TU Delft

Delft University of Technology

Feature-aware manifold meshing and remeshing of point clouds and polyhedral surfaces with guaranteed smallest edge length

Lipschütz, Henriette; Reitebuch, Ulrich; Polthier, Konrad; Skrodzki, Martin

DOI
10.1016/j.cad.2025.104010

Publication date
2026

Document Version
Final published version

Published in
CAD Computer Aided Design

Citation (APA)
Lipschütz, H., Reitebuch, U., Polthier, K., & Skrodzki, M. (2026). Feature-aware manifold meshing and remeshing of point clouds and polyhedral surfaces with guaranteed smallest edge length. *CAD Computer Aided Design*, *192*, Article 104010. https://doi.org/10.1016/j.cad.2025.104010

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Computer-Aided Design 192 (2026) 104010

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Feature-aware manifold meshing and remeshing of point clouds and polyhedral surfaces with guaranteed smallest edge length ☆

Henriette Lipschütz [a], Ulrich Reitebuch [a], Konrad Polthier [a], Martin Skrodzki [b,*,1]

[a] Freie Universität Berlin, Germany
[b] TU Delft, The Netherlands

ARTICLE INFO

Keywords:
Meshing
Remeshing
Geometry processing
Algorithm

ABSTRACT

Point clouds and polygonal meshes are widely used when modeling real-world scenarios. Here, point clouds arise, for instance, from acquisition processes applied in various surroundings, such as reverse engineering, rapid prototyping, or cultural preservation. Based on these raw data, polygonal meshes are created to, for example, run various simulations. For such applications, the utilized meshes must be of high quality. This paper presents an algorithm to derive triangle meshes from unstructured point clouds. The occurring edges have a close to uniform length and their lengths are bounded from below. Theoretical results guarantee the output to be manifold, provided suitable input and parameter choices. Further, the paper presents several experiments establishing that the algorithms can compete with widely used competitors in terms of quality of the output and timing and the output is stable under moderate levels of noise. Additionally, we expand the algorithm to detect and respect features on point clouds as well as to remesh polyhedral surfaces, possibly with features.

Supplementary material, an extended preprint, a link to a previously published version of the article, utilized models, and implementation details are made available online.

1. Introduction

Point cloud meshing is an important topic present in different fields of research and in various applications. Examples include reverse engineering [1], rapid prototyping [2], or architecture [3]. A common approach to enable this raw data for further processing is to create a triangle mesh from the point cloud. The quality of this mesh is, however, affected by outliers, noise, or non-uniform distribution of the input data. Thus, badly formed mesh elements can become apparent in the resulting geometric model. They can be long-stretched, thin triangles, so-called slivers, or topological issues. These faulty representations have to be repaired before the meshes are further processed.

While this issue is rather general and inherent to the workflow, recent research still struggles to circumvent it. Even when reducing to only a local mesh representation of a given geometry, established methods, such as Delaunay triangulations, do not guarantee to create a manifold mesh of well-shaped triangles [4, Sec. 4.4]. The present paper aims to close this gap.

We aim to reconstruct a surface from a given point cloud via a sphere-packing approach [5]. The goal is to create a manifold output with guaranteed smallest edge length and with strong consideration of triangle quality provided by a distribution close to uniformity of edge lengths. Furthermore, as opposed to other meshing approaches, our algorithm works directly on the surface geometry, that is, does not need any parametrization. Finally, the algorithm performs a greedy disk-growing approach, which enables the processing of the geometry in one pass, making further iterations unnecessary.

A first version of this algorithm has been presented at the 2024 International Meshing Roundtable [6]. The contributions of the original article included:

- introduction of a geometric approach suitable to mesh point clouds,
- which creates high-quality triangles with edge lengths close to uniformity and of a guaranteed minimum length,
- as well as manifold output, provided a suitable input geometry,
- in a single sweep over said input,

discussed in Sections 3 to 5. In this extended version of the article, we build upon the previous contributions and extend the algorithm to handle:

☆ This article is part of a Special issue entitled: 'SIAM IMR 2023/24' published in Computer-Aided Design.

* Corresponding author.
E-mail address: mail@ms-math-computer.science (M. Skrodzki).

[1] Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Germany – 455095046.

https://doi.org/10.1016/j.cad.2025.104010
Received 7 March 2025; Received in revised form 30 July 2025; Accepted 14 November 2025
Available online 17 November 2025
0010-4485/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

- detection of sharp feature ridges in point clouds,
- remeshing of polyhedral surfaces obtaining high-quality meshes with edge lengths close to uniformity,
- detection of sharp feature ridges on polyhedral meshes,

presented in Sections 6 to 9.

2. Related work

In the last decades, several attempts were made to reconstruct the ground truth from a given point cloud $\mathcal{P}$. The resulting reconstruction depends on the quality of $\mathcal{P}$, which can include noisy points or normals, outliers, or be sampled non-uniformly. On top of a reconstruction, the user may ask for guarantees such as correct topology [7], or convergence to the ground truth with increasing sampling density [8]. Some algorithms guarantee local connectedness of their output [9], while others guarantee their output to stay within the convex hull of the given input [10]. Other requirements might be, for instance, a result mesh of high quality, that is, consisting of triangles with edge length close to uniformity and vertices of degree close to 6. Finally, the reconstruction should be computed fast. For an overview of surface reconstruction algorithms, we refer to a recent survey [11]. The algorithms discussed in the following were chosen for their wide use in the field and will serve as a comparison in Section 5.

First, we consider surface reconstruction based on a Poisson equation [12], implemented in CGAL [13]. An implicit function framework is built, where the reconstructed surface appears by extracting an appropriate isosurface. The output is smooth and robustly approximates noisy data. Additionally, densely sampled regions allow the reconstruction of sharp features while sparsely sampled regions are smoothly reconstructed. In later work, these ideas are further developed to create watertight meshes fitting an oriented point cloud by using adaptive, finite elements multi-grid solvers capable of solving a linear system discretized over a spatial domain [14], implemented in MeshLab [15].

Second, the scale-space approach [16], implemented in CGAL [13], aims at topological correctness by choosing triangles based on a confidence-based criterion. This avoids the accumulation of errors, which is often detected in greedy approaches. The algorithm is interpolating, and can handle sharp features to a certain extent, but does not come with proven topological correctness.

The advancing front algorithm [17], implemented in CGAL [13], handles sets of unorganized points without normal information. It computes a normal field and meshes the complete point cloud directly, which leads to a high-level reconstruction of details as well as to an accurate delineation of holes in the ground truth. Therefore, a smoothing operator consistent with the intrinsic heat equation is introduced. By construction, this approach is almost interpolating and features are preserved given very low levels of noise.

The robust implicit moving least squares (RIMLS) algorithm [18], implemented in MeshLab [15], combines implicit MLS with robust statistics. The MLS approach [8] is a widely used tool for functional approximation of irregular data. The development of RIMLS is based on a surface definition formulated in terms of linear kernel regression minimization using a robust objective function which gives a simple and technically sound implicit formulation of the surface. Thus, RIMLS can handle noisy data, outliers, and sparse sampling, and can reconstruct sharp features. The number of iterations needed to achieve a reliable result increases near sharp features while smooth regions only need a single iteration. Furthermore, RIMLS belongs to the set of algorithms producing approximating meshes.

Another approach is based on placing triangles with regard to the restricted Voronoi diagram of a filtered input point set [19], available via MeshLab [15]. This approach has the largest similarity to our algorithm, as we will also employ Voronoi diagrams, however, only to filter points on the tangent plane. Another shared aspect is that both this and our algorithm work on a set of disks centered at the input points, oriented orthogonal to a guessed or provided normal direction.

All these algorithms come with different guarantees regarding the output. However, none of these algorithms comes with a guarantee on the edge length, and only some algorithms are guaranteed to provide a manifold mesh. As we base our surface reconstruction on a set of touching spheres placed on the underlying surface [5], we are able to provide certain theoretical guarantees on the output: given suitable input and parameter choices, our output is always manifold. Furthermore, the output of our algorithm has a guaranteed minimum edge length, while striving towards uniformity of occurring edge lengths. For better comparison to other algorithms, we also employ the "Isotropic Explicit Remeshing" filter of MeshLab [15]. This filter repeatedly applies edge flip, collapse, relax, and refine operations [20]. For remeshing polyhedral surfaces, we also make use of the "Remeshing" routine provided by the Polygon Mesh Processing (PMP) library [21]. Here, the triangle quality is improved by local modifications such as splitting and collapsing of edges and tangential smoothing [22]. Similar algorithms exist that are based on local operations like edge flips and vertex relocations to achieve anisotropic triangular meshes [23]. As opposed to our approach, in their work, the resulting edge lengths depend on the curvature of the geometry remeshed. Additionally, all of the works listed above require several iterations to obtain an isotropic triangulation while our approach works in a single sweep over the input geometry. In Sections Section 5.1, 8, and 9, we will provide a detailed comparison of our algorithm with the works listed here.

When it comes to feature-aware (re)meshing, as a comparison metric between the obtained meshes, we will employ the Hausdorff distance. This allows us to compare the input, be it a point cloud or a surface mesh, with the (re)meshed version. An estimate of the one-sided Hausdorff distance is computed by sampling the input and projecting the samples onto the output [24]. A variety of methods has been proposed to identify features on point clouds and meshes. We follow a basic approach using the variation of normals of two elements [25], allowing us to specify a critical dihedral angle between two normals from which a feature between the two is respected during remeshing. This follows similar approaches in point cloud and surface smoothing [26, 27].

3. Theory and methodology

Our algorithm aims to reconstruct a manifold $\mathcal{M}$ from a given point cloud $\mathcal{P}$. In order to obtain a manifold mesh with a guaranteed minimum edge length, we first present assumptions and theoretical results on both $\mathcal{M}$ and $\mathcal{P}$ in Section 3.1. Based on these theoretical results, we present a geometric approach in Section 3.2, which consists of creating a sphere packing from which the output is constructed. Sections 3.3 to 3.7 are devoted to explaining the different steps in detail.

3.1. Assumptions and theory

Here, we will derive the assumptions to be made on $\mathcal{M}$ and $\mathcal{P}$ to ensure that the constructed surface mesh is manifold. Let $\mathcal{M}$ be an orientable, compact C^2-manifold embedded into $\mathbb{R}^3$, which is assumed to be closed and of finite reach

$$\rho := \inf\left\{\|a-m\| \mid a \in \mathcal{A}_{\mathcal{M}} \wedge m \in \mathcal{M}\right\} \in \mathbb{R}_{>0},$$

where $\mathcal{A}_{\mathcal{M}}$ is the *medial axis* of $\mathcal{M}$ consisting of the points $q \in \mathbb{R}^3$ satisfying

$$\min_{p\in\mathcal{M}} \|q-p\| = \|q-\hat{p}\| = \|q-\tilde{p}\|$$

for $\hat{p} \neq \tilde{p} \in \mathcal{M}$. On the manifold $\mathcal{M}$, we define the *geodesic distance* $d_{\mathcal{M}}$ as follows:

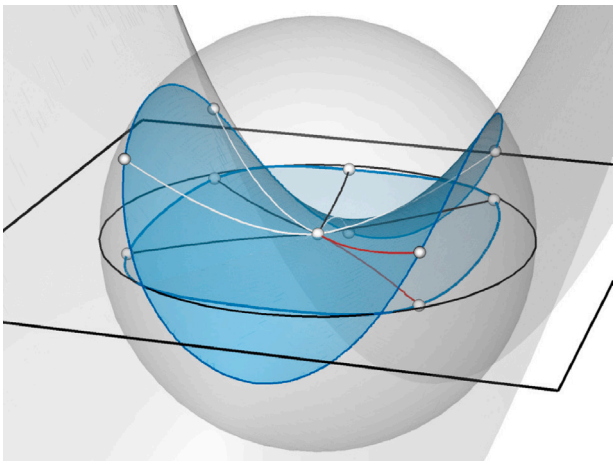


Fig. 1. Illustration of Lemma 3.1. Intersection of a saddle-shaped surface with a sphere (shown in blue). Six points on the surface are marked as well as their projections to the tangent plane belonging to the center of the sphere. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Definition 3.1. Let $\text{len}(f) = \int_0^1 \left|f'(t)\right| \, dt$ denote the length of a curve $f \in C^1([0,1], \mathcal{M})$ in $\mathcal{M}$. Then the *geodesic distance* $d_{\mathcal{M}}$ of $m, m' \in \mathcal{M}$ is defined as $\inf\left\{\text{len}(f) \mid f \in C^1([0,1], \mathcal{M}) : f(0) = m \wedge f(1) = m'\right\}$.

