figure 7.7: Mesh construction from a signed distance field, visualized as a color coded cross-section slice: outside, close to 0, inside. As there are no splat discs (point cloud tangents), the hull projection step is omitted.

7.4.3 General volumetric 3D data

The processing of general volumetric input, e.g., as acquired from computational tomography, requires a bit more effort but can be reconstructed as well. Figure 7.8 shows the reconstruction of a CT scanned neuron, given as sliced binary volume. Analogous to the signed distance field, there is again no trivial tangent space for hull projection. Orientations are not so easily determined as with the SDF, but nevertheless, they can be easily sampled and trilinearly interpolated from the volume at mesh vertex positions, with binary values indicating their sign. Update vectors are scaled down to zero length, once the mesh vertex is situated between both in- and outside labeled voxels, indicating that the mesh reached a surface. By construction, the result mesh is watertight. Segments of dendrites separated by noise in the scan data are not connected by our approach.

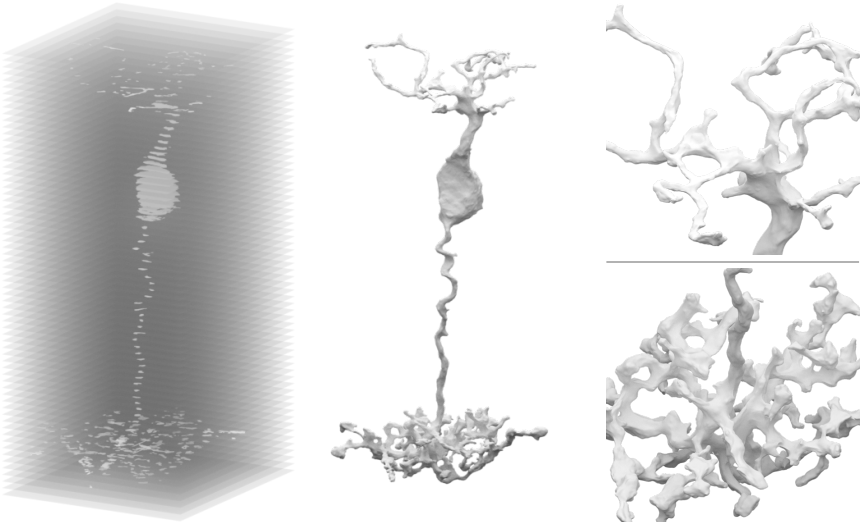


Figure 7.8: Tomography data [HBT*13]: A neuron, given as 1325 binary slices (not all shown) with 661×595 pixels each, reconstructed as surface mesh. Thanks to Christian Behrens for providing the data [BSH*16].

7.5 Experiments and Discussion

In this section, we experiment with the achievable mesh quality of our method, challenged with the five most common artifacts found in point cloud data, as described by Berger et al. [BTS*17]. A numerical comparison follows in Section 7.5.1 as well as a qualitative discussion in Section 7.5.3.

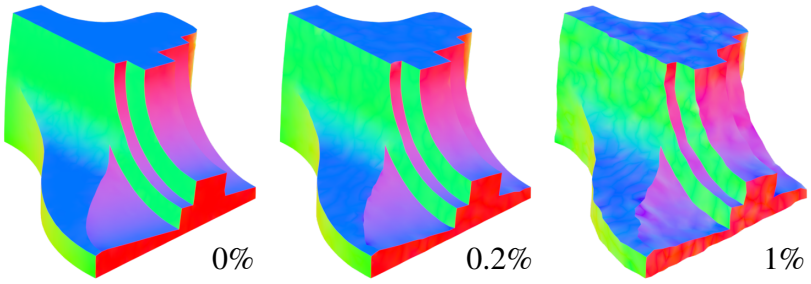
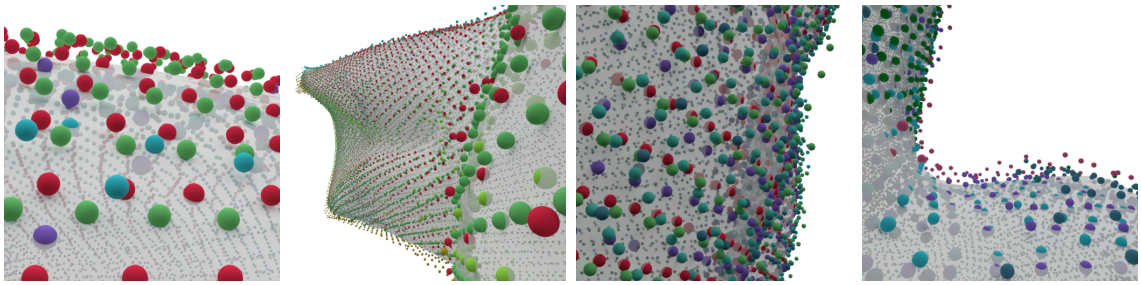


Figure 7.9: Mesh construction on noisy input. Besides some kinks, strong edges remain sharp even with increasing noise levels of randomly jittered samples. The noise magnitude is given in percent of the bounding box diagonal.

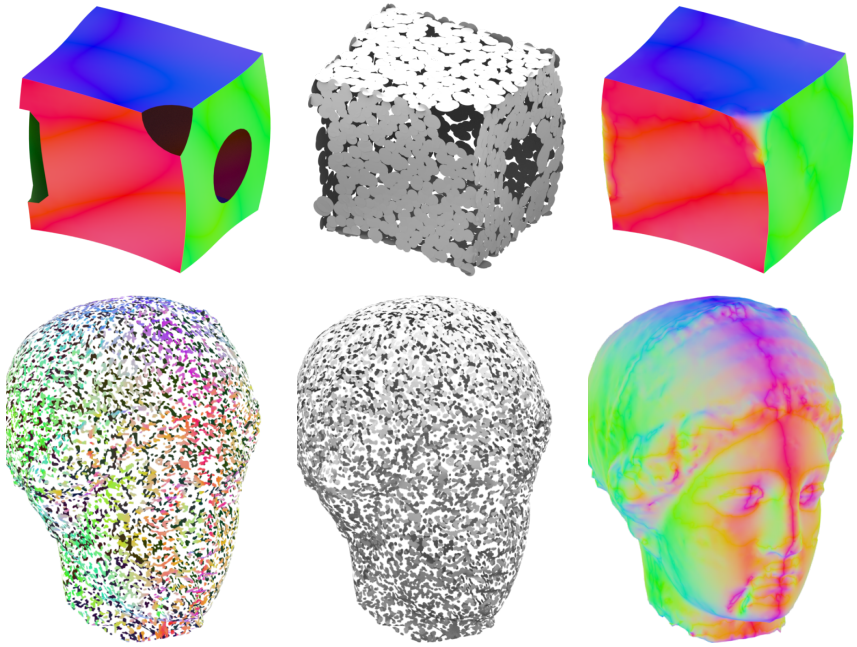
Noise and outliers are prominent artifacts in 3D scan data and can be robustly dealt with by our algorithm. Our algorithm is robust against prominent artifacts in 3D scan data, such as noise and outliers. Figure 7.9 shows results of the challenge designed for *robust moving least-squares fitting* [FCOS05]. A point cloud, sampled reasonably dense but randomly, is perturbed with increasing magnitudes of random spherical jitter noise, scaled by the *Fandisk*'s bounding box diagonal. Without noise, ground-truth-like results are achieved. At 0.2% noise scale [FCOS05], flat areas are still flat, curves smooth, edges sharp, and only a few kinks become noticeable on very obtuse edges. With noise even further increased by factor 5, our reconstruction still robustly captures all significant features of the object. Figure 7.10 shows details of the noisy overlap regions in a real 3D scan, along with the smoothly reconstructed surface. When not extracted in a prefiltering step, outliers do not cause any difficulties in our approach either: Mesh vertices are associated with a fixed number of nearest point cloud neighbors. As a result, far-out outliers are usually not part of any neighborhood. But if so anyway, their influence is still minimal as their contribution is weighted inversely by their distance.

Missing data in the input geometry is recovered by the reconstruction. The first example in Figure 7.11 shows the *Twistcube* with cutout geometry on a curved side area, an edge and on a corner. The reconstructed mesh smoothly interpolates the missing regions, stays within the implicit hull and does not grow through the holes. The second example shows *Igea*, where 75% of the source mesh was cut out and removed in randomly shaped patches. Our algorithm still reconstructs a closed mesh and recovers fine details.



head ear snout tail

Figure 7.10: Reconstruction on 3D scan data [TL14]. When aligned, the 10 individual partial scans (colored) create noisy overlap regions. As every mesh vertex is attracted by multiple sample tangents, the noise is leveled out and the surface is smooth, nevertheless.



Defective Source Splat Discs Reconstruction

Figure 7.11: Filling up holes: Results on defected and then sampled sources. The *Twistcube* is missing portions on a curved side region, an edge and a corner. 75% of *Igea*'s original surface was removed by cutting out random patches.

Sample density is not a decisive factor for successful reconstructions. While modern 3D scanners usually provide quite dense point clouds and more information always allows for more precise results, our algorithms splat discs also automatically adapt to fewer and sparser samples. Figure 7.12 shows the same object with different sampling but accurate results with smooth curves and sharp edges in both cases.

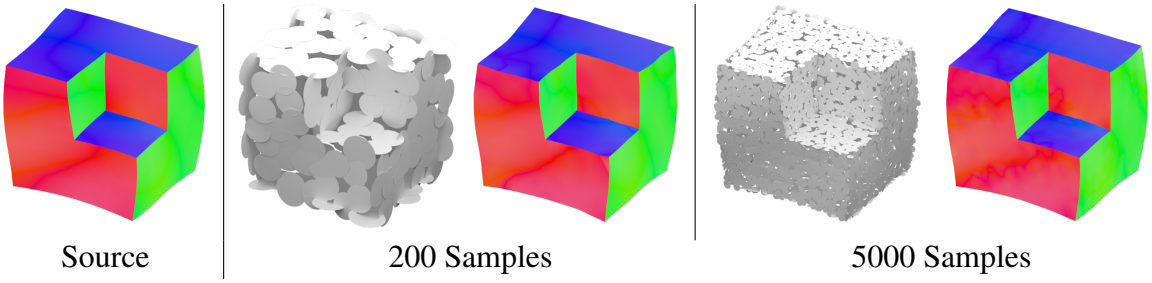


Figure 7.12: Sample density has little influence on the reconstruction accuracy as the disc radii automatically adapt, thus always provide a valid projection space.

One might argue that equidistantly distributed vertices - the default outcome of our algorithm - are not always a favorable choice for meshed objects. Consequently, our method allows to easily switch to a sample-density adaptive mode, as exemplified in Figure 7.13. For simulating varying density, the ground-truth mesh was importance-sampled, based on surface curvature. This leads to a higher sample density on the facial features and fewer, sparsely distributed samples on the head or neck region. Now we can locally scale the target-edge-length for each vertex individually, based on the radius of the closest splat discs. Figure 7.13 also shows the more challenging example, known from Figure 4.22, with a very steep gradient in sample density: The 10k samples on the sphere are separated by the equator line with 90% on the upper and 10% on the lower hemisphere. The resulting mesh adapts to this change in resolution with coarser geometry on the bottom and finer geometry on the top half.

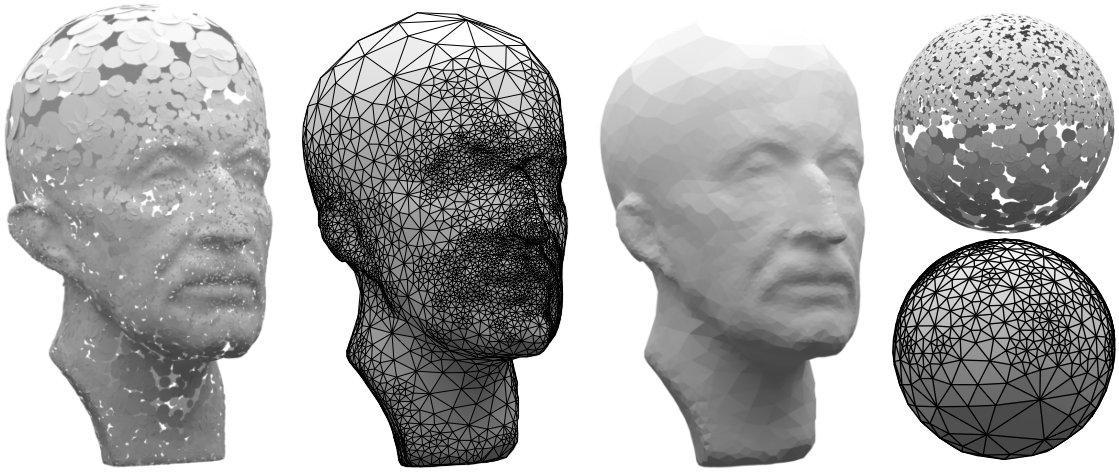


Figure 7.13: Adaptive meshing results: The *Max Planck* bust [LGS02] was importance-sampled based on local surface curvature. The reconstructed mesh adapts accordingly. The sphere features a very steep density gradient with 90% of the 10k samples located on the upper and 10% on the lower hemisphere. The leftmost bust and upper sphere show the input samples and their automatically size-adapted splat discs.

7.5.1 Comparisons

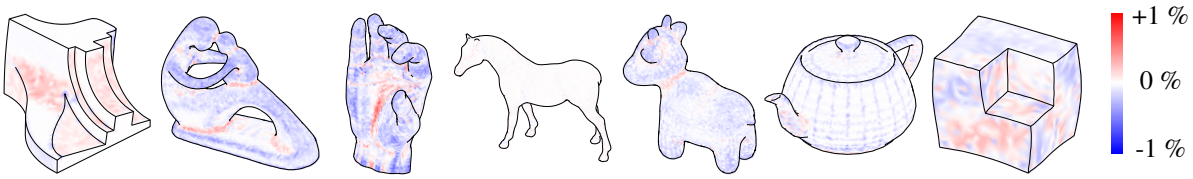
Besides validation against common point cloud challenges or the qualitative discussion in the upcoming section, we provide a numerical comparison to a variety of state-of-the-art reconstruction techniques evaluated on over 2000 randomly selected objects from the Thingi10K data set [ZJ16]. A comparison to the most closely related technique, *Competing Fronts* [SLS*06], is not included since there is no implementation available and the authors could not provide reference results. Ground truth objects are scaled down and centered to fit in a $[-1{:}1]$ cube. Oriented point clouds are sampled uniformly from the source objects, where the number of samples computes as object surface area $\times 1000$. Where possible, we specified $l_t \approx 0.025$, except for *IFAM* where we set the target vertex count to half the amount of the input point cloud samples to actually produce a decent result. Other parameters were left on auto- or the default settings, respectively. Total failure cases ($<5\%$), where the applications crashed and produced no output, could not be taken into account.

	HD		RMSE		SAD		TVD		NAD	WT
BPA	0.0572	0.0522	0.0027	0.0027	−0.0935	−0.0503	0.0653	−0.0148	0.1512	0.0000
SSM	0.0895	0.0909	0.0044	0.0040	−0.4705	−0.4252	0.0283	−0.4480	0.2201	0.0013
IFAM	0.1394	0.1294	0.0040	0.0040	−0.4130	−0.3714	0.1278	−0.3796	0.1972	0.0000
OSR	0.1241	0.1015	0.0078	0.0061	−0.1339	−0.0170	0.0258	−0.0013	0.2877	0.5171
SSDR	0.0295	0.0257	0.0019	0.0017	−0.0313	−0.0225	0.1154	−0.0033	0.0921	0.9317
SPSR	0.0350	0.0340	0.0026	0.0026	−0.0432	−0.0315	0.0110	0.0001	0.1280	0.9982
Ours	0.0406	0.0190	0.0013	0.0008	−0.0159	0.0009	0.0040	0.0006	0.0401	1.0000

Table 7.1: Results of multiple reconstruction techniques (BPA [BMR*99, Dig14], SSM [DMSL11, Dig15], IFAM [JTPSH15], OSR [STJ*17], SSDR [CT12], SPSR [KBH06, KH13, KH19, KCRH20] and ours), compared to the ground truth of over 2000 objects from the Thingi10K data set [ZJ16], given as mean and median. The errors (best close to 0, except WT best close to 1) include the symmetric Hausdorff-Distance (HD) measured with 200k samples, the root-mean-squared-error (RMSE) measuring the closest-point distance of result faces to the ground truth, the surface-area-deviation (SAD) and total-volume-deviation (TVD) comparing reconstructed objects against the ground truth, as well as the amount of watertight (WT) results, all given in percent, except for the normal-angle-deviation (NAD) which is given in radians. See Section 2.5 for more details.

Results of our numerical comparison are listed in Table 7.1 with means and medians of different error measures, respectively. In the average Hausdorff-Distance, our method is only outranked by two other approaches but supersedes (or is close to) the competition in all other measures. This slight drawback can be attributed to a limitation of our method, namely very, very thin geometry. Such a case is included in Figure 7.A.3 with object 1489590: With geometry becoming thinner than the target-edge-length, the mesh growth comes to a halt, as there are no further refinements. That portions of the point

cloud remain unexplored has the consequence of relatively high errors in the Hausdorff-distance metric. A heuristic indicating how many point cloud samples are actually in use for hull projections could potentially improve on this matter. The other models in Figure 7.A.3 and 7.A.4 exemplify, why some procedures fall behind in our numerical comparison: While flat or curved surfaces are often reconstructed quite well, edges on feature lines or sharp geometry are smoothed out. This especially shows on CAD-like models with sharp edges, for example on object 200080, where our result is the only one that accurately reconstructed the threads on the screw.



4.5017	4.9807	4.7738	10.9946	4.0569	3.4003	2.7820	HD
0.1928	0.2498	0.2202	0.3515	0.2505	0.2449	0.3660	RMSE
0.0561	0.1815	−0.4322	−7.8373	1.0409	−1.4329	−0.8017	SAD
0.2853	−1.2988	−0.2010	−3.2690	1.1300	−0.2009	0.0260	TVD

Figure 7.14: Approximation error: Objects are scaled to fit in a unit-cube; deviations from sampled ground truth meshes visualized in percent; symmetric Hausdorff-Distances (HD) measured with 200k random samples; RMSEs compute as distances of mesh triangle centers to the ground truth; surface-area-deviation (SAD) and total-volume-deviation (TVD); values multiplied by 100. Projections on tangent planes cause vertices to exaggerate on concave (vertex inside) and convex (vertex outside) curved regions. This is not as prominent on the *Horse* as it is reconstructed from an SDF without hull projections.

7.5.2 Other Ballooning Methods

Whereas the numerical comparisons are mostly focused on established state-of-the-art meshing procedures, we also want to point out clear conceptional distinctions between our and related ballooning approaches.

• **Competing Fronts** (CF) [SLS*06] also uses a deformable mesh complex, iteratively approaching the target shape. However, CF strictly grows from within the target as vertices only move along positive normal directions and get frozen once close enough to the point cloud. Our mesh remains flexible throughout the optimization as vertices move and align individually. Further, our method generalizes ballooning and shrink-wrapping, as the mesh may grow from within the target but also enclose on it from the outside. As shown in Figure 7.A.2, our mesh's growth direction is robust enough to even overcome a misaligned initialization. In order to maintain high-quality mesh structures, CF employs remeshing operations [BK04] every 6-18 iteration cycles. In our iteration, corrective operations are performed in every cycle but targeted at the affected mesh regions. The increase of an objects genus is approached in a similar defensive manner, as both methods replace opposing geometry with a triangulated tunnel before the mesh self-intersects. See step V of Section 7.3 for more details. CF finalizes the optimization with a dedicated projection step to guarantee ε-close reconstruction results, which, however, is invariant of the point cloud tangent space, thus does not recover any sharp features. Our optimization is designed to automatically transition from mesh growth to a projection scheme on an individual vertex basis. Further, in our projection, the vertices are explicitly drawn towards intersections of tangent planes, thus faithfully reconstruct sharp corners and edges.

• **Cooperative Evolutions** (CE) [LL20] is based on *two* mesh structures, approaching the target shape from the interior *and* exterior simultaneously. Once both hulls are close enough, individual vertices from both hulls have to be matched to each other, and the final geometry is extracted from the space between both meshes. Objects of a higher genus are also possible: Self-intersections are identified, the affected geometry is removed, again stitched together, smoothed and refined. The mesh evolution in CE is based on a signed distance field, recomputed every iteration cycle and mixed with a normal-distance field. In contrast, our mesh vertices are linked to individual, fixed-size sets of nearest samples, easily updated with sets from their 1-ring neighbor vertices, respectively. Mesh integrity in CE is, as in CF, maintained with dedicated global remeshing [BK04]. Further, CE is invariant of the point cloud orientations, thus solely relies on the distance fields. As shown in our SDF experiments (Section 7.4), our method can also easily operate on distance fields. However, this also comes with the same drawbacks and limitations of CE, as our method is then also no longer able to recover sharp edges.

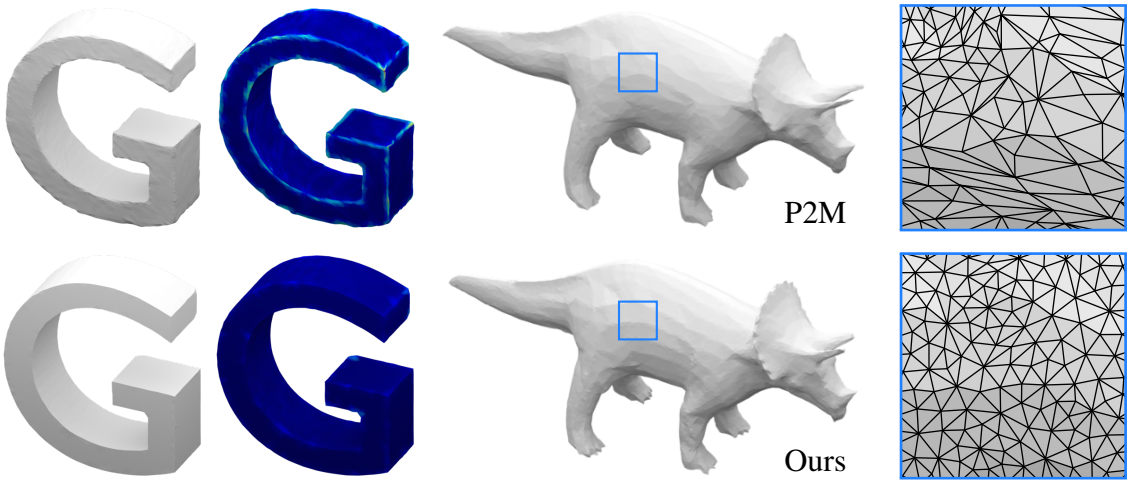


Figure 7.15: Comparing results of the learning based state-of-the-art approach Point2Mesh (P2M) [HMGCO20] to ours.

• **Point2Mesh** (P2M) [HMGCO20] is a state-of-the-art learning based approach, iteratively shrink-wrapping a convex hull mesh around a target point cloud. The individual vertex displacements are the results of a learning self-prior network, which does not penalize manifold violations. Consequently, P2M requires heavy manifold reconstructions [HSG18] every few iteration cycles to actually maintain a valid surface. Adapting the genus within the optimization is not possible. But P2M may also start on a given mesh with the correct genus, i.e., a coarse alpha shape [EM94] or Poisson [KH13] mesh. This is also natively supported by our method. Moreover, our algorithm may even start on separated geometry as shown in Figure 7.21 and 7.A.2. Comparisons featured in Figure 7.15 again exemplify the strengths of our approach, i.e., generating a uniform mesh structure and reconstructing sharp features and fine details. This can be seen, especially on the edges of the **G** as well as on the toes and epoccipitals on the triceratops' frill. The target-edge-lengths for our results were set to the average edge length of the P2M meshes, respectively. On the same hardware (GTX 1080 Ti) and starting from the same convex hull mesh, P2M finished the predetermined 6000 iterations for the triceratops within 70 minutes, whereas ours performed 1500 iterations and terminated after 108 seconds.

Ballooning Results Figure 7.16 compares results of CF, CE and P2M to ours. The visualization shows distance errors and normal angle deviations between reconstruction and input point cloud. Although the CF and CE results have a higher mesh resolution than ours, they still struggle to capture the rounded ridges and valleys of the *Bunny*. With the original point cloud (362k samples) and a memory-limited size of 65.5k faces, the official P2M code totally maxed out the 24 GB of memory on a Titan RTX card, performed 30k iterations, ran for 237 h, but the result is still riddled with artifacts. This is the best result we were able to generate. Even without special attention to sharp features, our method still provides the most accurate reconstruction of round and organic shapes.

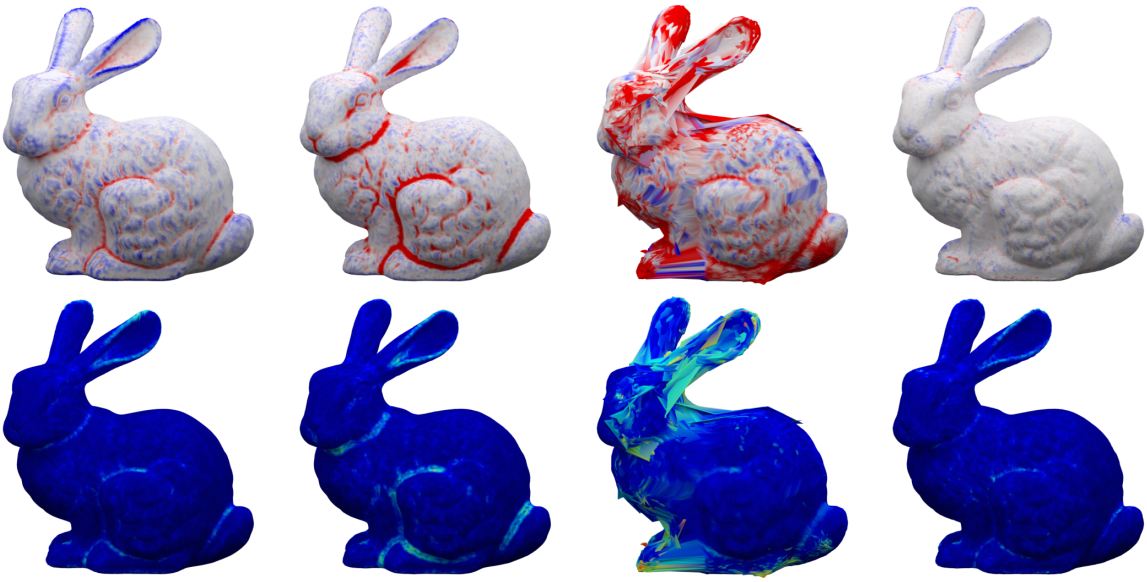


Figure 7.16: Error comparisons with related ballooning approaches, visualizing distance errors (top), normal deviations (bottom) and listing the number of faces (#f).