Now, for any $p, q \in \mathcal{M}$ such that $\|p - q\| < 2\rho$, the following estimation holds [28, Lemma 3]:

$$\|p - q\| \leq d_{\mathcal{M}}(p,q) \leq 2\rho \arcsin\left(\frac{\|p-q\|}{2\rho}\right). \tag{3.1}$$

Let $T_p\mathcal{M}$ and $T_q\mathcal{M}$ denote the tangent planes at $p, q \in \mathcal{M}$. Lemma 6 in [28] gives an upper bound for the angle $\sphericalangle\left(T_p\mathcal{M}, T_q\mathcal{M}\right)$ between them,

$$\sphericalangle\left(T_p\mathcal{M}, T_q\mathcal{M}\right) \leq \frac{d_{\mathcal{M}}(p,q)}{\rho}. \tag{3.2}$$

Hence, Inequalities (3.1) and (3.2) imply for the normal vectors n_p at p and n_q at q that

$$\begin{aligned} \varphi &:= \sphericalangle\left(n_p, n_q\right) \leq 2\arcsin\left(\frac{\|p-q\|}{2\rho}\right) \\ \Rightarrow \|p-q\| &\geq r(\varphi) := 2\rho \sin\left(\frac{\varphi}{2}\right). \end{aligned} \tag{3.3}$$

For a given angle $\varphi_{\max} \in \left[0, \frac{\pi}{2}\right[$, the second part of Eq. (3.3) implies that there is a constant $r_{\max} \in \mathbb{R}_{\geq 0}$ such that $\varphi \leq \varphi_{\max}$ if $\|p-q\| < r_{\max}$. Denote by $\mathcal{M}' := B_{r_{\max}}(p) \cap \mathcal{M}$ the part of $\mathcal{M}$ that is contained in $\mathcal{M}$ and the ball $B_{r_{\max}}(p)$ centered at p. Then, the normals n_q of all points $q \in B_{r_{\max}}$ have positive Euclidean scalar product with n_p. The assumption that $\mathcal{M}$ is of positive finite reach guarantees that $\mathcal{M}'$ is a single connected component. Hence, $\mathcal{M}'$ has a parallel projection to the tangent plane $T_p\mathcal{M}$ without over-folds (Fig. 1). Furthermore, we have:

Lemma 3.1. *Let $p \in \mathcal{M}$ be a point with normal n_p. Then, for $r < \rho$, the image of $B_r(p) \cap \mathcal{M}$ under the projection π in direction of n_p to the tangent plane $T_p\mathcal{M}$ is a convex set.*

Proof. The intersection $\mathcal{I}$ of a closed set $\mathcal{S} \subset \mathbb{R}^d$ having reach $\rho_{\mathcal{S}} > 0$ with a closed ball $B_r(x)$, $r < \rho_{\mathcal{S}}$ and $x \in \mathbb{R}^d$, is geodesically convex in $\mathcal{S}$ [28, Corollary 1]. That is, the shortest path between any two points in $\mathcal{I}$ lies itself in the intersection. Furthermore, the intersection $\mathcal{M}'$ of $B_{r_{\max}}$ and $\mathcal{M}$ is a topological disk as established above [28, Proposition 1].

Here, $\mathcal{I}$ is not empty and consists of a surface patch since p lies on $\mathcal{M}$. Hence, the boundary $\partial\mathcal{M}'$ can be parameterized by a closed curve γ. As $\mathcal{I}$ is geodesically convex, γ has positive geodesic curvature. The inner product of the normals n_p and n_q at an arbitrarily chosen point $q \in \partial\mathcal{I}$ is positive: $\langle n_p, n_q \rangle > 0$, by choice of r. Therefore, under projection along N_p to the tangent plane $T_p\mathcal{M}$, the sign of curvature is preserved. Hence, the projection $\pi(\gamma)$ is a convex curve. □

Finally, let $\mathcal{G}$ be a simple graph that can be embedded on $\mathcal{M}$ such that the vertices in $\mathcal{G}$ connected by an edge have Euclidean distance d. The connected components remaining after removing $\mathcal{G}$ from $\mathcal{M}$ are called *regions*, denoted by $\mathcal{R}$. The set of vertices and edges incident to a region $R \in \mathcal{R}$ is called its *border*, denoted by ∂R. Note that because $\mathcal{M}$ is closed, each edge of $\mathcal{G}$ belongs to the border of exactly two regions. Also, each vertex of $\mathcal{G}$ can belong to the borders of several regions at once. Fix one such region $R \in \mathcal{R}$. Lemma 3.1 implies a choice of points $q_1, \ldots, q_k \in \partial R$ is mapped to points $\pi(q_1), \ldots, \pi(q_k) \in T_p\mathcal{M}$ in cyclic order, for $p \in R$ arbitrarily chosen. Hence, the regions can be extracted correctly with respect to their topology from the cyclic order of the edges at each vertex from the local projection. Given that the reach criterion is satisfied and given a suitable normal field, we can thus reconstruct a manifold from the input.

3.2. Methodology

Our algorithm extracts a mesh from the input $\mathcal{P}$ by placing touching spheres of a predefined diameter d across an approximation of the surface [5]. As the spheres touch, this will guarantee a minimum edge length of the mesh and by the results from Section 3, the resulting mesh will be manifold.

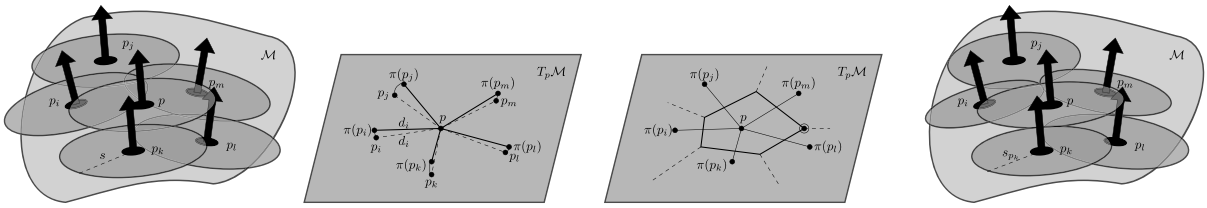
We assume to be given unstructured input in form of a point cloud $\mathcal{P} = \{p_i \mid i = 1, \ldots, n\} \subset \mathbb{R}^3$ with corresponding normals

$$\mathcal{N} = \{n_{p_i} \mid i = 1, \ldots, n\} \subset \mathbb{S}^2.$$

Furthermore, we assume that $\mathcal{P}$ is sampling an underlying, possibly itself unknown, manifold $\mathcal{M}$ with the properties as listed above. To approximate the surface, we associate to each point $p \in \mathcal{P}$ a *splat* S_p in the shape of a circular disk with radius $s_p \in \mathbb{R}_{>0}$. Each S_p is centered at the respective point p and placed such that S_p is orthogonal to the corresponding normal n_p. We assume the radii s_p to be chosen sufficiently large such that the manifold to be reconstructed is covered. Here, a manifold is said to be *covered*, if the union of projections of splats to the ground truth covers it. This is illustrated in Fig. 2(a).

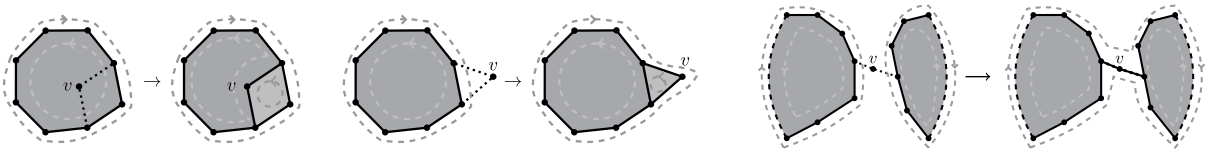
Note that following Lemma 3.1, the user has to choose the parameter d with respect to the reach ρ of the input, which can be estimated for point clouds [29], to ensure a manifold output. In this sense, choosing d is always a model-dependent choice of the user. If the model has, for instance, been scanned and the user has physical access to the model, a suitable value of d can be estimated based on the narrowest parts of the model. Also, the overall point distance in the input point cloud can serve as a means to approach a suitable value of d from below, for instance by taking the smallest point distance (or a small multiple of it) as value d. This results in an extremely fine-grained output, which might not be supported by the user's machine memory. Ultimately, it depends on the use-case scenario of the user how fine-grained they want the output. In Section 4.1, we will discuss a heuristic to iteratively estimate a suitable value for d from above, aiming for a coarse output that still satisfies the requirements formulated above.

Furthermore, the user also chooses a uniform initial splat size s. The individual splat sizes will be derived later as described in Section 4.3. In the following discussion, we will refer to elements of the input geometry $\mathcal{P}$ as *points* and to entities created by the algorithm as *vertices*.



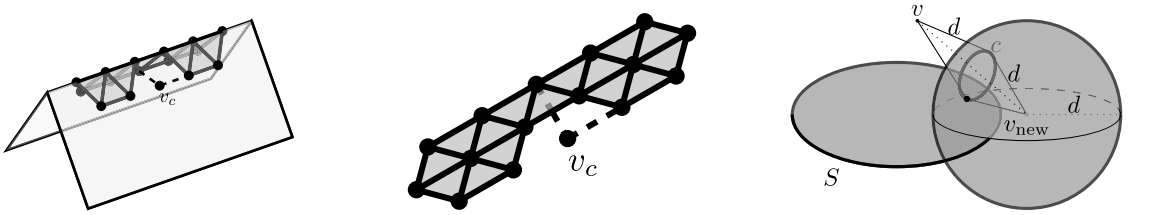
(a) Splats of radius s. (b) Projection to $T_p\mathcal{M}$. (c) Voronoi cells in $T_p\mathcal{M}$. (d) Individual splat radii.

Fig. 2. Illustration of uniform splat size 2(a), the projection of a point p and its vicinity to the tangent plane $T_p\mathcal{M}$ 2(b), and the Voronoi cells with the farthest point circled 2(c), leading to individual splat sizes 2(d).



(a) Connecting v by two edges to the same border. (b) Connecting v to two different borders.

Fig. 3. Possibilities when creating new vertices and edge connections in the graph $\mathcal{G}$: In 3(a), the new vertex v and its two edges connect elements of the same border. Here, v is created either in the in- or the outside region of the border. In 3(b), the new vertex v is connecting two borders. After introducing v and its edges, the respective outside regions are still connected, then v and its edges join the borders. However, if the outside regions are split by v and its edges, new borders are created which induce the corresponding regions.



(a) 3D view of a tent-like feature, a geometry with reach $\rho = 0$. (b) Projection of the graph along an approximated surface normal. (c) Intersection of circle c with splat S gives next vertex candidate.

Fig. 4. An input geometry with a vertex candidate v_c 4(a), the region projection shows an illegal edge crossing 4(b). Edges to parent vertices are shown dotted. Computation of vertex candidate positions 4(c).

3.3. Initialization

Aside from d and s, the user provides two *starting vertices* to initialize the algorithm. These vertices are chosen from $\mathbb{R}^3$ such that the projections of these vertices onto their closest splats are sufficiently close, that is, the distance between the projected vertices is in $[d, 2d]$, so a third vertex having distance d to both of the starting vertices can be placed by the algorithm. They do not have to be points from the input $\mathcal{P}$. The projected vertices form an initial vertex set of a graph $\mathcal{G}$ that will ultimately provide the manifold mesh discussed in Section 3.1. They can be manually provided or automatically created, for instance, around the maximum z-coordinate of the input. At this stage, $\mathcal{G}$ does not contain any edges. In the following, positions exactly d away from at least two already existing vertices are called *vertex candidates*. The two vertices that are d away from a candidate are its *parents*.

3.4. Disk growing to add vertices to $\mathcal{G}$

After initialization, disk growing is performed to create further vertices and edges for $\mathcal{G}$. A vertex is added from the list of vertex candidates, connected by edges to the two vertices distance d away (Fig. 5(a)), and new vertex candidates are added based on the newly placed vertex (Fig. 5(d)). Adding edges to $\mathcal{G}$ also changes the regions introduced in Section 3.1: By inserting a new vertex and its two edges, either one border is *split* into two borders or two borders *join* into one (Figs. 3(a) and 3(b)).

As shown in Fig. 7, there might be regions with comparably long borders. These lead to visible seams in both $\mathcal{G}$ and its triangulation. To avoid such seams, we prioritize joining borders over splitting borders. Thus, we aim to prioritize splits with a larger combinatorial distance between the parent vertices along the border over those with smaller distances. We do so with a priority assigned to the vertex candidates.

3.5. Prioritizing of vertex candidates

A vertex candidate v_c is chosen to become a vertex of $\mathcal{G}$ according to the following priorities, given in decreasing order:

1. At least one of the parent vertices has no edges incident to it. (Note: This parent vertex has to be a starting vertex from the initialization.)
2. At least one of the parent vertices is a vertex with only one edge incident to it.
3. Inserting v_c and its two edges joins two borders.
4. Inserting v_c splits a border—prioritize larger distance between parents along the common border.