7.5.3 Discussion

With our method, we achieve the same ε-closeness to the target shape as *Competing Fronts* [SLS*06], which is a guaranteed result of our projection scheme. Figure 7.14, 7.A.3 and 7.A.4 visualize magnified deviation errors form the sampled ground truth meshes. An effect that becomes visible here is the mesh's tendency to exaggerate curved regions. This is caused by the tangent projection, which, on concave regions intersect inside and for convex regions intersect outside of the mesh. By design, mesh vertices are drawn towards intersections of tangent planes, thus resulting in vertices slightly in- or outside of the actual hull. However, this is a small trade-off in our bilateral sample-weighting scheme, which generally allows for a robust and accurate reconstruction of smooth regions and sharp edge features, respectively, even under the presence of noise.

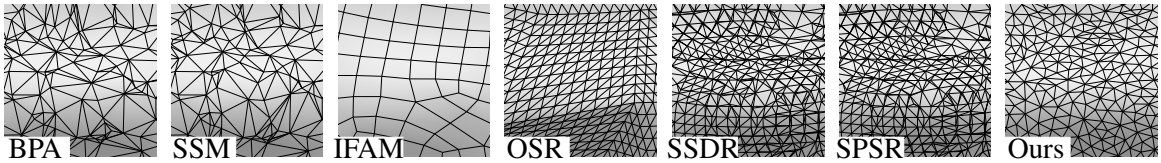
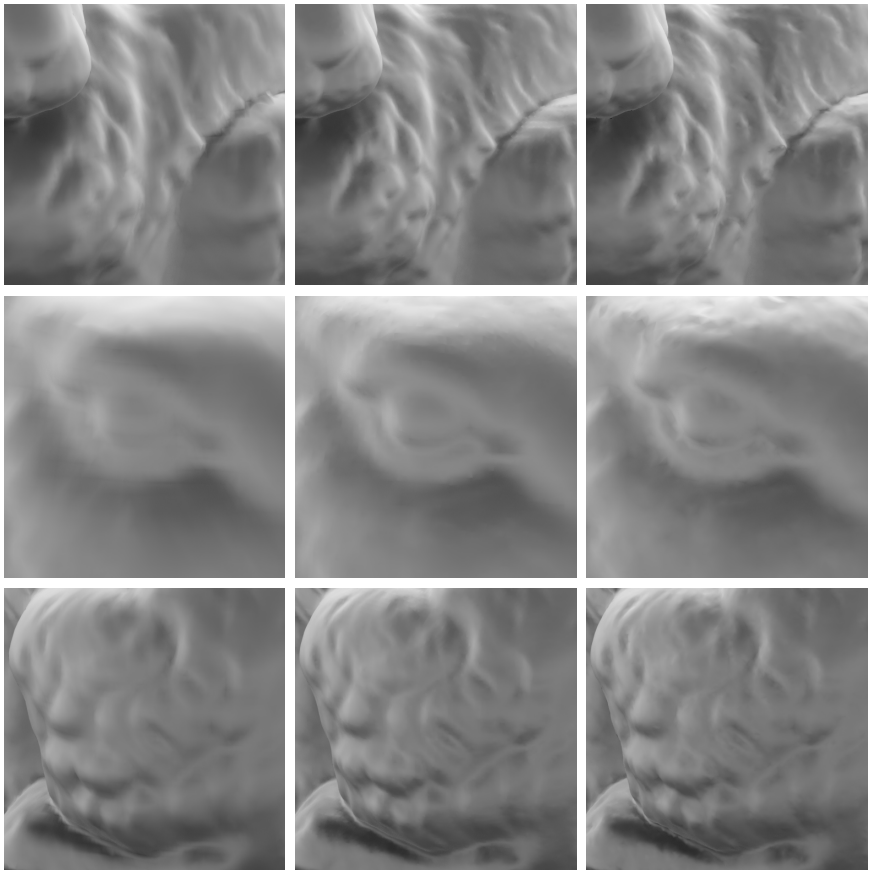


Figure 7.17: Comparison on mesh uniformity with a zoom-in on one of the nobs of object 101634 from Figure 7.A.3.

The listed numerical values in Figure 7.14 and Table 7.1 for HD and RMSE play in the same order of magnitude as compared state-of-the-art methods. Figure 7.18 compares reconstructions of the *Tanagra* high-precision 3D scan [DAL*11]. While the PSR [KBH06] result is again quite smoothed out, the MLS [FCOS05] filtered BPA [BMR*99] result features better details. Nevertheless, our mesh equals or even supersedes the sharpness and fine-detail quality of the SSM [DMSL11] result. The zoom-ins shown in Figure 7.17 demonstrate the achievable mesh quality. Whereas other methods' mesh structures either directly depend on the input points (BPA, SSM), smoothen out details for the sake of regularity (IFAM, OSR) or triangulate octree-cells with heavily varying triangle structures (SSDR, SPSR), our results natively approximate very regular, Delaunay-like triangulations. Figure 7.19 also highlights the prominence of subtle detail in our results, even with much mesh lower resolution compared to the SPSR result.

Figure 7.18: Comparing results of the *Tanagra* 3D scan with PSR, MLS & BPA, SSM and ours. Note the reconstructed mesh is significantly smoother in appearance along the folds of the toga while the inscription (on the back of the statue) is much better readable.



Zipper [TL94, TL14] #v:35k, #f:69k | SPSR [KH13] #v:148k, #f:296k | Ours #v:67k, #f:135k

Figure 7.19: Reconstructing subtle detail: Comparisons of our result with the reference mesh and SPSR obtained from the original point cloud. Even with less than half the number of vertices (#v) and faces (#f), our result features finer details and has an overall sharper appearance than the SPSR.

Section 7.4 introduced the ability to adapt our method with little effort to various other input scenarios, giving examples for high-quality point cloud reconstruction, remeshing, volume- and CT scan meshing, or surface reconstruction from signed distance fields which are currently popular in neural surface representations. Further examples for the versatility of our algorithm are shown in Figure 7.6, where *Spot* is given as a quad-meshed target, enclosed by a coarse initial mesh, which automatically assimilates its shape by *shrink wrapping*. Swapping target and initialization meshes leads to the inverse operation: The only parameter to be adapted is a larger target-edge-length, automatically leading to a coarsening of the mesh as small edges incrementally collapse.

The comparisons in Figure 7.20 include results reconstructed from only 300 very noisy sample points, the default input size for the official Occupancy Network (ONet) implementation [MON*19]. To be able to generate a boat-shaped result like this, the learning-based ONet approach first had to be specifically trained for that object class. Without that prior knowledge of what shape is to be reconstructed, our algorithm also managed to generate a boat-shaped result. Our reconstruction features a few more wrinkles due to the low sample density and heavy noise but achieved sharper edge features and performed better in all listed error measures, including HD, RMSE, SAD, and TVD.

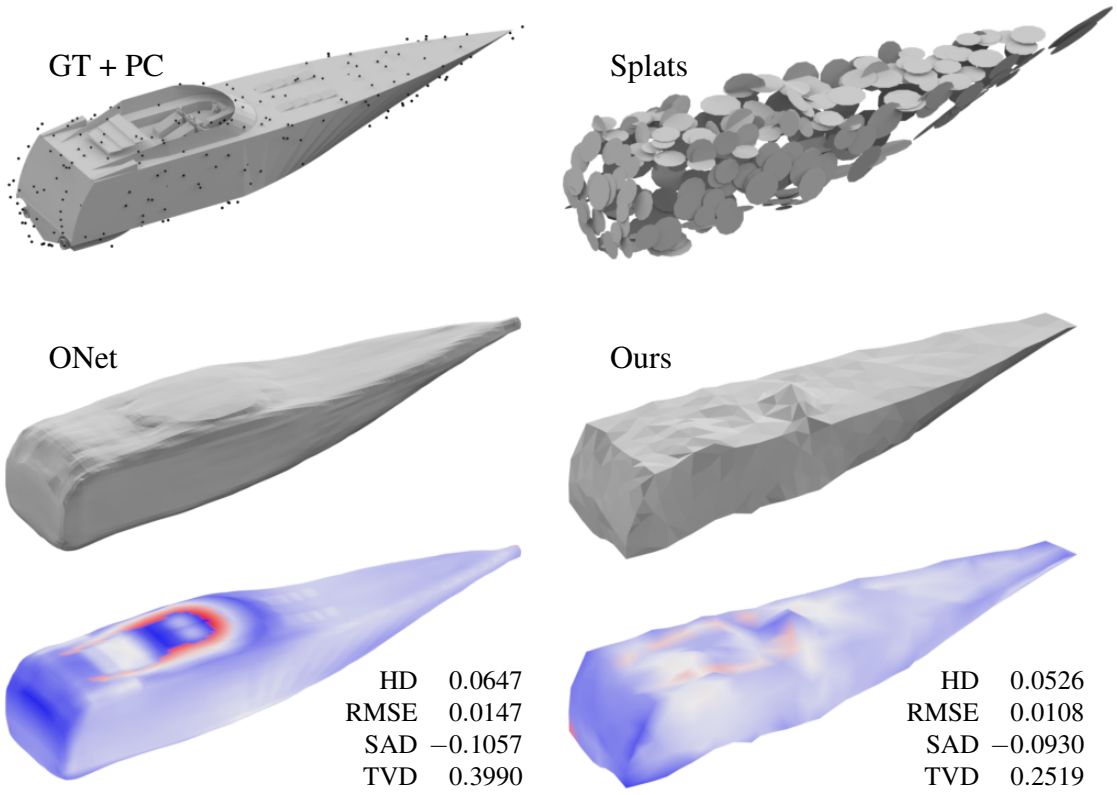


Figure 7.20: Comparison to an Occupancy Network result [MON*19] of a ShapeNet [CFG*15] object. Even on this very limited input of only 300 very noisy sample points and without any preconception of what a boat is, our algorithm reconstructed the shape of the object with sharp feature edges, where appropriate.

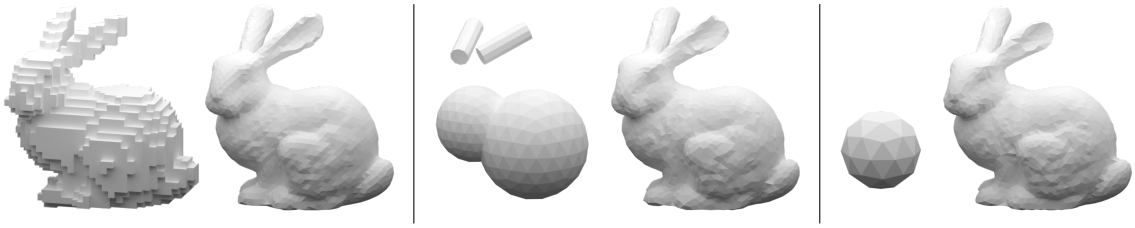


Figure 7.21: Reconstructions of the *Bunny* [TL14] from three different initialization meshes: Starting from a coarse octree-voxelization, separate geometric objects and the default icosphere respectively.

Examples in Figure 7.21 show reconstructions of the *Bunny* given as point cloud but starting from different initialization meshes. The initial mesh on the left is a coarse voxelization of the point cloud volume. As this initial hull already fills out most parts of the target shape, it is quite easy for our algorithm to project vertices into the point cloud while equalizing their distances. In the second case, the algorithm starts from separate low-poly geometry shapes which individually grow and connect via inter-mesh tunnels as they approach each other. The rightmost example is the default mesh, which automatically grows into the desired shape from within the target point cloud. While the same initial meshes lead to identical results, the algorithm will not produce canonical results from different initialization meshes. As exemplified in Figure 7.A.2 (top), the initialization mesh does not necessarily have to be placed entirely within the target shape. The normal-based direction for mesh growth is robust enough to cope with miss-aligned starting conditions so that the initial mesh will get *sucked in* and then starts to grow.

Performance comparisons to other related work might yet not be very meaningful, as they are often single-core CPU implementations and ours a mixture of Python and Cuda code. All vertex-based operations in our pipeline are parallelized and executed on the GPU. Single-threaded operations on the mesh data structure only interleave the fast optimization steps. Therefore, our algorithm is able to successfully finish hundreds of iterations within seconds. Full reconstructions of larger results, however, play in the same order of magnitude as competing techniques, ranging from seconds to a few minutes. A speedup can be gained from tailored input like coarse voxelizations as in Figure 7.21 (left), or well placed initial geometry as in Figure 7.A.2 (bottom), where separate initial meshes grow in parallel until fused to one structure via inter-mesh tunnels.

7.6 Conclusion

In this work, we present a mesh construction and remeshing algorithm that is easy to implement but still checks all the important boxes of the state-of-the-art in this well-researched field: ε-close surface reconstructions of point clouds and other (re)meshing tasks, e.g., volumetric functions or CT-scans, robustness to noise or outliers and support for higher genus object topology with a guaranteed closed, watertight manifold surface. Moreover, we can note several improvements over similar approaches: Instead of global subdivisions, our targeted mesh refinements avoid unnecessary increases of mesh resolution or repeated remeshing steps. The mesh resolution may adapt to the given sample density or distribute vertices uniformly. With a minimal parameter space (target-edge-length or closeness-threshold) to be specified, the process robustly completes fully automated, without the need for different reconstruction modes, finalizing projection or smoothing passes. This can be attributed to our concept of a seamless transition between mesh growth and surface assimilation on an individual vertex basis. Our bilateral weighting scheme for surface projections allows for perfectly captured sharp object features as well as smooth areas. This is substantiated with a bulk error-benchmark on a large dataset comparing state-of-the-art reconstruction methods as well as various qualitative comparisons, including other ballooning concepts. Furthermore, we demonstrate the flexibility of our approach to cope with arbitrary input and initialization situations. As our method is still able to adapt to any given target shape, it exemplifies a powerful advancement and generalization of related *ballooning* and *shrink wrapping* approaches.