In all cases, ties are broken by the breadth-first strategy. To determine the priority of the vertex candidate to be added, it is necessary to know to which of the parent vertices' borders the edges to be introduced will connect. To find the corresponding border, the candidate edge is projected to the plane defined by the parent vertex and its normal. Note that because of the results from Section 3.1, such a projection is possible

without over-folds. Given the prioritization of vertices, these can now be added to the graph $\mathcal{G}$.

3.6. Creating a new vertex

Once a vertex candidate v_c has been chosen, it is first determined whether there is a vertex v in $\mathcal{G}$ such that $\|v_c - v\|_2 < d$. If so, the vertex candidate is discarded.

Next, the priority of v_c is checked. In case the vertex does not satisfy the given priority anymore — for instance, because it was created with a parent vertex without any edges incident to it, but the parent vertex gained an edge by now — the vertex candidate's priority is reduced and another vertex candidate is chosen.

Adding v_c and the corresponding edges to $\mathcal{G}$ bears one additional problem. In practice, we do not always know whether the point cloud $\mathcal{P}$ fulfills the criteria listed in Section 3.2. If they are satisfied, the output is guaranteed to be manifold. However, the user might have chosen d too large or the input point cloud might not sample a manifold in the first place. In either of these cases, all edges created from new vertices still have an edge length of d, but the edges might create non-manifold connections. Consider Figs. 4(a) and 4(b) for an example of a surface with reach $\rho = 0$.

In these cases, we still want to prevent such faulty connections. Therefore, we find an approximated surface normal, which will be discussed in Section 4.1. We project v_c and its prospective edges as well as all edges already existing in the vicinity of v_c along this normal. For this projection, the vicinity of v_c is bounded by d in normal direction. We discard v_c if either of its edges crosses an already existing edge (Fig. 4(b)). While the algorithm creates manifold output for suitable input point clouds and choices of d, this mechanism improves the output even outside of this regime. If v_c has passed these checks, v_c and the two edges connecting it to its parent vertices are added to $\mathcal{G}$.

3.7. Triangulating the resulting regions

After the disk growing process has finished, the graph $\mathcal{G}$ provides a set of regions $\mathcal{R}$. On average, each vertex of $\mathcal{G}$ is connected to approximately four other vertices [5, Section 4.2]. Therefore, the average border length is approximately four. Hence, we are left with the task of triangulating these regions. In case of surfaces with boundary such as partial scans, we do not want to close the surface by triangulating the interior of the boundary. Therefore, we give the user the choice to specify a maximal border length $\partial_{\max}$ that will leave the region as a hole rather than triangulating it.

A region can be irregular in the sense that the inner angle of two consecutive edges can be larger than 180°. In such cases, a projection of a single region to a plane is not necessarily a convex polygon. These inner angles of the faces are found by projecting the edges onto a plane given by the vertex normal. Then, we triangulate each region by iteratively cutting away the smallest angle as this leads to triangles close to equilateral ones. Not only have we thereby created a triangulation of the input surface that has guaranteed minimum edge length d, but by the results provided in Section 3.1, provided that the input and the user-chosen parameters satisfy the restrictions made, the triangulation is also manifold.

4. Implementation

In this section, we will discuss implementation aspects of the algorithm presented above. In particular, this contains the introduction of data structures for efficient access. As stated in Section 3.2, we assume to be given a point cloud $\mathcal{P}$, its normal field $\mathcal{N}$, user-chosen parameters d, s, and in case of a surface with boundary, $\partial_{\max}$. If $\mathcal{P}$ does not come with a normal field, the user has to estimate one, for instance, via [30]. Furthermore, the user has to choose the implementation-related parameter w (Section 4.2).

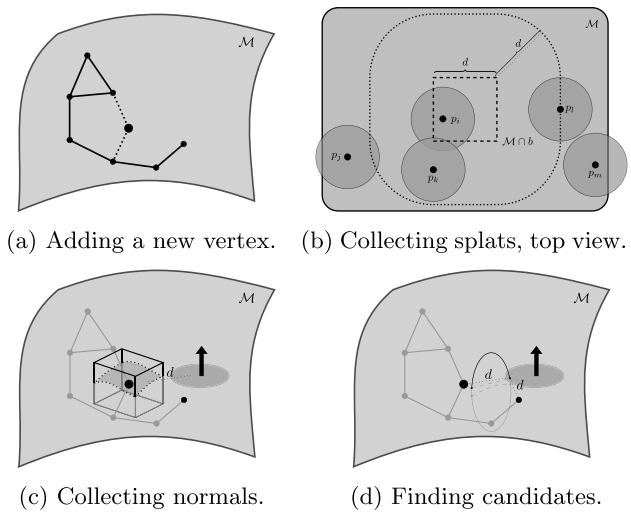


Fig. 5. Update steps of the algorithm.

4.1. Box grid data structure

When introducing new vertex candidates (Section 3.4), we need to know all splats close to a given, newly introduced vertex. In order to have access to these, we build a *box grid data structure* consisting of equal-sized, cubical boxes of side-length d partitioning the three-dimensional embedding space. Each box holds a pointer to those input points and their splats that are at most d away (Fig. 5(c)). This collection of points associated with box b_j is denoted by $\mathcal{B}_j$.

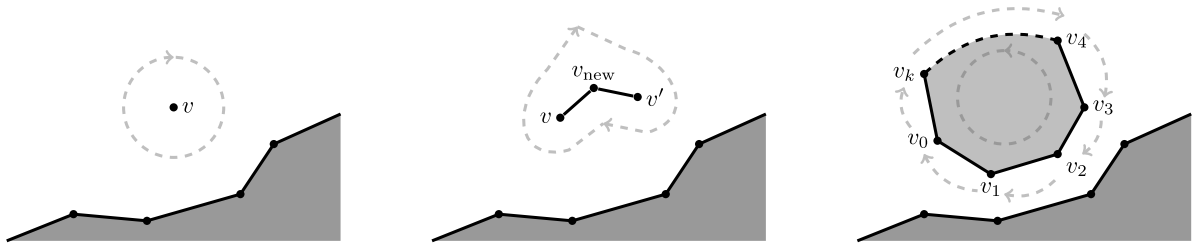
As preliminary filter step, we compute an average normal $\overline{n}_{b_j}$ for each box b_j by summing up the normals of all those points that b_j has a pointer to, without normalizing the sum. If $\|\overline{n}_{b_j}\|_2$ is smaller than 0.1, we keep all points in b_j. If the length is at least 0.1, we can assume that enough points agree on a normal direction in this box. Then, we remove those points p from the box for which $\langle n_p, \overline{n}_{b_j}\rangle < 0$. We choose a value of 0.1 for the length check to filter a small number of points while maintaining coherent normal information. This will ensure that the following step can succeed.

For each box b_j with at least one associated splat, we compute a *box normal* n_{b_j}. It will be used for projection steps that will ensure manifold properties of the resulting mesh. This will provide approximated surface normals that allow us to work on data which do not fulfill the requirements listed in Section 3.1 such as the example shown in Fig. 4(a). To compute an approximation efficiently, we take a finite sampling $S_F \subset \mathbb{S}^2$ and derive the box normal n_{b_j} as

$$n_{b_j} = \operatorname*{arg\,max}_{N \in S_F} \min_{p_i \in \mathcal{B}_j} \langle n_{p_i}, n\rangle \approx \operatorname*{arg\,max}_{n \in \mathbb{S}^2} \min_{p_i \in \mathcal{B}_j} \langle n_{p_i}, n\rangle.$$

That is, we search for the unit normal that maximizes the smallest scalar product with all point normals associated to the box b_j. For each box b_j, the scalar product $\langle n_{p_i}, n_{b_j}\rangle$ is ideally strictly positive for all points $p_i \in \mathcal{B}_j$, even in those cases where we did not filter the normals. Therefore, it allows for a projection onto a plane spanned by n_{b_j} as normal vector such that all points remain positively oriented by their normals. The newly computed box normal is also used as vertex normal for all vertices lying in b_j from now on.

To achieve a fast lookup, we can either build a uniform grid on the complete bounding box of the input or create a hash structure to only store those boxes that are including input points from $\mathcal{P}$. The uniform grid has faster access, but results in many empty boxes and thus large memory consumption. The hash structure does not use as much memory, but the access is slower. In our experiments, we utilize the uniform grid structure to be faster as memory consumption can be



(a) Border of length 0. (b) Border of length 4. (c) Border of length $k+1$.

Fig. 6. Different borders and their respective regions: Border of length 0 consisting of a single vertex and associated to a single, white, surrounding region 6(a); border of length 4, going back and forth between v and v', associated to a single, white, surrounding region 6(b); cycle of $k+1$ edges, separating the surface into an inner, light gray and outer, white region 6(c).

(a) Longest border (length 93). (b) Longest border (close-up). (c) Triangulated regions. (d) Triangulated region (close-up). (e) Longest border (length 11). (f) Longest border (close-up).

Fig. 7. Visible seams on the *Bottle Shampoo*. Figs. 7(a) to 7(d) show experimental results obtained by inserting new vertices via pure breadth-first-growing. The seams (shown in blue) appear as regions having a high number of border edges compared to all other regions on the surface. Figs. 7(e) and 7(f) show results obtained by prioritizing new vertex candidates. For better visibility, the target edge length d was chosen as 1. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

handled by our test machine, although the time difference will become significant only for larger models than used here.

Note that in Section 3, we stated that for creating new vertex candidates, we need to traverse all splats that are distance d away from a given point. However, here we are collecting all splats that are at distance $\leq d$ from the box, thus possibly resulting in a higher number of splats to be considered. That insures that the spheres from Section 3.2 are inscribed into the volume within d-distances around the boxes (Fig. 5(b)).

This leads to the question how to choose a good side-length of the boxes. As stated above, we use side-length d, that is, their size coincides with the target edge length for the triangulation. For smaller values, each splat would be associated to more boxes, hence the memory demand would grow. For larger boxes, there will be many splats associated to each box that do not actually lead to vertex candidates with the currently considered vertex. Thus, the runtime would grow when checking all splats being far away from the currently considered vertex. Finally, for larger boxes, it will also become more difficult to compute a suitable box normal for projections. Hence, we advocate for the middle ground and choose d as the box size.

The last observation regarding the box normals leads to a heuristic how to check the user's parameter choice of d. Namely, a choice of d is considered too large if there is a box whose box normal has negative scalar product with any point normal of a point registered in the box. This provides a mechanism to alert the user that they have chosen the parameter d outside of the specifications as provided in Section 3.1 and that the output is thus not guaranteed to be manifold anymore. This observation allows for the following binary search heuristic to find a suitable, yet not too small value of d (compare the discussion in Section 3.2): Start with a large initial value d_{init}, for instance, the bounding box diagonal of the geometry. Perform a binary search on the interval $[0, d_{\text{init}}]$ by picking the midpoint of the current interval, building the box grid data structure and evaluating the normal scalar products. If any is negative, continue the binary search in the lower half of the interval. If all are positive, the user could either stop, since a suitable value for d has been found, or continue in the upper half of the interval, if they seek for a coarser output.

4.2. Window size

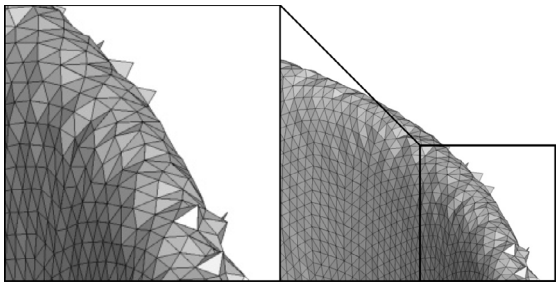
During the disk growing, we maintain a data structure representing the regions' borders. They consist of oriented half-edge cycles. Note that this includes degenerate cases, such as a single vertex, interpreted as a border of length 0 (Fig. 6(a)). Each time a new vertex and the two edges to its parent vertices are added to $\mathcal{G}$, this creates four new half-edges which have to be linked to the existing borders. To avoid traversing very long distances along the borders when computing priorities of vertex candidates, we introduce a *window size* w after which the traversal is stopped. A *window* then consists of $2w+1$ vertices on a common border, running in both directions centered at the vertex currently considered. This provides a considerable speedup compared to the previous solution [5].

Recall *split* and *join* from Section 3.4. Note that by cutting the traversal at a finite window size, it is not longer possible to distinguish between a split and join operation in all cases. Preliminary experiments showed that window sizes of $w \geq 8$ all produced the same quality output, despite not distinguishing splits or joins, as shown in the supplementary material [31].

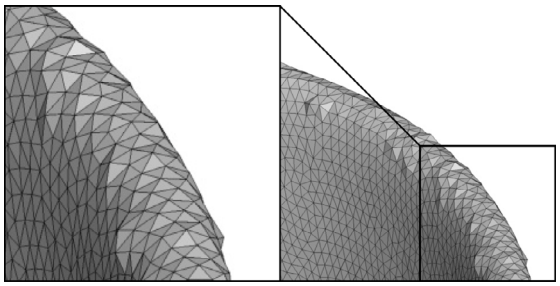
Furthermore, we experienced that setting the window size to $w = 0$ immediately creates noticeable negative effects on the result of the algorithm. In this case, our algorithm defaults to the *breadth-first* strategy of [5] and thus creates visible seams on the geometry (Figs. 7(a) to 7(d)). Starting from window sizes of $w = 2$ or $w = 3$, benefits in the quality of the output are apparent as larger visible seams are prevented. Theoretically, a larger window size will increase the lookup time. Therefore, in our implementation of the algorithm, we go for $w = 8$ as a large enough window size to reap its benefits, but a small enough one to not impact the algorithm's run time.

4.3. Discussion of splat size

In case of non-uniform sampling density, using a global splat size s might lead to areas covered multiple times. For more densely sampled areas, a smaller splat size guarantees the creation of vertices closer to the sampling points. In our experiments, we saw that in high-curvature regions, smaller splats have small deviation from the surface,



(a) Result with uniform splat sizes.



(b) Result with individual splat sizes.

Fig. 8. Running our algorithm on the *Bowl Chinese* from [11] with 8(a) using a global splat size and 8(b) using a local spat size. The letter reduces artifacts in high-curvature regions such as the rim of the bowl.

while larger splats deviate from the surface significantly. Hence, when looking for vertex candidates on large splats, the algorithm can place vertices that are somewhat distant to the input points (Fig. 8(a)). Therefore, we turn to individual, smaller splat sizes to reduce the deviation of the vertices with respect to an underlying surface represented by the input point cloud.

An additional benefit is that for smaller splat sizes, there are less splats registered per box, which speeds up the algorithm. However, the individual splats have to have sizes sufficient to cover the underlying geometry. To find the specific splat size s_p for each point $p \in \mathcal{P}$, we use the box data structure (Fig. 2(d)). We consider all points p_i associated to the box containing p. To map the points $\{p_i\}$ to $T_p\mathcal{M}$, consider the plane $N_\perp$ containing p and p_i and being orthogonal to $T_p\mathcal{M}$. For each p_i, an auxiliary point $\pi(p_i)$ is determined by rotating p_i around p around the smaller angle in $N_\perp$ until it lies in $T_p\mathcal{M}$. Hence, p and $\pi(p_i)$ have the same distance d_i as p and p_i have. Based on a cyclic sorting around p, we compute a central triangulation, connecting all projections to p and connecting them pairwise according to their angular sorting (Fig. 2(b)). For the resulting triangulation, we test whether or not we can flip a central edge to make the incident triangles Delaunay. Points p_i, whose edges are flipped, are removed from the following consideration. For those neighboring points that remain, consider the Voronoi diagram of their triangulation. We choose the local splat size s_p as distance from p to the farthest Voronoi vertex (Fig. 2(c)). This ensures that all Delaunay triangles are still completely covered. By choosing local splat sizes in this way, the visible deviation from the underlying geometry is reduced (Fig. 8(b)).

4.4. Processing vertex candidates

The processing of vertex candidates, following Section 3.4, consists of the following steps: popping a vertex candidate from the priority queue, checking feasibility of the candidate, adding a suitable candidate as well as its edges to $\mathcal{G}$, and adding new vertex candidates to the priority queue.

Because of the window size w, there is a finite number of priorities, as given in Section 3.5. Each of these priorities is handled via its own queue that follows a strict first-in-first-out strategy, which enables popping of candidates in constant time [32, Chapter 2.4].

If a vertex still has correct priority, checking for conflict with existing vertices and performing the projection check from Fig. 4(b) both requires access to nearby vertices. This is a constant-time operation because of the box data structure that holds all relevant vertices. Furthermore, the number of vertices within distance $2d$ is, by construction, bounded from above by the densest sphere packing in space, which is a constant.

Once a new vertex v_{new} is created, we compute new vertex candidates having v_{new} as parent vertex. Therefore, we need the set of all splats intersecting the ball of radius d centered at v_{new}. This is a subset of the set of those splats associated to the box containing v_{new}. To efficiently access potential second parent vertices, we maintain for each splat S a list of all vertices within distance d to S (Figs. 5(b) and 5(c)).

5. Experiments

This section is devoted to different experimental settings. As described in the beginning, we aim for the reconstruction of real-world scan data. When performing the comparison of different algorithms, we do so based on a quantitative analysis of the obtained triangle mesh $\mathcal{T}$. For this, given a triangle $t \in \mathcal{T}$, we denote the lengths of its edges by $\ell_{t,1}$, $\ell_{t,2}$, and $\ell_{t,3}$. The area of the triangle will be called A_t.

Following the approach in [33, p. 307, Eq. (13)], we measure the quality Q_t of a single triangle as

$$Q_t = \frac{4\sqrt{3}A_t}{\ell_{t,1}^2 + \ell_{t,2}^2 + \ell_{t,3}^2}.$$

This measure corresponds to a scaled version of the *scale-invariant (smooth)* conditioning quality measure discussed by Shewchuk [34, Table 3]. Based on the local, triangle-based measure Q_t, further following [33], we present a global metric for the entire triangle mesh $\mathcal{T}$ as average over the quality of the triangles, that is

$$Q_{\text{avg}} = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} Q_t.$$

Note that the factors normalize this quality metric to be 1 for equilateral triangles and close to 0 for very narrow slivers. Finally, we compute the root mean square deviation in percent Q_{RMS} as

$$Q_{\text{RMS}} = \frac{100}{Q_{\text{avg}}}\sqrt{\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\left(Q_t - Q_{\text{avg}}\right)^2}.$$

See Section 3 of [34] for a relation of this quality measure to the stiffness matrix. Furthermore, from the set of all edges in the triangulation, we consider the average edge length E_{avg} as well as the corresponding root mean square deviation E_{RMS}, also in percent.

In order to demonstrate the quality of the meshes achieved by our algorithm, we turn to 20 scanned objects provided as part of a surface reconstruction benchmark [11]. Here, we concentrate on high-resolution scans obtained by an *OKIO 5M* scanning device, resulting in 330k to 2000k points per surface after 20 shots. The shots are registered and do come with a normal field. Out of the 20 point clouds, we used 19 as they are provided in the repository. The scan of a remote control had a clear registration artifact, since one of the buttons of the remote was registered into the remote, pointing down, not up. This, we corrected manually by removing the wrongly registered points. Here, we compare our results to those made by various widely used algorithms from the field. Then, we add different levels of noise to the data and investigate the stability of our algorithm.

As mentioned in Section 4, there is a set of parameters which has to be chosen by the user. For our experiments, we made the following choices. The sphere diameter d was set to be 0.2, while the maximal border length $\partial_{\max}$ was equal to 40. For each model, the initial splat size s was chosen between 0.2 and 0.4 individually, depending on the considered point cloud.

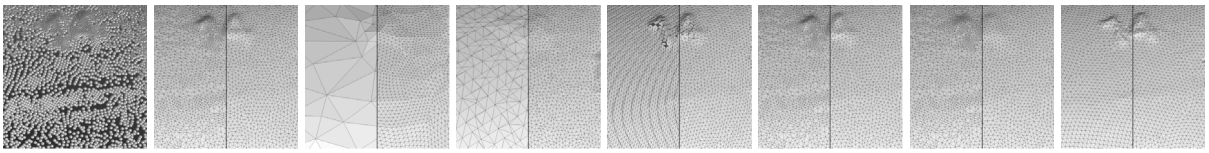


Fig. 9. Qualitative comparison of named algorithms without remeshing (left) and with remeshing (right).

Table 1
Bottle Shampoo (604,903 input points).

Algorithm	$\lvert\mathcal{T}\rvert$	E_{avg}	E_{RMS}	Q_{avg}	Q_{RMS}
Adv. Front	1.209546	0.1799	39.6	0.8247	16.0
Adv. Front (Re)	928,850	0.2028	15.3	0.9416	6.1
Poisson	16,280	1.2946	74.8	0.8760	12.3
Poisson (Re)	498,140	0.2657	38.6	0.9251	7.5
Poisson MG	150,770	0.5318	35.7	0.7204	33.7
Poisson MG (Re)	952,830	**0.2015**	16.3	0.9330	7.0
RIMLS	1,907,781	0.1499	35.8	0.7055	35.1
RIMLS (Re)	1,054,438	0.1905	19.3	0.9117	11.5
Scale Space	1,209,093	0.1798	39.1	0.8248	16.0
Scale Space (Re)	926,828	0.2028	15.2	0.9417	6.0
Voronoi	1,209,792	0.1799	52.3	0.8241	16.1
Voronoi (Re)	923,476	0.2044	20.8	0.9407	6.8
Ours	840,453	0.2131	**11.2**	**0.9577**	**4.5**
Ours (Re)	854,257	0.2098	**10.4**	**0.9701**	**3.8**

Table 2
Bowl Chinese (606,320 input points).

Algorithm	$\lvert\mathcal{T}\rvert$	E_{avg}	E_{RMS}	Q_{avg}	Q_{RMS}
Adv. Front	1,212,636	0.2920	38.2	0.8045	18.6
Adv. Front (Re)	2,407,002	0.2038	15.4	0.9405	6.2
Poisson	13,584	2.3850	63.2	0.8845	11.7
Poisson (Re)	637,488	0.3732	40.8	0.9301	6.9
Poisson MG	503,458	0.4710	39.8	0.7062	37.1
Poisson MG (Re)	2,409,076	0.2050	17.7	0.9223	7.9
RIMLS	6,458,589	0.1331	40.4	0.6877	39.6
RIMLS (Re)	2,441,143	0.2023	15.4	0.9394	6.3
Scale Space	1,093,339	0.2779	34.9	0.8054	18.7
Scale Space (Re)	1,947,592	**0.2006**	16.1	0.9351	7.3
Voronoi	1,212,636	0.2916	38.4	0.8042	18.7
Voronoi (Re)	2,398,584	0.2039	15.3	0.9405	**6.1**
Ours	2,137,650	0.2167	**14.8**	**0.9485**	6.2
Ours (Re)	2,246,434	0.2093	**11.4**	**0.9665**	**4.3**

Table 3
Cloth Duck (1,018,891 input points).

Algorithm	$\lvert\mathcal{T}\rvert$	E_{avg}	E_{RMS}	Q_{avg}	Q_{RMS}
Adv. Front	2,037,574	0.1839	40.1	0.8143	17.3
Adv. Front (Re)	1,739,214	0.1965	19.4	0.9179	9.5
Poisson	147,940	0.6300	44.3	0.8805	12.0
Poisson (Re)	1,488,112	0.2068	17.5	0.9311	7.2
Poisson MG	419,614	0.4086	38.5	0.7160	36.0
Poisson MG (Re)	1,463,018	0.2093	18.7	0.9154	8.8
RIMLS	5,878,521	0.1154	39.9	0.6919	38.9
RIMLS (Re)	1,728,371	**0.1978**	20.1	0.9143	12.7
Scale Space	2,036,816	0.1839	40.0	0.8139	17.4
Scale Space (Re)	1,735,814	0.1965	19.3	0.9179	9.5
Voronoi	2,037,270	0.1767	41.8	0.8067	18.1
Voronoi (Re)	1,514,160	0.2027	**15.4**	0.9407	**6.3**
Ours	1,435,604	0.2181	15.7	**0.9454**	6.6
Ours (Re)	1,535,058	0.2089	**12.5**	**0.9592**	**4.8**

Table 4
Toy Bear (607,501 input points).

Algorithm	$\lvert\mathcal{T}\rvert$	E_{avg}	E_{RMS}	Q_{avg}	Q_{RMS}
Adv. Front	1,214,998	0.1474	36.3	0.8474	13.9
Adv. Front (Re)	629,138	0.2024	15.2	0.9418	6.0
Poisson	20,134	1.0381	54.7	0.8882	11.7
Poisson (Re)	530,374	0.2193	22.8	0.9293	7.3
Poisson MG	432,268	0.2585	39.5	0.2623	37.1
Poisson MG (Re)	629,508	**0.2021**	15.0	0.9436	6.1
RIMLS	5,548,226	0.0730	40.0	0.6910	39.3
RIMLS (Re)	618,531	0.2049	16.2	0.9322	6.8
Scale Space	1,214,990	0.1474	36.3	0.8474	13.9
Scale Space (Re)	628,848	0.2025	15.2	0.9417	6.0
Voronoi	1,214,996	0.1471	36.5	0.8471	13.9
Voronoi (Re)	616,160	0.2041	15.0	0.9427	5.9
Ours	555,490	0.2159	**13.5**	**0.9499**	**5.6**
Ours (Re)	578,730	0.2096	**11.5**	**0.9657**	**4.3**

5.1. Experimental comparison for point cloud meshing

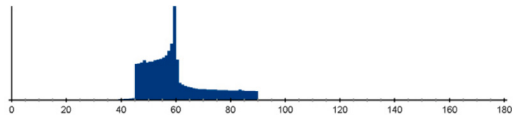
From the algorithms listed in Section 2, Poisson [12], advancing front [16], and scale space [17] are run with the standard parameters as implemented in [35] except for the cleaning steps, which were unnecessary because of the high-quality input. Multigrid Poisson [14] and Voronoi reconstruction [19] are run with the standard parameters as implemented in [36]. RIMLS [18] is run with the standard parameters from [15], using a smoothness of 2 and a grid resolution of 1000.