Chapter 7

Appendix

7.A More Results

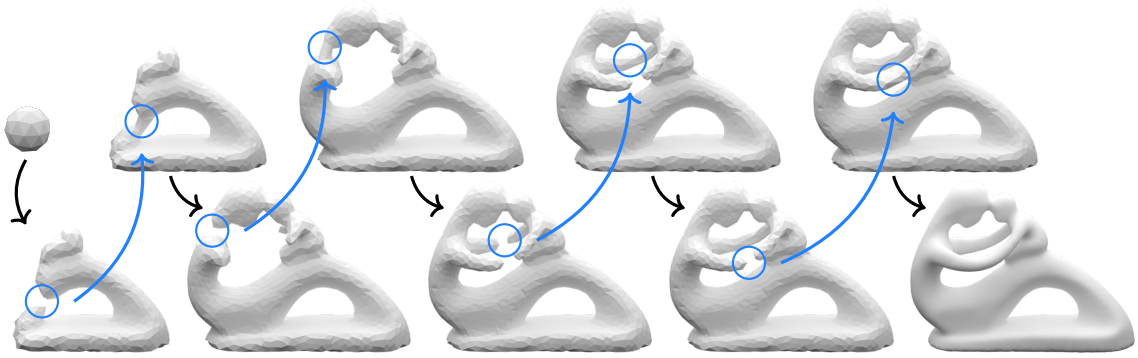


Figure 7.A.1: High-genus objects reconstruct, using intra-mesh tunnels. Black arrows indicate normal growth process. Proximity checks trigger tunneling operations on close (but not directly adjacent) geometry. The iteration continues normally afterwards.

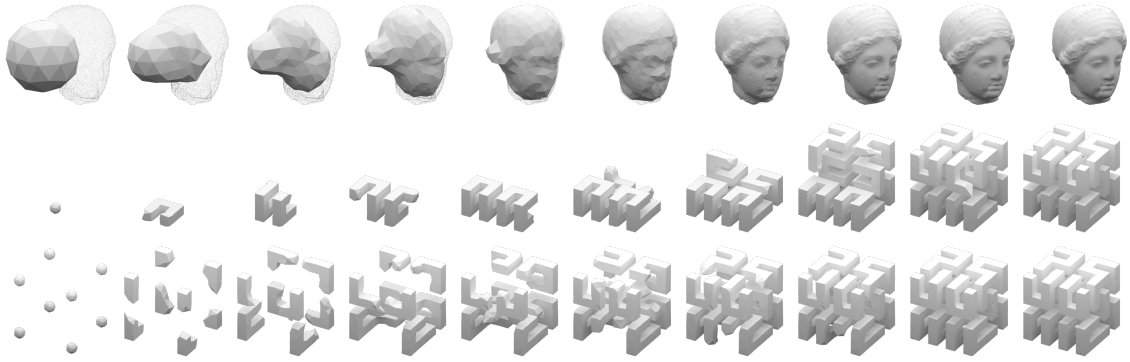


Figure 7.A.2: Top: Miss-aligned initialization meshes get sucked into the target shape via the robust mesh growth orientation. The Hilbert curve: Starting from one (mid row) position and from eight (bottom) individual positions, connected via inter-mesh tunnels.

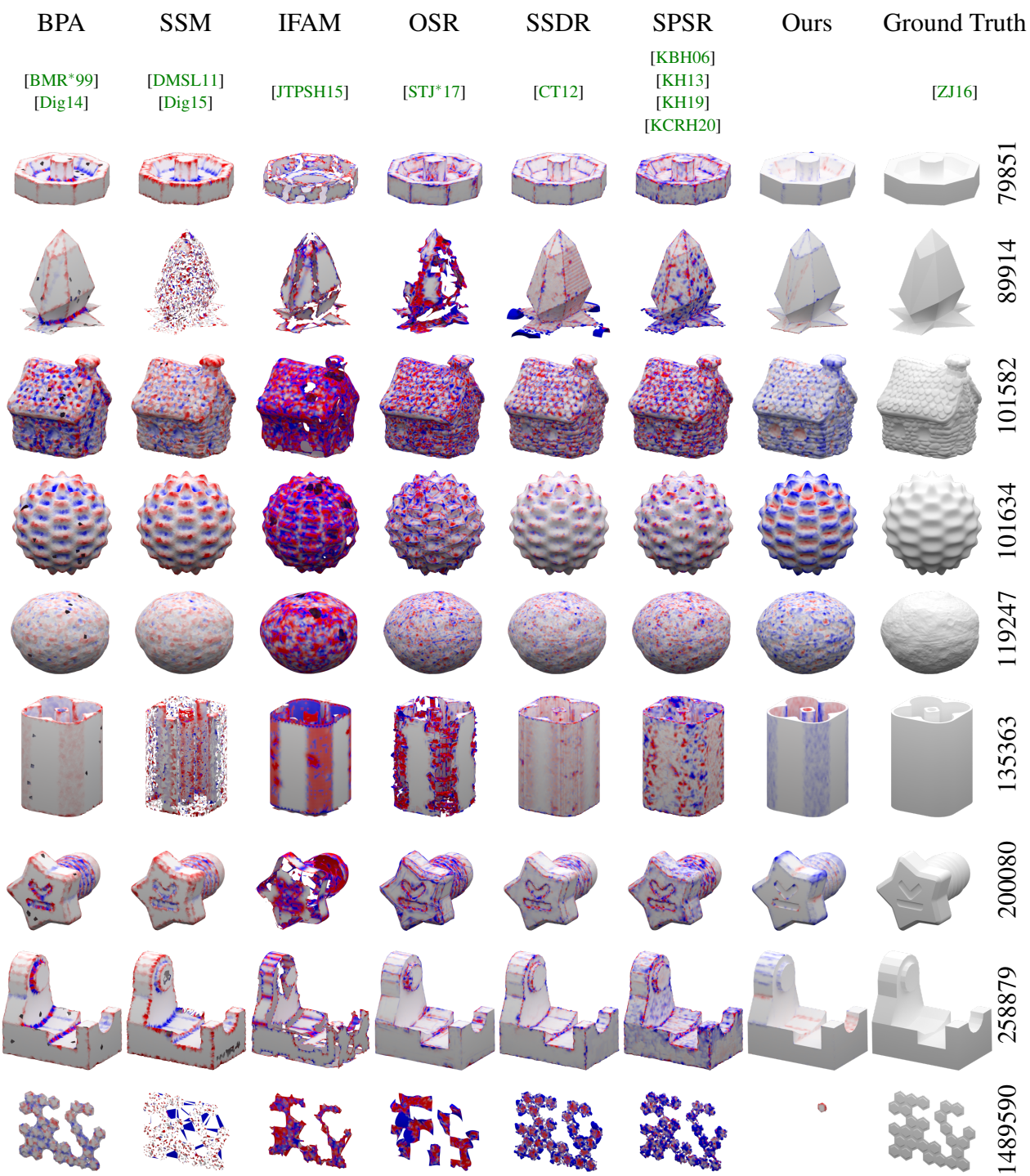


Figure 7.A.3: Randomly selected objects (IDs on the right) from the Thingi10K [ZJ16] data set serve as ground truth. All objects are scaled down to fit in a $[-1{:}1]$ cube and are uniformly sampled. The number of samples computes as surface area $\times$ 1000. 89914 is a thin hollow shell and therefore not as simple as it may appear. 1489590 is a failure case (Section 7.5.1) of our approach where the geometry is too thin for the mesh to grow.

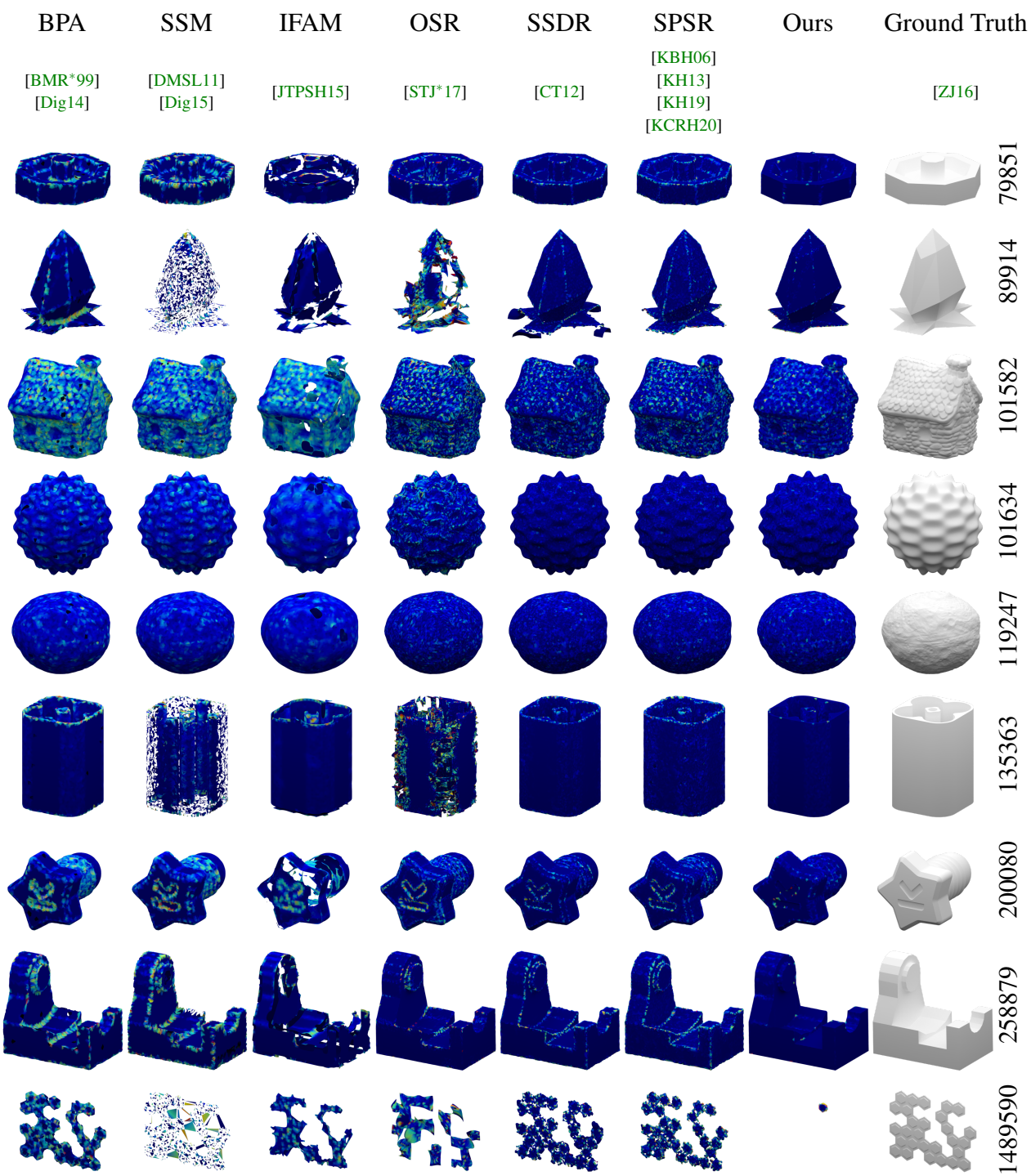


Figure 7.A.4: The same results as from Figure 7.A.3 with visualized surface normal errors. Many methods perform well on curved and flat regions, but sharp edges as on the CAD-like objects are often not recovered very well.

7.B Quad Experiments

As teased on before, the proposed mesh assimilation technique can also be applied to improve upon the reconstruction accuracy of existing meshes regarding their target point clouds, respectively. To exemplify this, we utilize the meshed results created by the procedures introduced in Chapter 4 and 5. We recall that the first proposed method creates pure quad-meshes only, and the second one volumetric meshes in our at-most-hexa class. For this demonstration, we will neglect the volumetric part and only modify the hull. Further internal adaptation can easily follow by propagating the hull movement updates. Firstly, we still have to adapt our procedure to the new the mesh structure. Per default, all operations were designed under the assumption of handling triangular meshes, but now the algorithm has to deal with quad-only and quad-dominant hull meshes, respectively. This is an issue because the *Equalization* step II operates in a context where every vertex is directly edge-linked to its 1-ring neighbors. Applying the algorithm *as is* on quad-meshes is no suitable option. It would only incorporate the directly linked neighbor vertices but not the ones *across the diagonal* of a quad face, thus lead to very disfigured and transformed faces. A simple triangulation would obviously convert any polygonal mesh into a suitable triangular structure. However, regarding a quad-only mesh, it would be quite wasteful to discard this valuable and non-trivial topology. It would furthermore cause immense complications with respect to the underlying at-most-hexa structure when only the quad-dominant hull is triangulated.

Therefore, we specify the following two adaptations to our core algorithm from Section 7.3, which allow for the handling of quadrangular meshes:

The Iteration Cycle We assume that the given mesh already meets the desired criteria in terms of resolution, regularity and genus. To ensure an unaltered mesh topology, we exclude steps I (mesh subdivision), IV (fix irregularities) and V (increase genus) from the iteration cycle. Then, the optimization is run as before, however, now only iterating over the two remaining steps, namely II to equalize distances between the vertices and III for the hull projection.

Mesh Rigidity To avoid disfigured quad faces, we have to treat the quadrangular layout during the relaxation *as rigid as possible* [SA07]. Consequently, each quad is extended temporarily with two crossing diagonal edges as illustrated in Figure 7.B.5. Triangular faces in a quad-dominant mesh remain unaltered. These new diagonal edges allow for a more stable interaction between two vertices of the same quad but on opposing corners.