We aim for an algorithm that provides high-quality triangulations out-of-the-box, right after reconstruction. However, as the comparison algorithms do not necessarily optimize for a uniform edge length, we take their respective results and process them with the "Isotropic Explicit Remeshing" filter of MeshLab [15]. This filter repeatedly applies edge flip, collapse, relax, and refine operations [20]. We run three iterations with a target edge length of 0.2 in absolute world units for the input [11].

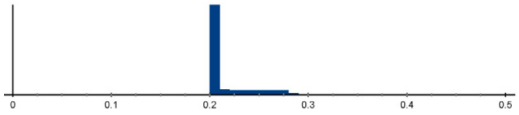
In Tables 1 to 4, we report both the results of the comparison algorithms and the result after these have been remeshed, indicated by "(Re)". These tables include representative models. A full report with data for all 20 models can be found in the supplementary material [31]. We chose the *Bottle Shampoo* and the *Bowl Chinese* because of their features, as explored in Figs. 8 and 9. The *Cloth Duck* is one of two models where the competing methods had the largest gain on our algorithm when measured by E_{avg} (see supplementary material [31] for the *Mug*).

A first thing to notice when regarding the results presented in Tables 1 to 4 is that our algorithm achieves the best, that is, highest values for Q_{avg} on all models. This holds consistently across all 20 models from the repository. That is, our method produces the highest quality meshes, even when compared with the remeshed results of the other algorithms. For comparison, we also add the remeshed version of our algorithm, which generally improves the quality metrics slightly while destroying the minimum edge length guarantee. The goal of this paper is not to compare different remeshing approaches, but to present a method that can provide high-quality triangle meshes right after reconstruction, without remeshing. Hence the remeshed version of our algorithm is set apart in gray and carries bold font if it causes an improvement on the previously best result. In this setting, the comparison to the remeshed results just serves to place our results in a broader setting.

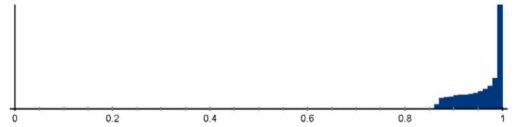
On most of the models, the deviation Q_{RMS} has also the lowest percentages for our algorithm. Notable exceptions are the *Bowl Chinese* (Table 2) and the *Cloth Duck* (Table 3). However, across all



(a) Angle distribution, target=60°.



(b) Edge lengths distribution, target=0.2.



(c) Distribution of quality Q_t, target=1.0.

Fig. 10. Distributions of the *Bottle Shampoo* as obtained by our algorithm (without remeshing).

models, the lowest deviation Q_{RMS} is at most 0.6% better than ours, cf. supplementary material [31].

Regarding the second metric, note that by construction, all edges produced by our algorithm are of length ≥ 0.2. Therefore, the average edge length is also always greater than 0.2, which places the remeshed output of other methods in the lead regarding the metric E_{avg}. However, the largest average edge length across all models is 0.2181 for our algorithm, attained on the *Cloth Duck* (Table 3), which is still very close to the target edge length.

Also, for almost all models, the width of the distribution of edge lengths, measured by E_{RMS}, is the lowest for our algorithm. That is, the triangulations produced are almost uniform. As a final observation regarding the quality metrics, note that those comparison algorithms that provide better metrics on the models do so only after an additional remeshing step. This shows that our algorithm does attain the goal of providing high-quality meshes immediately after reconstruction as it beats all comparison algorithms in this regard.

When inspecting the models visually, it is clear that, at least after remeshing, the triangulations are of high quality (Fig. 9). Note how some algorithms are not able to reproduce small details—for instance, a number 14 on the *Bottle Shampoo*. Even in the remeshed version, line-like artifacts are still visible for some of the comparison algorithms. Our algorithm creates a mesh close to uniformity while retaining the details.

This uniformity can be observed by plotting histograms on the distribution of angles, edge lengths, and quality measures for a triangulation obtained by our algorithm. See Fig. 10 for a corresponding set of plots for the *Bottle Shampoo* and find histograms for the other models in the supplementary material [31]. The histogram confirms that the angles of the triangles are centered around 60°, indicating a strong tendency towards equilateral triangles. Also, we see that the edge lengths are indeed starting from the set minimum of 0.2, with most edges actually attain this value. Finally, the histogram of the triangle quality reveals that there are many equilateral triangles (corresponding to $Q_t = 1$), with the distribution skewed towards this highest quality value.

Unlike some competitors and the remeshing step, our algorithm is not iterative but produces the output in a single sweep over the input. Run times for several models are given in Fig. 11, where the competitors are reported including the remeshing time. All experiments were run on a machine with an Intel® Core™ i7-5600U CPU 2.60 GHz with four cores and 16 GB of RAM. Five of the models did not fit the RAM of this comparison machine. Thus, we only provide timings for 15 models, while the qualitative data for the remaining five was acquired on another machine. Note that our algorithm performs similarly to most of the competitors.

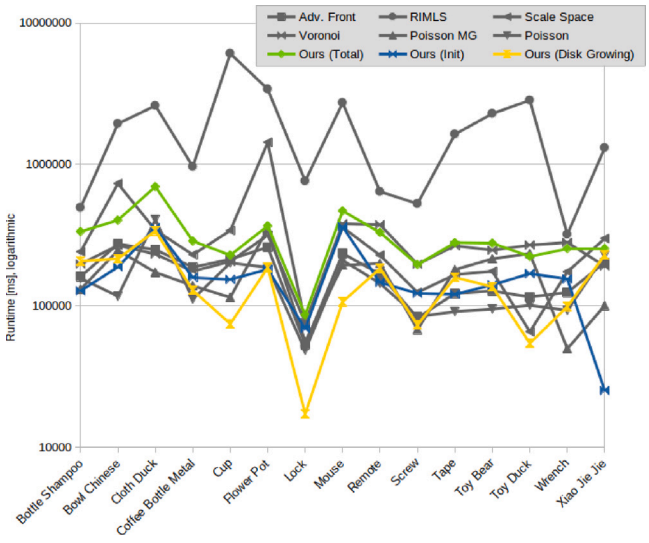


Fig. 11. Log of the run time of the algorithms on several models. Ours is additionally split into initialization and disk growing.

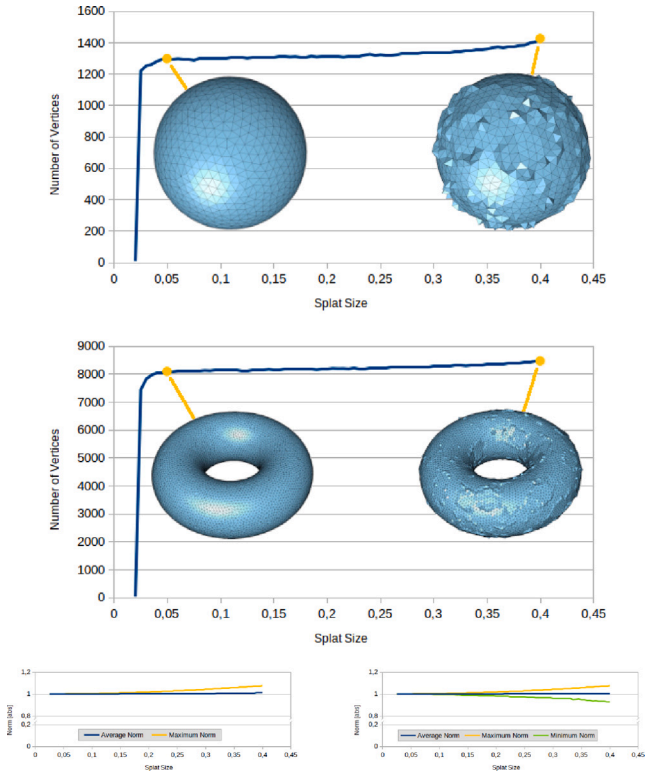


Fig. 12. Measuring the reconstruction quality.

5.2. Robustness to user input

As stated in Section 3.3, the user is asked to provide two starting vertices to run the algorithm. To investigate whether the quality of the obtained mesh is independent of the chosen starting vertices, we selected the *Toy Bear* because of its various differently curved regions. Further models promoting this observation are included in the supplementary material [31]. As illustrated in Fig. 13, we chose eight different regions to place the starting vertices in. The results of these experiments show that the quality of the output is not sensitive to the choice of starting vertices. For the eight resulting triangle meshes, the average

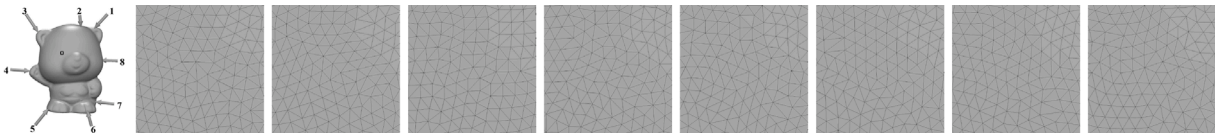


Fig. 13. Results with various starting vertices. From left to right: The *Toy Bear* with positions for starting vertex pairs, close up for pairs 1 to 8, showing the area at one eye emphasized in the first image.

edge length varies from 0.2158 to 0.2159, as does the average quality: from 0.9499 to 0.9504. In all cases, E_{RMS} is equal to 13.5 while Q_{RMS} equals 5.6.

Next, we investigate the robustness of surface reconstruction depending on the splat size. As the models discussed so far are real-world scans, there is no ground truth to compare the reconstruction with. For this experiment, we turn to two models that satisfy all assumptions made in Section 3.1 and that have an explicit mathematical parametrization to evaluate the reconstruction: the unit-sphere and a torus parametrized as a unit circle swept around a circle of radius 2. We sample both models randomly, the sphere with 10,000 and the torus with 60,000 points, resulting in a similar density on the models. The norm is a direct measure of the reconstruction quality. For the sphere model, vertices with norm 1 lie directly on the sampled sphere. For the torus model, we measure the norm as the distance to the circle of rotation, hence, a vertex with norm 1 lies directly on the sampled torus. In this scenario, the sphere diameter d was chosen as 0.1 while a global splat size was chosen between 0.02 and 0.4.

For both models, given too small splat sizes, the algorithm fails to cover the entire model, resulting in a very small number of vertices. Once a splat size is reached for which the entire model is covered, both the number of vertices and the reconstruction quality are stable until larger splat sizes are reached, which causes visible distortion in the reconstructed models (Fig. 12). This shows that for splat sizes, just large enough to cover the geometry, our algorithm achieves close to optimal reconstruction results. For the sphere model, all points created are on or outside the sphere, placing the closest vertex at a norm of 1 directly on the sphere. On the torus model, points are lying both in and outside of the torus. Even for the largest splat size of 0.4, which creates visible reconstruction artifacts, the reconstructed models are still manifold, in line with our guarantees from Section 3.1.

5.3. Robustness to noise

In order to investigate the robustness of our algorithm with respect to noisy data, we equip a selection of the high-quality models [11] with different levels of noise $\nu \in \mathbb{R}_{\geq 0}$. Here, we use those models that allow for placing moderate noise, for instance, the *Bottle Shampoo* or the *Bowl Chinese*, whereas we ignore those that already have details and elements that hinder manifold reconstruction even for tiny levels of noise. Thereby, each input point is moved by a uniformly distributed random vector of length smaller or equal to ν. This moves the noisy points within a bound of $\pm\nu$ around the ground truth. To measure the quality of the output, for each level ν of noise, we computed a triangulation based on the same parameter choices as above (Section 5). We find that the level of noise directly influences the number of vertices, similar to the observations made while increasing the splat size (Section 5.2). That is, with increasing noise level, the number of vertices of the output increases as well. Depending on the model, we experience that for values $\nu \in [0.06, 0.1]$, the output begins not to be manifold anymore. That is, manifoldness is lost from 2ν between 60% to 100% of d. For the user, this experiment suggests that for a geometry with known or estimated noise level ν, choosing $d \geq 2\nu$ yields the best results (see Table 5).

Table 5
Noise levels ν for which the reconstruction is (✓) or is not (✗) manifold.