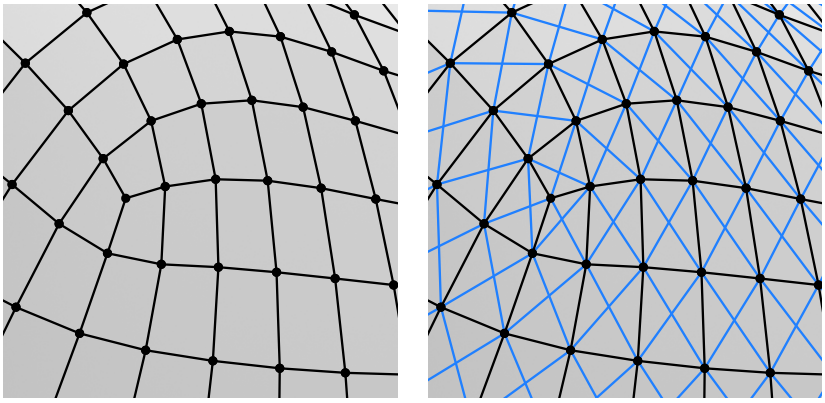


Figure 7.B.5: The quad mesh on the left is supplemented with diagonal edges as shown on the right. While this provides rigidity to the quad structure during the optimization, additional edges are eventually discarded and only the quad mesh is exported.

Furthermore, we adapt step II of the optimization routine: But instead of locally *equalizing* the edge lengths around a vertex, the existing edge-lengths are now to be *maintained* over the iteration procedure.

$$\vec{e}_i' = \left(\frac{1}{k_i} \sum_{v_j \in N_i} v_j - \mathbf{l}_{ij} \frac{\vec{v}_{ij}}{|\vec{v}_{ij}|} \right) - v_i \tag{7.B.1}$$

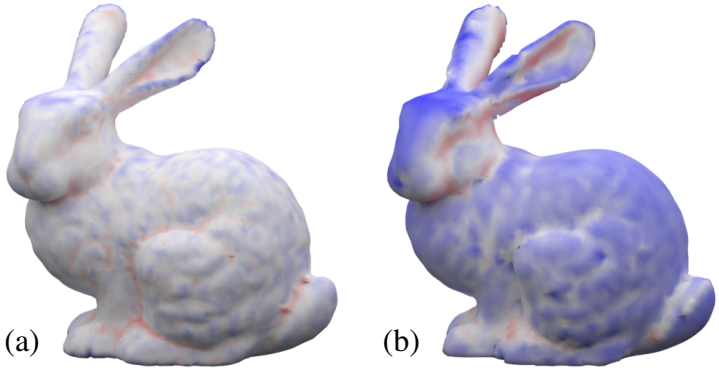
Therefore, the vertex-individual averaged edge length l_i in Equation (7.2) is replaced by $\mathbf{l}_{ij} = |v_j - v_i|$, denoted in bold as it is determined once before the iteration and then kept fix. The adapted update vectors are formulated in Equation (7.B.1). This now also incorporates the new vertex neighbors, linked via the inserted diagonal edges. Movements are incrementally applied in the same manner as before, using a soft update.

7.B.1 Results

Examples of this quad-mesh adaptation are shown in Figure 7.B.6, featuring the native quad-mesh results from Chapter 4, the quad-dominant hull of an at-most-hexa mesh from Chapter 5, the two improved outcomes (using the mesh assimilation algorithm) as well as a native mesh assimilation result, serving as a reference. The listed RMSEs compute the closest distances to the results surface over all point cloud samples. As there is no ground-truth or reference surface, other error measures like the HD, surface or volume deviations are not given. Individual distances to the point cloud samples are accumulated as errors for the individual faces of the resulting surface and visualized in blue and red. In the quad-meshing procedure, final vertices compute as representative points, projected on fitted spheres. While this eventually allows for the hierarchical assembly of quad-tiles, the projection also introduces mild deviations from the point cloud hull, visible in

7.B.6a. In the at-most-hexa pipeline, final geometry vertices are extracted by merging sets of virtual vertices. Whereas this merge has smoothing effects on internal structures, vertices residing on the outer hull might get pulled a bit too far away from the implicit point cloud surface, thus the deviation in 7.B.6a is higher in some regions.

Results of both quad (dominant) meshes treated with the adapted mesh assimilation algorithm are shown in Figures 7.B.6c and 7.B.6d, respectively. In both cases, we can denote significant improvements regarding the reconstruction accuracy, as the RMSEs were brought close to the fine triangulated reference featured in Figure 7.B.6e.



	Method	RMSE
(a)	QM	0.169341
(b)	AMH	0.342660
(c)	QM+MA	0.117783
(d)	AMH+MA	0.136575
(e)	MA	0.106235

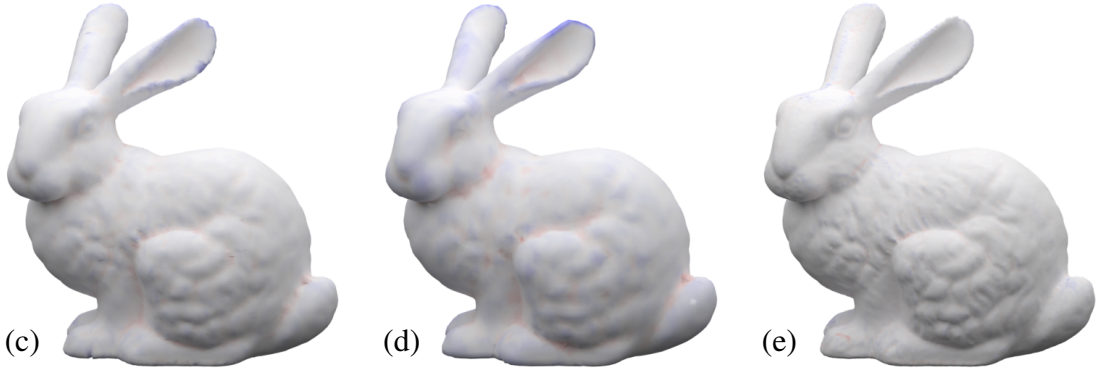


Figure 7.B.6: Native results of our quad-meshing (QM) procedure (Chapter 4) and the quad-dominant hull of an at-most-hexa (AMH) mesh (Chapter 5) are shown in the top row. Improved results with additional mesh assimilation (MA) steps, using the target point cloud, are shown in the bottom row. A native triangulated mesh assimilation result serves as a reference, shown in (e). Listed RMSEs are multiplied by factor 100.

This concludes the experimental demonstration of our proposed assimilation technique to also further improve upon existing reconstructions. It is trivially applicable on triangular meshes, which are then understood as the initialization mesh for the native algorithm. As shown in this section, it is also easily adapted to be compatible with quad-only as well as quad-dominant meshes, allowing for a very wide range of applications scenarios.

Chapter 8

Discussion

This chapter aims to position the presented work within related fields of research and, furthermore, present possibilities for future developments. A recapitulation of our main contributions at the end of this chapter will eventually also conclude this thesis.

8.1 Standing on the Shoulders of Giants

Inspiration comes in many strange ways, for the work presented in this thesis ranging from mosaic image stylization techniques to the philosophical thoughts of Bruce Lee. However, the motivation and main driving force behind each project always come down to curiosity. This work summarizes novel and experimental methods to approach the challenge of point cloud reconstruction and abstraction. A shared characteristic of all the presented ideas is their resourcefulness. Initial ideas were neither related to nor directly incremented from common state-of-the-art approaches. Each concept was always designed with the intention to explore more unconventional ways, hence the name of this thesis, categorizing them as *Maverick Meshing Methods*. Some explorations even lead back to established paths, eventually sharing the same spirit as other related methods in their domain. Nevertheless, each expedition yielded novel insights on their own matter and thus can be credited with valuable additions to their individual fields of research.

8.2 Positioning of this Thesis

• Chapter 4 fits nicely into the field of quad-mesh generation, generally applicable to various reconstruction or remeshing tasks. Standard methods usually rely on given frame-fields and discrete optimization to produce not only a hull but also a feature-aligned mesh. Our approach, however, is based on an initial divide-and-conquer step, subsampling, a *k*d-tree bottom-up assembly routine, and a maze filling algorithm. This unconventional approach sets our method apart from other established procedures, which has advantages but also disadvantages. In comparison to field-aligned methods like IFAM and OSR, our results feature more singularities and a less optimal mesh flow. This can

be attributed to the fact that other methods might leave challenging parts of the reconstruction unresolved by introducing holes or non-quad faces in regions with insufficient sample data. Our algorithm reconstructs these areas with lower mesh uniformity, but a watertight quad-only result is guaranteed, nevertheless.

• Similar to the quad-meshing task, common hex-meshing algorithms usually also require frame-fields, parameterizations, or other complex - often even manually constructed - auxiliary inputs. However, our method from Chapter 5 is able to produce such meshes with much fewer requirements. A simple meshed hull or even only a point cloud is already sufficient to construct nicely feature aligned volume meshes. The proposed at-most-hexa mesh class inherits many benefits from general hex-dominant meshes, like the ease of construction, while being directly applicable to typical hex-only meshing applications. Recent developments on this front usually focus on very specific subtasks of the hex-meshing problem. Our concept presents a novel start-to-finish alternative in the complex field of hex-meshing procedures.

• The methods introduced in Chapter 6 explore advancements in tet-mesh geometry optimization, using a density dimension. Contrasting common fallback solutions, based on voxelized approximations, we formulate applications to optimize density fields using tetrahedral volume meshes, ideally incorporating given 3D geometry without introducing aliasing bias. We furthermore propose the application of this density domain for the analytic formulation of an L_∞ energy, applicable in cells of a non-euclidean 3D Lloyd relaxation. The proposed hybrid $L_{2|\infty}$ relaxation converges in uniformly oriented 3D space and may be utilized in the future to fully overcome current approximation methods.

• Simple point-cloud-to-triangle-mesh reconstruction methods are an exhaustively studied research topic. Nevertheless, with our method, introduced in Chapter 7, we are still able to introduce improvements in this field. A distinction from other ballooning or shrink-wrapping approaches is our bilateral weighting scheme, allowing individual vertices to transition from general mesh growth mode to a hull projection. Whereas other methods usually operate under strict constraints of smoothness, ours is drawn to strong edges and eventually produces much higher reconstruction accuracy on fine details and sharp features. As our method is a generalization of common approaches, it is further able to mesh various types of implicit or explicit surface and volume data. We also demonstrate our procedure's applicability to improve the accuracy of other reconstruction results, exemplified on our own quad- and hex-mesh results.

8.3 Outlook

Potential future improvements on our individual procedures were already discussed in the corresponding chapters, respectively. We still want draw some broad outlines on future work topics, related to or influenced by our methods.

Machine Learning For the longest parts of its history, 3D reconstruction was dominated by classic geometrical concepts. Nevertheless, learning-based methods recently also made some progress towards the formulation of a 3D context [MON*19]. However, S2M [DN19] is, by definition, limited to reconstruct only objects of a specific pretrained class and does not generalize well for arbitrary point clouds. P2M [HMGCO20] on the other hand, can do precisely that: reconstruct point clouds of arbitrary shape without prior training. In spirit, the approach is quite similar to our mesh assimilation algorithm from Chapter 7 but operates without topological constraints. Thus, it requires heavy remeshing steps every other iteration and is therefore much, much slower than the competition. Furthermore, we demonstrate in a comparison that the achievable mesh quality of P2M is not very uniform, and the results are also less precise than ours. Still, learning-based methods bear some potential in this field and could prove their value in future developments, regarding geometry construction.

The relaxation used for our at-most-hexa mesh construction in Chapter 5 requires to construct a Voronoi diagram and then integrate over its cells' volume to evaluate a Lloyd relaxation. While concepts that rely on the L_2 norm suggest omitting the explicit diagram computation [RSLL18, BAR*21], it still seems necessary in our context, as we use the anisotropic L_∞ norm. This task is very challenging to formulate and our approach for the integration, described in Chapter 6, only solves one half of the problem. If a proper solution to this problem can be formulated and solved geometrically or analytically, it definitely should be computed that way. Nevertheless, learning-based estimates for the relaxation's update steps could potentially be *close enough* to the real increments, thus serve as a fast and sufficiently accurate substitute.

As already teased on in associated Chapters 4 and 5, the graph structures generated in our quad- and hex-meshing procedures encode shape information, potentially useful for other tasks, rather than just surface reconstruction. For example, to determine segmentation, using classic graph-cut algorithms or for the matching and understanding of geometric objects with more dedicated learning-based algorithms.

Mesh Improvement Although being primarily proposed as a meshing procedure for volumes, SDF or point cloud input, the methods presented in Chapter 7 on mesh assimilation are quite versatile, thus easily applicable to various other challenges. The technique is also related to every other chapter in this thesis, as every meshed object (hull or

volume) is eventually wrapped in a triangulatable mesh. Further, its projection scheme formulates a basic quality improvement strategy for any meshing task, thus could be incorporated in other meshing procedures, directly improving their outcome, applied on existing results as a post-processing step, or used on its own as a remeshing procedure.

Manufacturing Our Chapter 6 on tet-geometry utilizes a variable density domain and merely teased on some exemplary application scenarios for this technique. Research on additive manufacturing gained much popularity in recent years due to the fast developments in available hardware. We included examples of how variable density in 3D prints can be achieved with a straightforward method and a rather basic FDM printer setup. Still, we are looking forward for the discovery of more novelties in this field and possible combinations of our proposed concepts with new materialization pipelines, allowing to discover the full potential of variable density in 3D printing.