Name \ ν	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
Bottle Shampoo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Bowl Chinese	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Cup	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
Flower Pot 2	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
Toy Bear	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
Toy Duck	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗

Table 6
Experimental results for remeshing *Kitten*.

	d	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{T}\rvert$	$\alpha_{\min}$	$\alpha_{\max}$	α_{avg}
input	–	10,000	20,000	7.6919°	153.3379°	60°
remesh	0.02	17,673	35,346	30.2108°	112.2197°	60°
remesh	0.03	7,837	15,674	28.3165°	115.9218°	60°
	$E_{\min}$	$E_{\max}$	E_{avg}	$Q_{\min}$	$Q_{\max}$	Q_{avg}
	0.0039	0.1119	0.0287	0.2296	0.9999	0.8438
	0.02	0.0391	0.0213	0.6742	1.0000	0.9558
	0.03	0.0613	0.0320	0.6391	1.0000	0.9552

6. Extension to surface remeshing

Here, we modify the ansatz presented in Sections 3 and 4 to remesh polyhedral surfaces to achieve an isotropic triangular mesh with guaranteed smallest edge length. To guarantee the theoretical results collected in Section 3, we assume the input polyhedral surface $\mathcal{F}$ to represent an orientable, closed, and compact C^2-manifold embedded into $\mathbb{R}^3$, which is of finite, positive reach ρ. The strict definition from Section 3.1 would give a reach of $\rho = 0$ for polyhedral surfaces. However, several methods are available to compute an approximation of an idealized surface that the polyhedral input is assumed to represent, where a user-given parameter steers how closely the input should be taken into account [37].

In Section 3.2, we equipped the input point cloud with circular splats on which the output surface is built. Now, turning to polyhedral surfaces, we can skip the splats and place the spheres on the faces of the polyhedral surface directly. Consequently, we use the face normals instead of splat normals for further calculations. This also removes the need for finding individual splat radii. To build the box data structure described in Section 4.1, to each box b_j, we associate all faces of the input with distance less than d to b_j. Afterwards, we run the algorithm as described in Sections 3.3 to 3.7 on the faces of the input geometry. Hence, the resulting surface interpolates the input one.

We illustrate the applicability of our algorithm to remesh a freeform mesh by running it on the *Kitten* model. Here, we run our algorithm with target edge length d varying between 0.01 and 0.08. Since the input consists of 10,000 vertices and has an average edge length of 0.0287, we consider the results achieved for $d = 0.02$ and $d = 0.03$ in more detail. As shown in Table 6, for both target edge lengths, the mesh quality is improved. Both, the resulting edge lengths as well as the resulting angles are less widely distributed than provided by the input mesh. Further, the model remeshed has a better average quality Q_{avg} equal to 0.9558 for $d = 0.02$ or to 0.9552 for $d = 0.03$, respectively, in comparison to 0.8438 of the input model. The meshes resulting for the

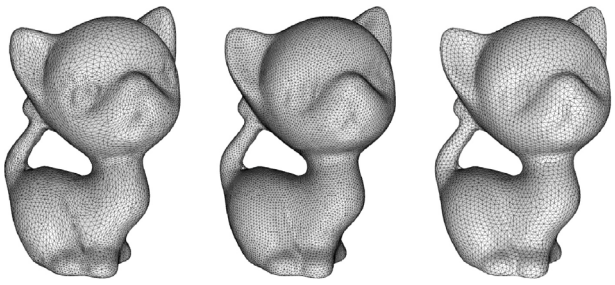


Fig. 14. Result of the algorithm run on *Kitten* model. Left: input geometry. Middle: input geometry remeshed with target edge length 0.02. Right: input geometry remeshed with target edge length 0.03.

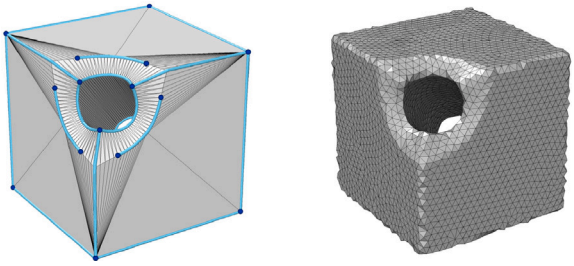


Fig. 15. Result of the algorithm run on CAD model. Left: the input geometry with detected features (shown in blue). Right: the remeshed output with more nearly regular triangles, but also a loss of features. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

mentioned target edge lengths are shown in Fig. 14. The complete data set can be found in the supplementary material [31].

Also for the CAD model shown in Fig. 15, the modified algorithm returns a remeshed triangular polyhedral surface. The result has more triangles than the input and the resulting triangles are closer to regular triangles than those of the input geometry. The improved triangle quality does come with a caveat. Since we place the spheres such that a newly introduced one touches two spheres already placed, the sphere centers — which later form the vertex set of the resulting mesh — do not necessarily coincide with input vertices nor are they likely to lie on edges of the input surface. Hence, features like ridges, as shown in Fig. 15, are worn off. In the next section, we therefore discuss how to further modify the algorithm to maintain features on both polyhedral surfaces and point clouds.

7. Feature detection

In this section, we focus on retaining features on point clouds and polyhedral surfaces throughout our algorithm. To define and to detect features, we use the dihedral angle α formed by the normals of intersecting splats introduced in Section 3.2. In case of a polyhedral surface input, we use the dihedral angle α formed by adjacent faces. In either case, we additionally introduce an angle threshold ϑ chosen by the user. This threshold defines a lower bound such that every intersection with $\alpha > \vartheta$ will be considered to be a feature and thus to be retained throughout the remeshing.

7.1. Feature detection on point clouds

We first turn to input point clouds and will discuss polyhedral surfaces as input subsequently. Consider two splats S and S' intersecting in a straight line segment ℓ; we say that ℓ is a *feature segment*, if α is larger than the threshold ϑ (see Fig. 17). In Section 8, we will illustrate the connection between the input geometry and ϑ experimentally.

For two splats, the length of the resulting feature segment ℓ is at most equal to the diameter of the splat with smallest radius, depending on the individual splat sizes. To avoid such — possibly very short — feature segments, we use the global splat diameter during feature segment detection, in contrast to Section 4.3. This leads to an increasing number of intersecting splats as larger splats are more likely to intersect other splats. The features detected by a given splat size do depend on various properties such as, for instance, the noise level, the chosen value of α, or the feature scales of the input geometry. Therefore, choosing a suitable splat size is highly dependent on the input settings at hand.

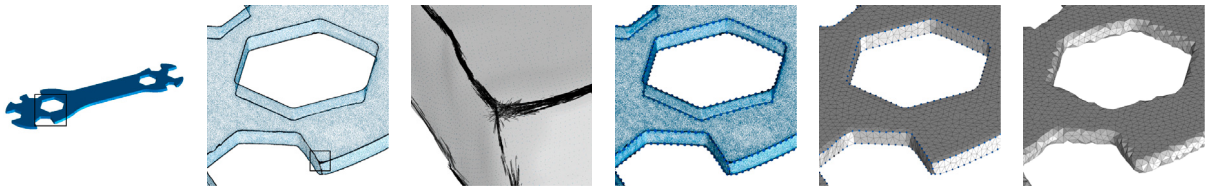
As depicted in Figs. 16(b) and 16(c), after collecting all feature segments, regions of aligned feature segments occur. Similar to the disconnected splats representing the input point cloud, the feature segments are a disconnected representation of features of the surface underlying the input. To structure this further, we collect a set of *feature vertex candidates*, which is formed by all intersection points of three splats.

For initialization of our algorithm, we construct the graph $\mathcal{G}$ based on the detected features of the input geometry. First, we sequentially add all feature vertex candidates to $\mathcal{G}$. However, we only add a feature vertex candidate, if it is at least distance d away from previously added ones to maintain the minimal edge lengths, otherwise, the feature vertex candidate is discarded. Next, we grow feature lines by iteratively looking for a position on a feature segment at distance d from an already existing feature vertex v. Again, these positions have to be at least distance d away from all other previously placed feature vertices. As long as we find such positions, we place a feature vertex there and connect it to the parent vertex v to which it has distance d. The connecting edges between two feature vertices are called *feature edges*. Hence, in contrast to the initialization described in Section 3.3, $\mathcal{G}$ does not contain solely two starting vertices, but vertices and edges representing the features of the input geometry. In this way, feature lines can be grown similar to the disk growing process.

On a point cloud, not all feature segments might be close enough to a feature line evolving from a previously placed feature vertex such that spheres are placed on them being connected to a feature vertex. Therefore, after the growing process described above, we iterate through all feature lines that we have not yet placed a feature vertex on. If we can still do so, according to the distance criterion to all other vertices, we place a feature vertex and continue the growing process from there. Once all feature segments are sampled by vertices, we look for pairs of vertices with valence 1 in the graph, which have a distance smaller than $2d$ to each other, and connect them by an edge. This closes gaps in $\mathcal{G}$, where vertices were added iteratively starting from both ends on a feature line of the geometry. The positions of the two vertices connected by such a closing edge are moved slightly towards the center of the closing edge to avoid edges on feature lines with length close to $2d$. Finally, after having processed all feature lines, we perform disk growing based on $\mathcal{G}$ as described in Section 3.4.

7.2. Feature detection on polyhedral surfaces

In contrast to an input point cloud, on a polyhedral surface input, we already have a set of edges. As stated above, an edge e in the input is called a *feature segment* if $\alpha > \vartheta$. Similarly, a vertex of the input is said to be a *feature vertex* if it is incident to either one feature segment or to at least three. In Fig. 15, the features detected for $\vartheta = 40°$ are shown. Feature vertices are depicted as dark blue dots. All the other end points of feature segments are incident to exactly one other feature segment. Hence, the feature segments form feature lines consisting of consecutive feature segments connecting the feature vertices. In case, a feature segment is not yet contained in such a feature line, it is part of a closed



(a) Input point cloud with highlighted sector. (b) Collection of feature edges (black), $\vartheta = 60°$. (c) Collection of feature edges (close up). (d) Feature vertices distributed on feature segments. (e) Meshed point cloud with feature detection. (f) Meshed point cloud without feature detection.

Fig. 16. Feature detection on the *Wrench* model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

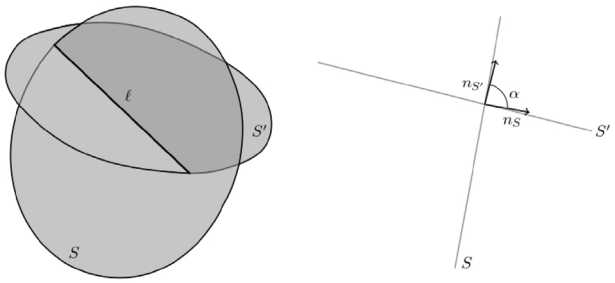


Fig. 17. Intersection of two splats S and S' and resulting feature segment ℓ. Side view of intersecting splats.

Table 7
Evaluation of one-sided Hausdorff distance from point cloud to approximating surface, without and with feature detection (f.d.) in comparison to input point cloud.

Model	d	ϑ	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{T}\rvert$	$d_{\max}$	d_{avg}	d_{RMS}
Wrench	1.0	–	8946	17,896	0.6630	0.0360	7.4933
with f.d.	1.0	60°	8871	17,746	0.4353	0.0112	2.1373
Screw	0.6	–	13,411	26,842	0.4605	0.0438	6.0548
with f.d.	0.6	90°	13,472	26,988	0.3595	0.0190	2.3354
Xiao Jie Jie	0.8	–	22,584	45,164	0.7918	0.0312	4.0479
with f.d.	0.8	70°	22,180	44,364	0.6131	0.0269	3.1905
Lock	0.8	–	5,951	11,902	0.4609	0.0167	3.2285
with f.d.	0.8	60°	5,837	11,674	0.2963	0.0113	1.6142
Remote	0.8	–	22,595	45,186	0.7816	0.0296	5.6993
with f.d.	0.8	50°	22,239	44,474	0.4880	0.0179	2.9222

feature line. This allows for their reconstruction as shown in Fig. 25. On each feature line, we distribute feature vertices with distance d until no further vertex can be placed. In order to distribute the sphere centers more regularly, feature vertices are moved iteratively on the feature lines. Finally, after having processed all feature lines, we perform disk growing based on $\mathcal{G}$ as described in Section 6. Figs. 18 and 19 show the results achieved after running the algorithm on a CAD model. Further evaluations of the output achieved by the algorithm presented here are included in the supplementary material [31]. Here, complete data sets in addition to graphical representations of the models used in this paper as well as models not presented so far can be found.

8. Experiments on point clouds

From the set of models provided in [11], we run the modified version of the algorithm on those models possessing feature ridges like the *Wrench*, the *Screw*, and the *Xiao Jie Jie*. Complete data sets of the models listed and others are provided in Table 7. As illustrated in Fig. 16, there is a visually striking improvement made. The *Wrench* model with feature ridges shown in Fig. 16(e) gives the impression of being a CADed version of the result shown in Fig. 16(f).

To quantify the results, we measure the level of interpolation achieved by the triangulation in comparison to the input point cloud. Therefore, we calculate the shortest distance between an input point $p \in \mathcal{P}$ and the output mesh $\mathcal{T}$,

$$d(p,\mathcal{T}) = \min_{p' \in \mathcal{T}} \|p - p'\|,$$

where p' denotes a point on the output mesh. We use $d(p,\mathcal{T})$ to determine the one-sided Hausdorff distance between $\mathcal{P}$ and $\mathcal{T}$ as

$$d_{\max} = \max_{p \in \mathcal{P}} d(p,\mathcal{T}).$$

We measure the one-sided Hausdorff distance from P to T in order to find the biggest deviation between output and input. Next, we determine the average distance between $\mathcal{P}$ and $\mathcal{T}$ as

$$d_{\text{avg}} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} d(p,\mathcal{T})$$

and the corresponding root mean square deviation in percent d_{RMS}

$$d_{\text{RMS}} = \frac{100}{d_{\text{avg}}} \sqrt{\frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \left(d(p,\mathcal{T}) - d_{\text{avg}}\right)^2}.$$

We use the implementation of the one-sided Hausdorff distance [24] in MeshLab [15] to compute these values practically. In this sample-based approach, we always sample on vertices, edges, and faces. In this, we utilize ten times the number of samples suggested by MeshLab to obtain an even better approximation of the Hausdorff distance.

As shown in Table 7, applying feature detection does not significantly change the number of vertices and edges in the output geometry. The maximal distance between point cloud and output geometry, the average distance, and the root mean square error are reduced.

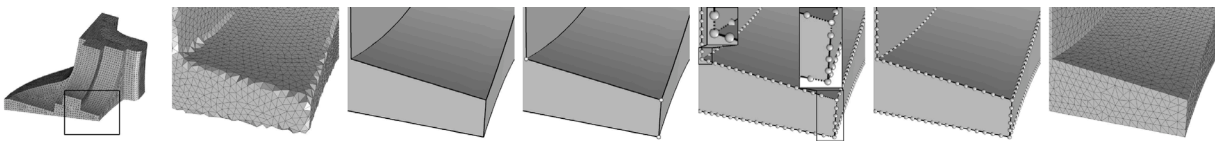


Fig. 18. Feature detection on polyhedral surface (*Fandisk* model). From left to right: Input with highlighted section; algorithm applied without feature detection; detected feature edges; detected feature edges and feature vertices; end of feature lines where no further vertex can be placed; feature vertices regularly distributed along feature lines; remeshed geometry with feature detection.

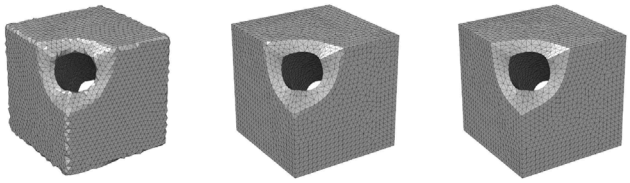


Fig. 19. Result of the algorithm run on *Cube* model. Left: input geometry remeshed without feature detection (see Fig. 15). Middle: input geometry remeshed with $\vartheta = 40°$ and target edge length 0.1. Right: input geometry remeshed with $\vartheta = 24°$ and target edge length 0.1.

In Section 7, we introduced the threshold ϑ for feature detection. Experiments for decreasing values of ϑ showed an increase in the number and density of feature segments as depicted in Fig. 20. A suitable choice for ϑ depends on the input as well as on the remeshing goals of the user. Choosing $\vartheta = 90°$ can be taken as a recommendation to start the search for a reasonable value of ϑ with. That is, since this allows for the detection of geometry parts intersecting under an angle equal to 90° like those found on the *Wrench* model as illustrated in Fig. 16(c). Table 7 lists the angle ϑ used per model, which was chosen to preserve the majority of features present in the model.

Fig. 20 illustrates the increase in the number of feature segments with shrinking values of ϑ on the *Xiao Jie Jie* model. Since for decreasing angle ϑ the number of feature segments increases, regions in the point cloud occur being densely covered with feature segments. In contrast to a single feature segment or a small number of nearly parallel ones, the features are blurred. The quality of the meshes does not differ significantly from the result achieved without feature detection. For target edge length 0.4 and $\vartheta = 40°$, we achieve the worst value for Q_{avg}, which is 0.9420, while without feature detection, the corresponding value of Q_{avg} is equal to 0.9552 determined for the same target edge length. An overview on all data collected for this model is contained in the supplementary material [31], Table 25. Based on the results achieved, the feature angle to choose depends on the geometry.

8.1. Varying sampling density

Depending on the scanning process, the derived sampling points may be distributed in varying density over the geometry. In the upper row of Fig. 21, close-ups of two models taken from [11] are depicted. Both show a higher density of sampling points in areas of high curvature than in low-curvature areas. In the algorithm presented here, the individual splat sizes are chosen based on the distribution of sample points. Hence, areas with low curvature are well represented by a few splats of larger radii while a higher number of splats of smaller radii cover regions of higher curvature. The images in the lower row of Fig. 21 illustrate that the quality of the resulting meshes is not influenced by the variation of the sampling density of the input.

8.2. Handling reach criterion

As mentioned in Section 4.1, the parameter d has to be chosen by the user, depending on the reach ρ of the input. In case d is smaller or equal to the reach, the output consists of an isotropic triangular mesh, as discussed in Section 3. However, choosing d larger than the reach might lead to issues in the growing process. Both choices are illustrated in Fig. 22. For $d > \rho$, the growing process was started on the top side of the *Plate* model (available online [31]) and failed to grow over the rim to the lower side. For any $d < \rho$, the *Plate* model is meshed successfully, as illustrated in the lower image of Fig. 22.

9. Experiments on polyhedral surfaces

To further illustrate the effectiveness of the modifications to the algorithm presented in Sections 6 and 7, we run the algorithm on a broad choice of polyhedral surfaces, with a varying number of sharp features as well as an increasing level of topology. The models we remesh in the following are either taken from an online repository of commonly used meshes [38] or self-made (available online [31]).

We compare the results achieved by our algorithm to the *Isometric Explicit Remeshing* [20] pipeline of MeshLab [15] and to the *Remeshing* [22] module of the PMP library [21]. In all experiments, for both algorithms, we use the standard number ten of iterations. Furthermore, we do not use adaptive remeshing for either implementation as it conflicts the goal of obtaining a uniform edge length. All other parameters are left in their standard configuration of the respective implementation, except for using world-length coordinates in PMP—in the standard configuration, lengths are given relative to the bounding box. In Table 8, a representative choice of CAD models processed with said routines is listed. The derived data show that our single-sweep algorithm produces comparable mesh quality with and without feature detection. A larger variety of models with more detailed data sets can be found in the supplementary material [31].

9.1. Influence of feature detection on Hausdorff distance

In Fig. 19, we observe the striking differences between the results without feature detection and with feature detection. While all features in the result achieved without feature detection are lost, varying the threshold angle ϑ shows an increase of features maintained. In addition to investigating the one-sided Hausdorff distance $d_{\max}$ as introduced in Section 8, we will consider the *relative error* $\frac{d_{\max}}{d}$ as well. Remeshing the *Cube* with $\vartheta = 40°$ does not allow for a complete maintenance of all feature lines and vertices while choosing $\vartheta = 24°$ does as illustrated in Table 9 and depicted in Fig. 19. Further, we can observe the same behavior on the *Fandisk* as well.

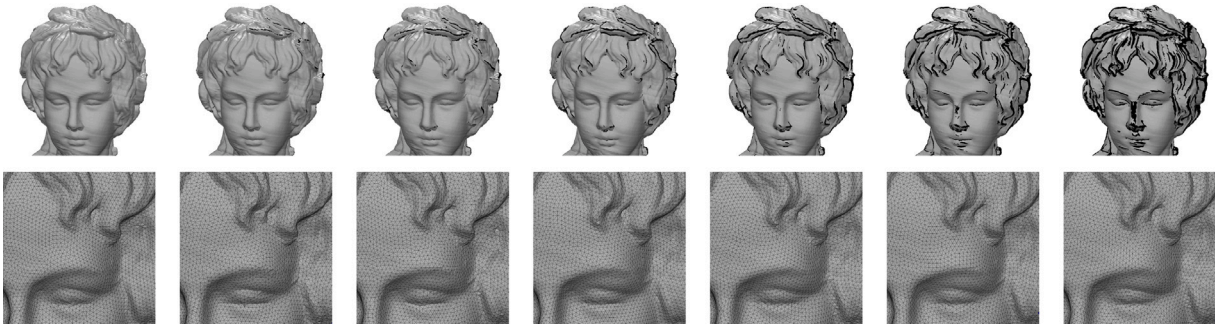


Fig. 20. Feature detection on point clouds, illustrated on the *Xiao Jie Jie* model. Upper row: point cloud and features detected. Lower row: close-up of left eye. Both rows, left to right: without feature detection, detected features for $\vartheta = 90°, 80°, 70°, 60°, 50°, 40°$.

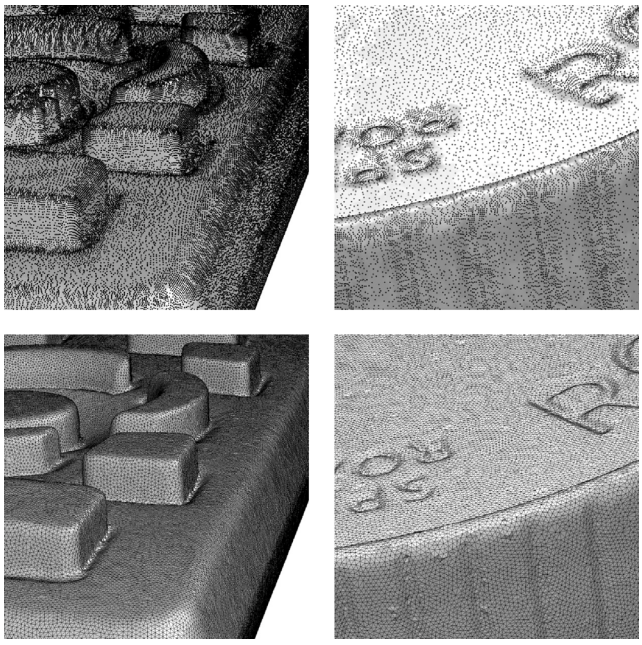


Fig. 21. Upper row: varying sampling density, detected on the *Remote* model and on the *Coffee Bottle Plastic* model. Lower row: resulting isotropic meshes derived from the point samplings shown above.

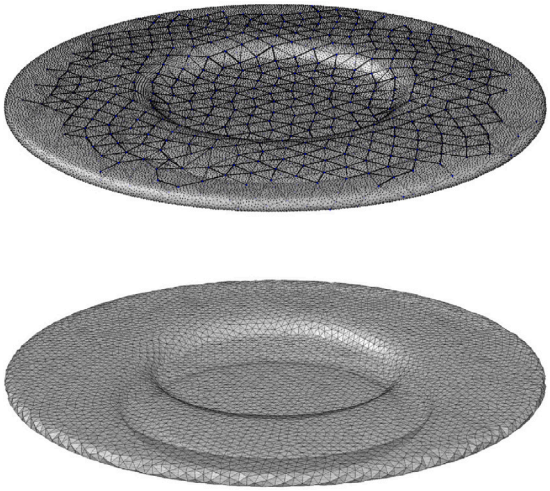


Fig. 22. Results achieved by processing the *Plate* model. Above: failed attempt based on d chosen too big. Below: successful application.

9.2. Influence of feature detection on edge length and angle distribution

In Fig. 23, the angle distribution, the edge length distribution, and the distribution of quality are shown, evaluated on a remeshing of the *Cube* model with and without feature detection. Respecting features of a geometry results in a broader distribution for each of the three quantities evaluated. While the edge length distribution does not change drastically, the angle distribution has a peak at a smaller angle. The quality Q_t is still good, but does not contain as many ideal triangles as the remeshing without feature detection. In contrast to the results achieved by the algorithms mentioned above, we perform better with respect to the minimum angle as well as to the minimum triangle quality. The *Cube* model possesses several feature lines and corners easily to be detected visually. Most angles at points detected as feature vertices of higher valence by our algorithm are close to 90°.

Table 8
Selection of CAD models remeshed with and without feature detection using MeshLab (ML), Polygon Mesh Processing (PMP), and our algorithm. A more detailed evaluation is presented in the supplementary material [31].