Performance All shown examples are based on implementations with more or less optimized code written in *Python*. Math is usually broken down into fast C-array-based *NumPy* vector operations. Where possible, large bulk operations are executed in parallel using multithreading or specialized Cuda GPU code. Nevertheless, a proper and clean reimplementation of the fast-prototype-oriented research code would certainly further improve listed performance statements.

8.4 Conclusion

With the initially formulated goal for this thesis, to explore and extend the field of meshing procedures with novel alternatives, we will now conclude by listing several accomplishments of the individual approaches.

In the fields of surface reconstruction and meshing, the methods described in Chapters 4 and 7 extend and supersede established state-of-the-art procedures in various domains. Both methods are able to deal with the most common point cloud artifacts: Our quad-meshing algorithm is designed to adapt to varying sample densities with reduced or increased mesh resolution. The assimilation algorithm is able to produce a homogeneous mesh resolution, invariant of the given point cloud density but provides an option for adaptive results as well. Per design, both methods produce watertight manifold meshes, which, among meshing algorithms, cannot be taken for granted. They are invariant to outliers as well as robust to noise and missing data in the given point cloud. The first algorithm produces pure quad-meshes, alignable to orientations and features of the input object, without the requirement for additional information besides the point cloud. Our second algorithm is more focused on reconstructing high-quality surfaces and fine details with uniform triangular meshes. By utilizing normal information of the point cloud, it is able to snap edges and vertices on sharp corners and strong feature edges,

thus supersedes the accuracy and sharpness of related meshing procedures. Divide-and-conquer algorithms, as well as space-partitioning, are quite common tools in computer science. But to the best of our knowledge, the way hierarchical structures are used in our quad-meshing technique, has never been approached in this manner before. Our assimilation algorithm is the result of a goal-driven development but eventually shares some similarities to existing methods. However, when comparing the results, our algorithm demonstrates the true potential of this common concept and exemplifies what level of detail can actually be achieved if done properly. Point cloud reconstruction was set as the common goal and is therefore also the primary field of application in which both procedures excel. Nevertheless, a byproduct of the quad-meshing algorithm is a spatial hierarchy, associable with distinct surface patches, thus potentially valuable for other kinds of application. The mesh assimilation is furthermore able to reconstruct meshes from various kinds of input, rather than only point clouds, e.g., other meshes, isosurfaces, or volumes encoded as binary data or SDFs.

Geometric objects with varying densities are common in physics environments and simulation tasks. Thus, in Chapter 6 we recapitulate formulas, easy to use and implement for incorporating varying density of tet-meshes in our exemplary application scenarios. We demonstrate its usage for optimizing mass properties of geometric objects, simply balancing them or improving their axial stability when rotated. The chapter furthermore exemplifies the practical applicability of optimized density structures for 3D printing, a relatively young and still growing field of research. However, another valuable contribution of this tets-with-density concept is the ability to analytically formulate energy terms of cells within a 3D Lloyd relaxation under the L_∞ norm, like the one used for our at-most-hexa meshing concept of Chapter 5. Unlike other hex-meshing procedures, ours does not require much more than a simple hull or even only a point cloud of the target object. The shape is populated with L_∞ cells, aligned in the relaxation procedure to form hexahedral grid structures. A graph-matching algorithm eventually assembles the final mesh primitives, topologically not larger than a hexahedron, thus specifying a generalized subclass of common hex-meshes, named at-most-hexa meshes. This new specification allows for simplified storing and handling of primitives and meshes with indexed data structures, thus our at-most-hexa meshes are easily compatible with downstream applications, initially designed for hex-only meshes. The generalized formulation of primitives is trivially suitable for volume integration, whereas arbitrary polyhedra of other hex-dominant techniques are not. Our results level with the state-of-the-art in terms of feature alignment, hex/non-hex proportions as well as mesh quality. We propose the required algorithms and demonstrate how our at-most-hexa meshes are easier to construct while requiring much simpler input than related methods.

List of Abbreviations

ASJ Average Scaled Jacobian. 19, 94, 95

BPA Ball Pivoting Algorithm [BMR*99]. 34, 143, 155, 159, 160, 166, 167

BSP Binary Space Partitioning. 43

CAD Computer-Aided Design. 29, 37, 67, 68, 69, 77, 94, 99, 102, 104, 156, 167

CE Cooperative Evolutions [LL20]. 143, 157, 158, 159

CF Competing Fronts [SLS*06]. 143, 157, 158, 159

CGI Computer-Generated Imagery. 30

CNN Convolutional Neural Network. 35

CPU Central Processing Unit. 64, 128, 163

CT Computed Tomography. 30, 31, 151, 161, 164

CVT Centroidal Voronoi Tessellation. 22, 23, 24, 25, 69, 73, 82, 98, 125

fcc Face-Centered Cubic. 24, 76

FDM Fused Deposition Modeling. 124, 174

FEM Finite Element Method. 36, 114, 116, 117

GPU Graphics Processing Unit. x, 68, 70, 73, 76, 114, 116, 128, 142, 163, 174

HD Hausdorff Distance. 26, 27, 59, 61, 85, 98, 121, 155, 156, 160, 162, 169

ICP Iterative Closest Point. 32

IFAM Instant Field-Aligned Meshes [JTPSH15]. 35, 52, 59, 60, 61, 62, 144, 155, 159, 160, 166, 167, 171

*k***NN** *k* Nearest Neighbor. xii, xiii, 73, 76, 77, 78, 81, 92, 108

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm. 126

MCA Marching Cubes Algorithm [LC87]. 34

MLS Robust Moving Least-Squares Fitting [FCOS05]. 160

MRI Magnetic Resonance Imaging. 30, 31

MSJ Minimum Scaled Jacobian. 19, 36, 89, 90, 94, 95

NAD Normal Angle Deviation. 28, 61, 155

ONet Occupancy Network [MON*19]. 162

OSR Field Aligned Online Surface Reconstruction [STJ*17]. 35, 52, 59, 60, 61, 62, 144, 155, 159, 160, 166, 167, 171

P2M Point2Mesh [HMGCO20]. 35, 143, 158, 159, 173

PN Point-Normal triangles [VPBM01]. 145

PSR Poisson Surface Reconstruction [KBH06]. 34, 160

RANSAC Random Sample Consensus. 58

RMSE Root Mean Squared Error. 27, 59, 61, 85, 98, 121, 155, 156, 160, 162, 169, 170

S2M Scan2Mesh [DN19]. 35, 173

SAD Surface Area Deviation. 28, 155, 156, 162

SDF Signed Distance Field/Function. 35, 36, 143, 150, 151, 156, 157, 173, 175

slerp Spherical Linear Interpolation. 15

SPSR Screened Poisson Surface Reconstruction [KH13]. 143, 155, 159, 160, 161, 166, 167

SSDR Smooth Signed Distance Colored Surface Reconstruction [CT12]. 155, 159, 160, 166, 167

SSM Scale Space Meshing [DMSL11]. 143, 155, 159, 160, 166, 167

TVD Total Volume Deviation. 28, 155, 156, 162

WT Watertight. 11, 28, 155

Bibliography

[AB99] AMENTA N., BERN M.: Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry 22*, 4 (1999), 481–504.

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 415–421.

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications* (2001), pp. 249–266.

[AUGA08] ALLIEZ P., UCELLI G., GOTSMAN C., ATTENE M.: Recent advances in remeshing of surfaces. In *Shape analysis and structuring*. Springer, 2008, pp. 53–82.

[BAR*21] BASSELIN J., ALONSO L., RAY N., SOKOLOV D., LEFEBVRE S., LÉVY B.: Restricted power diagrams on the gpu. In *Eurographics 2021* (2021).

[Bau72] BAUMGART B. G.: *Winged edge polyhedron representation*. Tech. rep., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1972.

[BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4 (July 2013), 98:1–98:12. doi:10.1145/2461912.2462014.

[BDS*18] BARILL G., DICKSON N. G., SCHMIDT R., LEVIN D. I., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 43.

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), pp. 185–192.

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon mesh processing*. AK Peters/CRC Press, 2010.

[BL17] BOLTCHEVA D., LÉVY B.: Surface reconstruction by computing restricted voronoi cells in parallel. *Computer-Aided Design 90* (2017), 123–134.

[BL18] BUKENBERGER D. R., LENSCH H. P. A.: Hierarchical Quad Meshing of 3D Scanned Surfaces. *Computer Graphics Forum 37*, 5 (2018), 131–141. doi:10.1111/cgf.13497.

[BL21a] BUKENBERGER D. R., LENSCH H. P. A.: Be Water my Friend: Mesh Assimilation. *The Visual Computer 37*, 9 (2021), 2725–2739. doi:10.1007/s00371-021-02183-6.

[BL21b] BUKENBERGER D. R., LENSCH H. P. A.: Tetrahedra of Varying Density and Their Applications. *The Visual Computer 37*, 9 (2021), 2447–2460. doi:10.1007/s00371-021-02189-0.

[BLC16] BOTELLA A., LÉVY B., CAUMON G.: Indirect unstructured hex-dominant mesh generation using tetrahedra recombination. *Computational Geosciences 20*, 3 (2016), 437–451.

[Ble20] BLENDER ONLINE COMMUNITY: *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2020. URL: https://www.blender.org.

[BLN*13] BERGER M., LEVINE J. A., NONATO L. G., TAUBIN G., SILVA C. T.: A benchmark for surface reconstruction. *ACM Transactions on Graphics (TOG) 32*, 2 (2013), 1–17.

[BLNZ95] BYRD R. H., LU P., NOCEDAL J., ZHU C.: A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing 16*, 5 (1995), 1190–1208.

[BLP*13] BOMMES D., LÉVY B., PIETRONI N., PUPPO E., SILVA C., TARINI M., ZORIN D.: Quad-mesh generation and processing: A survey. *Computer Graphics Forum 32*, 6 (2013), 51–76. doi:10.1111/cgf.12014.

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics 5*, 4 (1999), 349–359.

[Bow81] BOWYER A.: Computing dirichlet tessellations. *The computer journal 24*, 2 (1981), 162–166.

[BRM*14] BAUDOUIN T. C., REMACLE J.-F., MARCHANDISE E., HENROTTE F., GEUZAINE C.: A frontal approach to hex-dominant mesh generation.

Advanced Modeling and Simulation in Engineering Sciences 1, 1 (2014), 8.

[BSH*16] BEHRENS C., SCHUBERT T., HAVERKAMP S., EULER T., BERENS P.: Connectivity map of bipolar cells and photoreceptors in the mouse retina. *Elife 5* (2016), e20041.

[BTL21] BUKENBERGER D. R., TARINI M., LENSCH H. P. A.: At-Most-Hexa Meshes. *Computer Graphics Forum 41*, 1 (2021). doi:10.1111/cgf.14393.

[BTP*19] BRACCI M., TARINI M., PIETRONI N., LIVESU M., CIGNONI P.: Hexalab.net: An online viewer for hexahedral meshes. *Computer-Aided Design 110* (2019), 24 – 36. URL: https://www.hexalab.net/, doi:10.1016/j.cad.2018.12.003.

[BTS*17] BERGER M., TAGLIASACCHI A., SEVERSKY L. M., ALLIEZ P., GUENNEBAUD G., LEVINE J. A., SHARF A., SILVA C. T.: A survey of surface reconstruction from point clouds. In *Computer Graphics Forum* (2017), vol. 36, Wiley Online Library, pp. 301–329.

[BWBSH14] BÄCHER M., WHITING E., BICKEL B., SORKINE-HORNUNG O.: Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 96.

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3 (July 2009), 77:1–77:10. doi:10.1145/1531326.1531383.

[CAS*19] CHERCHI G., ALLIEZ P., SCATENI R., LYON M., BOMMES D.: Selective padding for polycube-based hexahedral meshing. In *Computer Graphics Forum* (2019), vol. 38.1, Wiley Online Library, pp. 580–591.

[Cat74] CATMULL E.: *A subdivision algorithm for computer display of curved surfaces*. Tech. rep., DTIC Document, 1974.

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design 10*, 6 (1978), 350–355.

[CC19] CORMAN E., CRANE K.: Symmetric moving frames. *ACM Trans. Graph. 38*, 4 (2019).

[CCC*08] CIGNONI P., CALLIERI M., CORSINI M., DELLEPIANE M., GANOVELLI F., RANZUGLIA G.: MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference* (2008),

Scarano V., Chiara R. D., Erra U., (Eds.), The Eurographics Association. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[CDP21] CHEILARIS P., DEY S. K., PAPADOPOULOU E.: L infinity segment delaunay graphs. In *CGAL User and Reference Manual*, 5.2.1 ed. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgSegmentDelaunayGraphLinf2.

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: *ShapeNet: An Information-Rich 3D Model Repository*. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[CGAY13] CAMPOS R., GARCIA R., ALLIEZ P., YVINEC M.: Splat-based surface reconstruction from defect-laden point sets. *Graphical Models 75*, 6 (2013), 346–361.

[CI00] CALVO N. A., IDELSOHN S. R.: All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision. *Computer Methods in Applied Mechanics and Engineering 182*, 3-4 (2000), 371–378.

[Cra20] CRANE K.: Keenan's 3D Model Repository, 2020. https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/.

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: measuring error on simplified surfaces. In *Computer Graphics Forum* (1998), vol. 17.2, Blackwell Publishers, pp. 167–174.

[CS98] CONWAY J. H., SLOANE N. J. A.: *Sphere packings, lattices and groups*. Springer, 1998.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *ACM SIGGRAPH 2004 Papers* (2004), pp. 905–914.

[CT12] CALAKLI F., TAUBIN G.: Ssd-c: Smooth signed distance colored surface reconstruction. In *Expanding the Frontiers of Visual Analytics and Visualization*, Dill J., Earnshaw R., Kasik D., Vince J., Wong P. C., (Eds.). Springer London, 2012, pp. 323–338. doi:10.1007/978-1-4471-2804-5_18.

[CW10] CHANG H.-C., WANG L.-C.: A simple proof of thue's theorem on circle packing. *arXiv preprint arXiv:1009.4322* (2010).

[DAL*11] DIGNE J., AUDFRAY N., LARTIGUE C., MEHDI-SOUZANI C., MOREL J.-M.: Farman Institute 3D Point Sets - High Precision 3D Data Sets. *Image Processing On Line 1* (2011), 281–291. doi:10.5201/ipol.2011.dalmm_ps.

[Daw] DAWSON-HAGGERTY ET AL.: *trimesh - a pure Python library for loading and using triangular meshes*. URL: https://trimsh.org/.

[DEJ06] DU Q., EMELIANENKO M., JU L.: Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM journal on numerical analysis 44*, 1 (2006), 102–119.

[Dig14] DIGNE J.: An Analysis and Implementation of a Parallel Ball Pivoting Algorithm. *Image Processing On Line 4* (2014), 149–168. doi:10.5201/ipol.2014.81.

[Dig15] DIGNE J.: An Implementation and Parallelization of the Scale Space Meshing Algorithm. *Image Processing On Line 5* (2015), 282–295. doi:10.5201/ipol.2015.102.

[DML11] DONG W., MOSES C., LI K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 577–586.

[DMSL11] DIGNE J., MOREL J.-M., SOUZANI C.-M., LARTIGUE C.: Scale space meshing of raw data point sets. In *Computer Graphics Forum* (2011), vol. 30.6, Wiley Online Library, pp. 1630–1642.

[DN19] DAI A., NIESSNER M.: Scan2mesh: From unstructured range scans to 3d meshes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 5574–5583.

[DQ01] DUAN Y., QIN H.: Intelligent balloon: a subdivision-based deformable model for surface reconstruction of arbitrary topology. In *Proceedings of the sixth ACM symposium on Solid modeling and applications* (2001), pp. 47–58.

[DSC09] DANIELS J., SILVA C. T., COHEN E.: Semi-regular quadrilateral-only remeshing from simplified base domains. In *Computer Graphics Forum* (2009), vol. 28/5, Wiley Online Library, pp. 1427–1435.

[D'U14] D'URSO M.: Gravity effects of polyhedral bodies with linearly varying density. *Celestial Mechanics and Dynamical Astronomy 120*, 4 (2014), 349–372.

[Dug66] DUGUNDJI J.: *Topology*. Allyn and Bacon series in advanced mathematics. Allyn and Bacon, 1966.

[EBCK13] EBKE H.-C., BOMMES D., CAMPEN M., KOBBELT L.: Qex: robust quad mesh extraction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 168.

[Ebe02] EBERLY D.: Polyhedral mass properties (revisited). *Geometric Tools, LLC, Tech. Rep* (2002).

[EM94] EDELSBRUNNER H., MÜCKE E. P.: Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG) 13*, 1 (1994), 43–72.

[Ent20] ENTHOUGHT INC.: *Mayavi: 3D visualization of scientific data in Python*, 2020. URL: https://docs.enthought.com/mayavi/mayavi/.

[ESCK16] EBKE H.-C., SCHMIDT P., CAMPEN M., KOBBELT L.: Interactively controlled quad remeshing of high resolution 3d models. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 218:1–218:13. doi:10.1145/2980179.2982413.

[FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C. T.: Robust moving least-squares fitting with sharp features. *ACM transactions on graphics (TOG) 24*, 3 (2005), 544–552.

[FDCO03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 950–953. doi:10.1145/1201775.882368.

[For87] FORTUNE S.: A sweepline algorithm for voronoi diagrams. *Algorithmica 2*, 1 (1987), 153–174.

[GJTP17] GAO X., JAKOB W., TARINI M., PANOZZO D.: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 114.

[GLL*04] GOESELE M., LENSCH H. P., LANG J., FUCHS C., SEIDEL H.-P.: Disco: acquisition of translucent objects. *ACM Transactions on Graphics (TOG)* (2004), 835–844.

[GMGP05] GELFAND N., MITRA N. J., GUIBAS L. J., POTTMANN H.: Robust global registration. In *Symposium on geometry processing* (2005), vol. 2.3, Vienna, Austria, p. 5.

[GPW*17] GAO X., PANOZZO D., WANG W., DENG Z., CHEN G.: Robust structure simplification for hex re-meshing. *ACM Transactions on Graphics 36*, 6 (2017).

[GRL17] GROH F., RESCH B., LENSCH H. P. A.: *Multi-view Continuous Structured Light Scanning*. Springer International Publishing, Cham, 2017, pp. 377–388. doi:10.1007/978-3-319-66709-6_30.

[GSP19] GAO X., SHEN H., PANOZZO D.: Feature preserving octree-based hexahedral meshing. In *Computer Graphics Forum* (2019), vol. 38.5, Wiley Online Library, pp. 135–149.

[GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-hex mesh generation via volumetric polycube deformation. In *Computer graphics forum* (2011), vol. 30.5, Wiley Online Library, pp. 1407–1416.

[HAB*17] HALES T., ADAMS M., BAUER G., DANG T. D., HARRISON J., LE TRUONG H., KALISZYK C., MAGRON V., MCLAUGHLIN S., NGUYEN T. T., ET AL.: A formal proof of the kepler conjecture. In *Forum of mathematics, Pi* (2017), vol. 5, Cambridge University Press.

[Han99] HANSEN R.: An analytical expression for the gravity field of a polyhedral body with linearly varying density. *Geophysics 64*, 1 (1999), 75–77.

[Hau01] HAUSNER A.: Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 573–580.

[HBT*13] HELMSTAEDTER M., BRIGGMAN K. L., TURAGA S. C., JAIN V., SEUNG H. S., DENK W.: Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature 500*, 7461 (2013), 168–174.

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1992), SIGGRAPH '92, ACM, pp. 71–78. doi:10.1145/133994.134011.

[HIKL*99] HOFF III K. E., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 277–286.

[HJ19] HALEEM A., JAVAID M.: 3d scanning applications in medical field: a literature-based review. *Clinical Epidemiology and Global Health 7*, 2 (2019), 199–210.

[HK06] HORNUNG A., KOBBELT L.: Robust Reconstruction of Watertight 3D Models from Non-uniformly Sampled Point Clouds Without Normal Information. In *Symposium on Geometry Processing* (2006), Sheffer A., Polthier K., (Eds.), The Eurographics Association. doi:10.2312/SGP/SGP06/041-050.

[HKD*20] HORNUS S., KUIPERS T., DEVILLERS O., TEILLAUD M., MARTÍNEZ J., GLISSE M., LAZARD S., LEFEBVRE S.: Variable-width contouring for additive manufacturing. *ACM Transactions on Graphics 39*, Siggraph'2020 Conference proceedings (2020).

[HM91] HUDSON D. G., MARSH E. W.: The Making of 'Terminator 2: Judgment Day', 1991.

[HMGCO20] HANOCKA R., METZER G., GIRYES R., COHEN-OR D.: Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph. 39*, 4 (July 2020). doi:10.1145/3386569.3392415.

[HS17] HORMANN K., SUKUMAR N.: *Generalized barycentric coordinates in computer graphics and computational mechanics*. CRC Press, 2017.

[HSG18] HUANG J., SU H., GUIBAS L.: Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698* (2018).

[HZG*18] HU Y., ZHOU Q., GAO X., JACOBSON A., ZORIN D., PANOZZO D.: Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 60.

[HZM*08] HUANG J., ZHANG M., MA J., LIU X., KOBBELT L., BAO H.: Spectral quadrangulation with orientation and alignment control. In *ACM SIGGRAPH Asia 2008 papers*. ACM, 2008, pp. 1–9.

[IKL*08] IHRKE I., KUTULAKOS K. N., LENSCH H. P., MAGNOR M., HEIDRICH W.: State of the art in transparent and specular object reconstruction. In *EUROGRAPHICS 2008 STAR–STATE OF THE ART REPORT* (2008), Citeseer.

[JBG19] JAKOB J., BUCHENAU C., GUTHE M.: Parallel globally consistent normal orientation of raw unorganized point clouds. In *Computer Graphics Forum* (2019), vol. 38.5, Wiley Online Library, pp. 163–173.

[JTPSH15] JAKOB W., TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Instant field-aligned meshes. *ACM Trans. Graph. 34*, 6 (2015), 189–1.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), SGP '06, Eurographics Association, pp. 61–70.

[KBLK14] KREMER M., BOMMES D., LIM I., KOBBELT L.: Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proceedings of the 22nd International Meshing Roundtable*. Springer, 2014, pp. 147–164.

[KCRH20] KAZHDAN M., CHUANG M., RUSINKIEWICZ S., HOPPE H.: Poisson surface reconstruction with envelope constraints. In *Computer Graphics Forum* (2020), vol. 39.5, Wiley Online Library, pp. 173–182.

[KH13] KAZHDAN M., HOPPE H.: Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG) 32*, 3 (2013), 1–13.

[KH19] KAZHDAN M., HOPPE H.: An adaptive multi-grid solver for applications in computer graphics. In *Computer Graphics Forum* (2019), vol. 38.1, Wiley Online Library, pp. 138–150.

[Kle89] KLEIN R.: *Concrete and abstract Voronoi diagrams*, vol. 400. Springer Science & Business Media, 1989.

[KMJ*19] KOCH S., MATVEEV A., JIANG Z., WILLIAMS F., ARTEMOV A., BURNAEV E., ALEXA M., ZORIN D., PANOZZO D.: Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384. doi:10.1111/j.1467-8659.2007.01060.x.

[Knu95] KNUPP P.: Mesh generation using vector-fields. *J. Comput. Phys. 119*, 1 (June 1995), 142–148. doi:10.1006/jcph.1995.1122.

[Kob00] KOBBELT L.: $\sqrt{3}$-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 103–112.

[KPL*12] KLÖCKNER A., PINTO N., LEE Y., CATANZARO B., IVANOV P., FASIH A.: PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing 38*, 3 (2012), 157–174. doi:10.1016/j.parco.2011.09.001.

[KS00] KAPLAN C. S., SALESIN D. H.: Escherization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 499–510.

[KWW19] KUIPERS T., WU J., WANG C. C.: Crossfill: Foam structures with graded density for continuous material extrusion. *Computer-Aided Design 114* (2019), 37–50.

[LA] LEVY B., ALONSO L.: Graphite, a research platform for computer graphics, 3D modeling and numerical geometry. URL: http://alice.loria.fr/index.php/software/3-platform/22-graphite.html.

[LaT20] LATEX PROJECT TEAM: *The LaTeXProject*, 2020. URL: https://www.latex-project.org.

[LBA*17] LEAL R., BARREIROS F., ALVES L., ROMEIRO F., VASCO J., SANTOS M., MARTO C.: Additive manufacturing tooling for the automotive industry. *The International Journal of Advanced Manufacturing Technology 92*, 5-8 (2017), 1671–1676.

[LBK16] LYON M., BOMMES D., KOBBELT L.: Hexex: robust hexahedral mesh extraction. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 123.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics 21*, 4 (1987), 163–169.

[LCOL07] LIPMAN Y., COHEN-OR D., LEVIN D.: Data-Dependent MLS for Faithful Surface Approximation. In *Geometry Processing* (2007), Belyaev A., Garland M., (Eds.), The Eurographics Association. doi:10.2312/SGP/SGP07/059-067.

[LCOLTE07] LIPMAN Y., COHEN-OR D., LEVIN D., TAL-EZER H.: Parameterization-free projection for geometry reconstruction. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. doi:10.1145/1275808.1276405.

[LFF*20] LI B., FU J., FENG J., SHANG C., LIN Z.: Review of heterogeneous material objects modeling in additive manufacturing. *Visual Computing for Industry, Biomedicine, and Art 3*, 1 (2020), 1–18.

[LGS02] LENSCH H. P., GOESELE M., SEIDEL H.-P.: Digital collections of real world objects. *D-Lib Magazine 8*, 2 (2002). URL: http://webdoc.sub.gwdg.de/edoc/aw/d-lib/dlib/february02/goesele/02goesele.html.

[LHJ*14] LING R., HUANG J., JÜTTLER B., SUN F., BAO H., WANG W.: Spectral quadrangulation with feature curve alignment and element size control. *ACM Transactions on Graphics (TOG) 34*, 1 (2014), 1–11.

[LKG*01] LENSCH H. P., KAUTZ J., GOESELE M., HEIDRICH W., SEIDEL H.-P.: Image-based reconstruction of spatially varying materials. In *Eurographics Workshop on Rendering Techniques* (2001), Springer, pp. 103–114.

[LKG*03] LENSCH H. P., KAUTZ J., GOESELE M., HEIDRICH W., SEIDEL H.-P.: Image-based reconstruction of spatial appearance and geometric detail. *ACM Transactions on Graphics (TOG) 22*, 2 (2003), 234–257.

[LL10] LÉVY B., LIU Y.: l_p centroidal voronoi tessellation and its applications. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29.4, ACM, p. 119.

[LL20] LU W., LIU L.: Surface reconstruction via cooperative evolutions. *Computer Aided Geometric Design 77* (2020), 101831.

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE transactions on information theory 28*, 2 (1982), 129–137.

[LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 177.

[LMR07] LINSEN L., MULLER K., ROSENTHAL P.: Splat-based ray tracing of point clouds. *Journal of WSCG 15* (01 2007), 51–58.

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 131–144. URL: https://graphics.stanford.edu/data/mich/, doi:10.1145/344779.344849.

[LPP*20] LIVESU M., PIETRONI N., PUPPO E., SHEFFER A., CIGNONI P.: LoopyCuts: Practical Feature-Preserving Block Decomposition for Strongly Hex-Dominant Meshing. *ACM Transactions on Graphics 39*, 4 (2020). doi:10.1145/3386569.3392472.

[LPW*06] LIU Y., POTTMANN H., WALLNER J., YANG Y.-L., WANG W.: Geometric modeling with conical meshes and developable surfaces. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH 06,

Association for Computing Machinery, pp. 681–689. doi:10.1145/1179352.1141941.

[LR09] LIU Y.-S., RAMANI K.: Robust principal axes determination for point-based shapes using least median of squares. *Comput. Aided Des. 41*, 4 (Apr. 2009), 293–305. doi:10.1016/j.cad.2008.10.012.

[LSVT15] LIVESU M., SHEFFER A., VINING N., TARINI M.: Practical hex-mesh optimization via edge-cone rectification. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 141.

[LZC*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained octahedral fields for hexahedral meshing. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 93.

[MAB*15] MUSIALSKI P., AUZINGER T., BIRSAK M., WIMMER M., KOBBELT L.: Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph. 34*, 4 (2015), 102–1.

[Mar93] MARSH E. W.: Under Pressure: Making 'The Abyss', 1993.

[MAR*20] MATVEEV A., ARTEMOV A., RAKHIMOV R., BOBROVSKIKH G., PANOZZO D., ZORIN D., BURNAEV E.: Def: Deep estimation of sharp geometric features in 3d shapes. *arXiv preprint arXiv:2011.15081* (2020).

[Max60] MAX J.: Quantizing for minimum distortion. *IRE Transactions on Information Theory 6*, 1 (1960), 7–12.

[Max99] MAX N.: Weights for computing vertex normals from facet normals. *Journal of graphics tools 4*, 2 (1999), 1–6.

[MB12] MOUTON T., BÉCHET E.: Lloyd relaxation using analytical voronoi diagram in the l_infinite norm and its application to quad optimization. *Proceedings of the 21st International Meshing Roundtable* (2012).

[MCAG07] MOUSA M.-H., CHAINE R., AKKOUCHE S., GALIN E.: Efficient spherical harmonics representation of 3d objects. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2007), PG '07, IEEE Computer Society, pp. 248–255. doi:10.1109/PG.2007.19.

[MCCO07] MARKLEY F. L., CHENG Y., CRASSIDIS J. L., OSHMAN Y.: Averaging quaternions. *Journal of Guidance, Control, and Dynamics 30*, 4 (2007), 1193–1197.

[McG00] MCGUIRE M.: The half-edge data structure. *Website: http://www. flipcode. com/articles/article halfedgepf. shtml* (2000).

[Mea82] MEAGHER D.: Geometric modeling using octree encoding. *Computer graphics and image processing 19*, 2 (1982), 129–147.

[Mir96] MIRTICH B.: Fast and accurate computation of polyhedral mass properties. *Journal of graphics tools 1*, 2 (1996), 31–50.

[MON*19] MESCHEDER L., OECHSLE M., NIEMEYER M., NOWOZIN S., GEIGER A.: Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2019).

[Mor09] MORIGI S.: Dept. of Math. Bologna University Scan Repository, 2009. https://www.dm.unibo.it/~morigi/homepage_file/research_file/scan_db/res_scan.html.

[MT00] MESHKAT S., TALMOR D.: Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *International Journal for Numerical Methods in Engineering 49*, 1-2 (2000), 17–30.

[NAB] NELLES J. N., AL-BADRI N.: Nefertiti Head Model. URL: https://nefertitihack.alloversky.com/.

[NRP11] NIESER M., REITEBUCH U., POLTHIER K.: Cubecover–parameterization of 3d volumes. In *Computer graphics forum* (2011), vol. 30.5, Wiley Online Library, pp. 1397–1406.

[Num20] NUMPY COMMUNITY: Array programming with NumPy. *Nature 585* (2020), 357–362. doi:10.1038/s41586-020-2649-2.

[NZH*18] NI S., ZHONG Z., HUANG J., WANG W., GUO X.: Field-aligned and lattice-guided tetrahedral meshing. In *Computer Graphics Forum* (2018), vol. 37.5, Wiley Online Library, pp. 161–172.

[OBB*13] ORZAN A., BOUSSEAU A., BARLA P., WINNEMÖLLER H., THOLLOT J., SALESIN D.: Diffusion curves: a vector representation for smooth-shaded images. *Communications of the ACM 56*, 7 (2013), 101–108.

[OBS03] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *Shape Modeling International, 2003* (2003), IEEE, pp. 153–161.

[Onl20] ONLINE-SOURCES: Recent Gradient Infill Developements, 2020. https://ultimaker.com/en/resources/52670-infill https://www.sublimelayers.com/2019/05/dynamic-infill-density-in-new-kisslicer.html https://www.cnckitchen.com/blog/gradient-infill-for-3d-prints.

[PAM11] PELLENARD B., ALLIEZ P., MORVAN J.-M.: Isotropic 2d quadrangle meshing with size and orientation control. In *Proceedings of the 20th International Meshing Roundtable*. Springer, 2011, pp. 81–98.

[PBJSH16] PRÉVOST R., BÄCHER M., JAROSZ W., SORKINE-HORNUNG O.: Balancing 3d models with movable masses. In *VMV* (2016).

[PdC76] PERDIGÃO DO CARMO M.: *Differential geometry of curves and surfaces*. Prentice-Hall, 1976.

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 165–174.

[PJVR18] PELLERIN J., JOHNEN A., VERHETSEL K., REMACLE J.-F.: Identifying combinations of tetrahedra into hexahedra: A vertex based strategy. *Computer-Aided Design 105* (2018), 1–10.

[PNA*21] PIETRONI N., NUVOLI S., ALDERIGHI T., CIGNONI P., TARINI M.: Reliable feature-line driven quad-remeshing. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 17. URL: https://www.quadmesh.cloud.

[Pow64] POWELL M. J.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal 7*, 2 (1964), 155–162.

[Pri08] PRICE D. A.: *The Pixar Touch*. Vintage, 2008.

[PTS*08] PÉBAY P. P., THOMPSON D., SHEPHERD J., KNUPP P., LISLE C., MAGNOTTA V. A., GROSLAND N. M.: New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *Proceedings of the 16th International Meshing Roundtable* (2008), Springer, pp. 535–552.

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., SORKINE-HORNUNG O.: Make it stand: balancing shapes for 3d fabrication. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 81.

[Pyt20] PYTHON SOFTWARE FOUNDATION: *Python*, 2020. URL: https://www.python.org.

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling* (2001), IEEE, pp. 145–152.

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph. 25*, 4 (Oct. 2006), 1460–1485. doi:10.1145/1183287.1183297.

[RNV07] RATHOD H., NAGARAJA K., VENKATESUDU B.: Numerical integration of some functions over an arbitrary linear tetrahedra in euclidean three-dimensional space. *Applied Mathematics and Computation 191*, 2 (2007), 397–409.

[RR96] RATHOD H., RAO H. G.: Integration of polynomials over an arbitrary tetrahedron in euclidean three-dimensional space. *Computers & structures 59*, 1 (1996), 55–65.

[RSLL18] RAY N., SOKOLOV D., LEFEBVRE S., LÉVY B.: Meshless Voronoi on the GPU. *ACM Transactions on Graphics 37* (2018). doi:10.1145/3272127.3275092.

[RSR*18] RAY N., SOKOLOV D., REBEROL M., LEDOUX F., LÉVY B.: Hex-dominant meshing: mind the gap! *Computer-Aided Design 102* (2018), 94–103.

[Rus04] RUSINKIEWICZ S.: Estimating curvatures and their derivatives on triangle meshes. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on* (2004), IEEE, pp. 486–493.

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Symposium on Geometry processing* (2007), vol. 4, pp. 109–116.

[SCC11] STANCULESCU L., CHAINE R., CANI M.-P.: Freestyle: Sculpting meshes with self-adaptive topology. *Computers & Graphics 35*, 3 (2011), 614–622.

[Sch95] SCHULTZ J.: The Making of 'Jurassic Park', 1995. Amblin Entertainment.

[Sci20] SCIPY 1.0 CONTRIBUTORS: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272. doi:10.1038/s41592-019-0686-2.

[SHG*19] SCHNEIDER T., HU Y., GAO X., DUMAS J., ZORIN D., PANOZZO D.: A large scale comparison of tetrahedral and hexahedral elements for finite element analysis. *arXiv preprint arXiv:1903.09332* (2019).

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985), pp. 245–254.

[Si15] SI H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS) 41*, 2 (2015), 1–36.

[SJ08] SHEPHERD J. F., JOHNSON C. R.: Hexahedral mesh generation constraints. *Engineering with Computers 24*, 3 (2008), 195–213.

[SLS*06] SHARF A., LEWINER T., SHAMIR A., KOBBELT L., COHEN-OR D.: Competing fronts for coarse–to–fine surface reconstruction. In *Computer Graphics Forum* (2006), vol. 25.3, Wiley Online Library, pp. 389–398.

[SRUL16] SOKOLOV D., RAY N., UNTEREINER L., LÉVY B.: Hexahedral-dominant meshing. *ACM Transactions on Graphics (TOG) 35*, 5 (2016), 157.

[SSW09] SILVA C. T., SCHEIDEGGER C. E., WANG H.: Bandwidth selection and reconstruction quality in point-based surfaces. *IEEE Transactions on Visualization & Computer Graphics 15* (2009), 572–582. doi:10.1109/TVCG.2009.13.

[STJ*17] SCHERTLER N., TARINI M., JAKOB W., KAZHDAN M., GUMHOLD S., PANOZZO D.: Field-aligned online surface reconstruction. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 77.

[SVB17] SOLOMON J., VAXMAN A., BOMMES D.: Boundary element octahedral fields in volumes. *ACM Trans. Graph. 36*, 3 (May 2017), 28:1–28:16. doi:10.1145/3065254.

[Tak19] TAKAYAMA K.: Dual sheet meshing: An interactive approach to robust hexahedralization. *Computer Graphics Forum (proceedings of Eurographics) 38*, 2 (2019), 37–48. doi:10.1111/cgf.13617.

[Tau95] TAUBIN G.: Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on* (1995), IEEE, pp. 852–857.

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. In *ACM transactions on graphics (TOG)* (2004), vol. 23.3, ACM, pp. 853–860.

[TL94] TURK G., LEVOY M.: Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), pp. 311–318.

[TL14] TURK G., LEVOY M.: The Stanford 3D Scanning Repository, 2014. URL: https://graphics.stanford.edu/data/3Dscanrep/.

[TLGS05] TARINI M., LENSCH H. P., GOESELE M., SEIDEL H.-P.: 3d acquisition of mirroring objects using striped patterns. *Graphical Models 67*, 4 (2005), 233–259.

[Ton04] TONON F.: Explicit exact formulas for the 3-d tetrahedron inertia tensor in terms of its vertex coordinates. *Journal of Mathematics and Statistics 1*, 1 (2004), 8–11.

[TPC*10] TARINI M., PIETRONI N., CIGNONI P., PANOZZO D., PUPPO E.: Practical quad mesh simplification. *Computer Graphics Forum (Special Issue of Eurographics 2010 Conference) 29*, 2 (2010), 407–418. doi:10.1111/j.1467-8659.2009.01610.x.

[TW98] THÜRRNER G., WÜTHRICH C. A.: Computing vertex normals from polygonal facets. *Journal of graphics tools 3*, 1 (1998), 43–46.

[VPBM01] VLACHOS A., PETERS J., BOYD C., MITCHELL J. L.: Curved pn triangles. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM, pp. 159–166.

[Wat81] WATSON D. F.: Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal 24*, 2 (1981), 167–172.

[WAWS17] WU J., AAGE N., WESTERMANN R., SIGMUND O.: Infill optimization for additive manufacturing-approaching bone-like porous structures. *IEEE transactions on visualization and computer graphics 24*, 2 (2017), 1127–1140.

[WHG*15] WU S., HUANG H., GONG M., ZWICKER M., COHEN-OR D.: Deep points consolidation. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 176:1–176:13. doi:10.1145/2816795.2818073.

[WK04] WU J., KOBBELT L.: Optimized sub-sampling of point sets for surface splatting. In *Computer Graphics Forum* (2004), vol. 23.3, Wiley Online Library, pp. 643–652.

[YDS*18] YAN Q., DONG H., SU J., HAN J., SONG B., WEI Q., SHI Y.: A review of 3d printing technology for medical applications. *Engineering 4*, 5 (2018), 729–742.

[YS03] YAMAKAWA S., SHIMADA K.: Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *International journal for numerical methods in engineering 57*, 15 (2003), 2099–2129.

[YWLL13] YAN D.-M., WANG W., LÉVY B., LIU Y.: Efficient computation of clipped voronoi diagram for mesh generation. *Computer-Aided Design 45*, 4 (2013), 843–852.

[ZBLN97] ZHU C., BYRD R. H., LU P., NOCEDAL J.: Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS) 23*, 4 (1997), 550–560.

[ZHLB10] ZHANG M., HUANG J., LIU X., BAO H.: A wave-based anisotropic quadrangulation method. In *ACM SIGGRAPH 2010 papers*. ACM, 2010, pp. 1–8.

[ZJ16] ZHOU Q., JACOBSON A.: Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).