	Algo.	d	ϑ	E_{avg}	E_{RMS}	Q_{avg}	Q_{RMS}
Cube	ML	0.04	24°	0.0408	12.8	0.9600	4.5
	PMP	0.04	24°	0.0364	12.2	0.9636	3.5
	ours	0.04	24°	0.0436	14.1	0.9295	5.8
	ML	0.04	–	0.0404	14.5	0.9525	7.3
	PMP	0.04	–	0.0364	11.8	0.9663	3.1
	ours	0.04	–	0.0426	11.3	0.9584	4.5
Fandisk	ML	0.012	60°	0.0124	18.9	0.9526	5.0
	PMP	0.012	60°	0.0112	12.3	0.9713	2.9
	ours	0.012	60°	0.0131	14.2	0.9221	5.3
	ML	0.012	–	0.0124	12.9	0.9531	4.9
	PMP	0.012	–	0.0125	12.2	0.9720	2.8
	ours	0.012	–	0.0127	11.1	0.9592	4.4
Flange	ML	1.0	80°	1.0241	13.2	0.9562	5.1
	PMP	1.0	80°	0.9240	12.6	0.9601	3.8
	ours	1.0	80°	1.0645	11.8	0.9585	4.9
	ML	1.0	–	0.9361	27.7	0.8823	21.4
	PMP	1.0	–	0.9269	12.2	0.9626	3.5
	ours	1.0	–	1.0666	11.4	0.9572	4.5
Oloid	ML	0.02	80°	0.202	15.8	0.9455	9.4
	PMP	0.02	80°	0.0183	12.3	0.9624	3.4
	ours	0.02	80°	0.0214	11.8	0.9508	4.5
	ML	0.02	–	0.0201	17.3	0.9373	11.2
	PMP	0.02	–	0.0183	12.3	0.9625	3.4
	ours	0.02	–	0.0212	10.9	0.9606	4.4

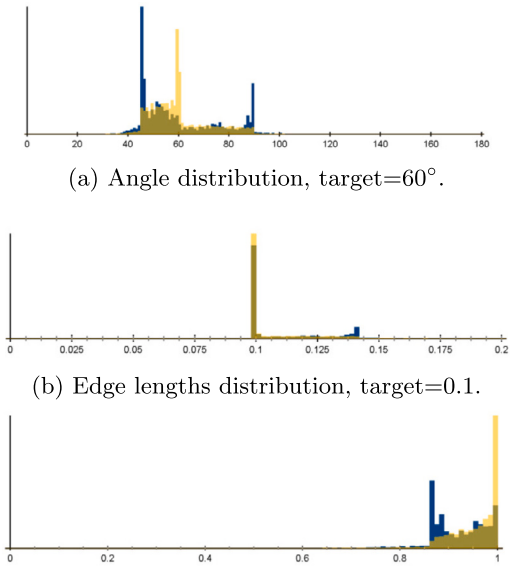
Table 9
Evaluation of one-sided Hausdorff distance of CAD models (*Cube* and *Fandisk*), comparing to the output of MeshLab (ML) and Polygon Mesh Processing (PMP).

	Algo.	d	ϑ	$d_{\max}$	$\frac{d_{\max}}{d}$	d	ϑ	$d_{\max}$	$\frac{d_{\max}}{d}$
Cube	ML	0.04	—	0.0347	0.8687	0.1	—	0.0808	0.8082
	PMP	0.04	—	0.0334	0.8353	0.1	—	0.2781	0.6623
	ours	0.04	—	0.0324	0.8122	0.1	—	0.0776	0.7765
	ML	0.04	24°	0.0043	0,1078	0.1	24°	0.0062	0.0620
	PMP	0.04	24°	0.0038	0.0951	0.1	24°	0.0061	0.0614
	ours	0.04	24°	0.0042	0.1059	0.1	24°	0.0084	0.0843
	ML	0.04	40°	0.0043	0.1033	0.1	40°	0.0190	0.1908
	PMP	0.04	40°	0.0065	0.1637	0.1	40°	0.0196	0.1961
	ours	0.04	40°	0.0097	0.2427	0.1	40°	0.0213	0.2134
Fandisk	ML	0.012	—	0.0111	0.9302	0.024	—	0.0216	0.9039
	PMP	0.012	—	0.0080	0.6721	0.024	—	0.0180	0.7524
	ours	0.012	—	0.0097	0.8147	0.024	—	0.0184	0.7679
	ML	0.012	60°	0.0035	0.2929	0.024	60°	0.0061	0.2561
	PMP	0.012	60°	0.0027	0.2331	0.024	60°	0.0055	0.2302
	ours	0.012	60°	0.0034	0.2835	0.024	60°	0.0072	0.2983
	ML	0.036	—	0.0318	0.8840	0.048	—	0.0461	0.9618
	PMP	0.036	—	0.0261	0.7252	0.048	—	0.0334	0.6976
	ours	0.036	—	0.0302	0.8376	0.048	—	0.0337	0.7025
	ML	0.036	60°	0.0060	0.1690	0.048	60°	0.0115	0.2405
	PMP	0.036	60°	0.0088	0.2454	0.048	60°	0.0125	0.2612
	ours	0.036	60°	0.0085	0.2367	0.048	60°	0.0134	0.2806

This leads to a larger number of nearly isosceles triangles with angles close to 45° and 90°, which can be seen in the histograms in Figs. 23. In case the sharp features of the input geometry do not meet at an angle close to 90°, the angle distribution of the remeshed geometry varies less, comparing the results achieved with and without feature detection as depicted in Fig. 24 on the *Flange*.

9.3. Sharp angles

In Fig. 25, two feature lines are shown, meeting under an angle γ less than 60°. Triangulating the area between the feature lines may introduce edges within the triangulation step shorter than the target edge length. Since the length of the third side of the triangle filling the space at the meeting point of the feature lines depends on γ, there are two different ways to go. The first one consists of maintaining the features detected and accept the occurrence of (a comparably small number of) edges shorter than the target edge length, while the second one maintains the target edge length, but loses one of the features. Here, we decided to follow the first possibility, which is illustrated in Fig. 25.

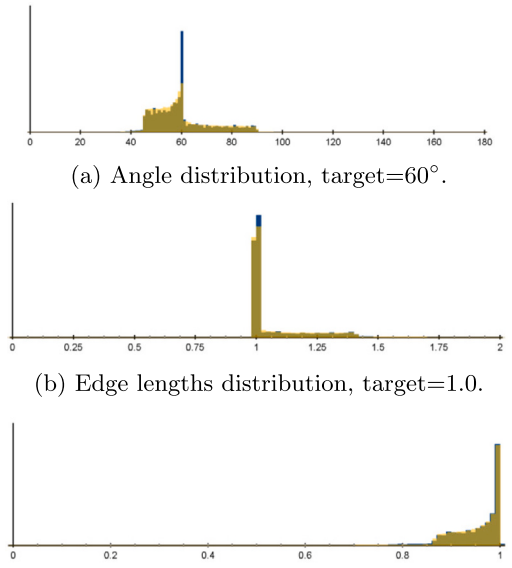


(a) Angle distribution, target=60°.

(b) Edge lengths distribution, target=0.1.

(c) Distribution of quality Q_t, target=1.0.

Fig. 23. Distributions obtained by our algorithm applied to the *Cube* model with feature detection (shown in blue) for $\vartheta = 24°$ and without feature detection (shown in yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Angle distribution, target=60°.

(b) Edge lengths distribution, target=1.0.

(c) Distribution of quality Q_t, target=1.0.

Fig. 24. Distributions obtained by our algorithm applied to the *Flange* model with feature detection (shown in blue) for $\vartheta = 80°$ and without feature detection (shown in yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A similar obstacle occurs in case the angle between the two parts of the surfaces joint along a single feature line is less or equal to 60°. Here, the disk growing step is restricted to one side solely, since no sphere can be placed on the other side of the ridge because it is too close to spheres already placed. Hence, internal connections between the two sides may occur as well as a non-manifold result since the lack of possibilities to place spheres cuts a hole into the surface as illustrated in Fig. 26. In case such a feature line forms a closed curve on the geometry, as it does on the *Boat* model, the geometry can be cut into

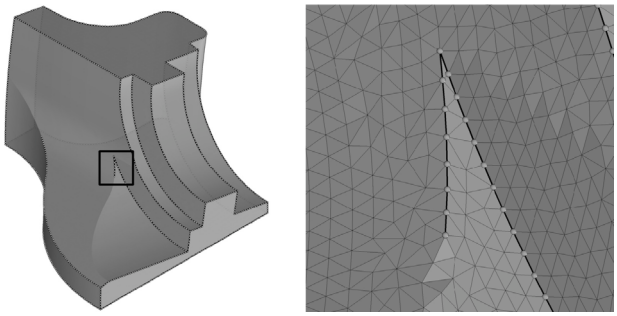


Fig. 25. Feature lines meeting at angle < 60°. Left: *Fandisk* model with emphasized feature lines meeting. Right: triangulation of surface between meeting features, containing edges shorter in length than the preset target edge length.

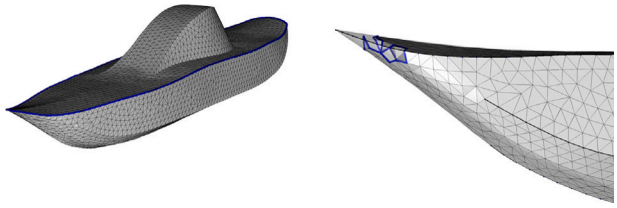


Fig. 26. Sharp feature ridges on *Boat* model. Left: *Boat* model remeshed with features. Right: close up of *Boat* model showing faulty edges.

two parts along the feature curve. Since the boundaries of both parts are already covered with spheres placed in the feature detection step, we use them for initialization. Subsequently, we place further spheres in a disk growing process applied to each part of the geometry individually. Hence, we prevent the spheres placed to intersect the other part of the geometry.

The ansatz we described above to mesh feature lines, is applicable to geometries with boundaries as well. Here, the spheres placed on the boundaries directly are used for initialization. In case, a feature line does not close, but makes an angle less than 60°, the geometry has to be segmented in order to prevent the spheres to intersect the geometry several times. To find such a segmentation is left as future work.

10. Conclusion and future work

In this paper, we built on a surface representation by equally sized spheres [5] to provide a feature-aware algorithm for meshing of point clouds and remeshing of polyhedral surfaces. A prior publication of the algorithm introduced the meshing of point clouds with a guaranteed smallest edge length [6], see Sections 3 to 5. This extension enables the algorithm to remesh polyhedral surfaces with and without sharp features, see Sections 6 to 9. The algorithm still only needs a single, greedy sweep across the input geometry to obtain the resulting mesh. Respecting features of the input point cloud or surface mesh makes it possible to process a broader variety of input geometries in a way that is suitable for a variety of follow-up applications. The results achieved remain guaranteed to be manifold, based on the theory discussed in Section 3.

The *Fandisk* model and the *Boat* model discussed in Section 9.3 already hinted at some shortcomings of our algorithm. These arise when the model demands for very sharp angles in the geometry that cannot be modeled while maintaining the desired minimum edge length. While this can be interpreted as a feature of the algorithm, in these cases — especially with the tip of the *Boat* model — it would be preferential to let the user decide to violate the distance requirement between non-connected vertices to obtain a faithful reconstruction of the input

geometry. A similar case could be made for the *Fandisk* model, where the angle made between features might be worthwhile preserving over the cost of having few badly shaped triangles. Highlighting these cases and letting the user decide how to handle them is left as future work.

On a different note, none of the surfaces handled here are equipped with boundary. Technically, handling boundary is not very different from handling features: If the boundary is given as a polygon, it can be used for initialization. Then, when triangulating the regions following Section 3.7, regions solely bounded by boundary edges are omitted. The tricky element here is to identify a boundary polygon for point cloud input. While corresponding approaches exist in the literature [39], implementing such in our context is left as future work.

Finally, this paper used the uniform sphere representation of surfaces [5] in the context of remeshing. Fundamentally, the representation is not limited to surfaces and could also be applied to represent volumes by a collection of single-sized spheres. Similar to the remeshing of surfaces, a volume representation could thus be used for tetrahedral meshing and remeshing of volumetric objects. Like the previous considerations, this is left as future work.

CRediT authorship contribution statement

Henriette Lipschütz: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Ulrich Reitebuch:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation. **Konrad Polthier:** Supervision, Resources, Project administration, Funding acquisition. **Martin Skrodzki:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Martin Skrodzki reports financial support was provided by German Research Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is shared on a dedicated website together with supplementary material [31].

References

[1] Helle RH, Lemu HG. A case study on use of 3D scanning for reverse engineering and quality control. Mater Today: Proc 2021;45:5255–62.

[2] Chaudhari R, Loharkar PK, Ingle A. Medical applications of rapid prototyping technology. In: Recent advances in industrial production. Springer; 2022, p. 241–50.

[3] Mellado N, Marcadet Q, Espinasse L, Mora P, Dutailly B, Tournon-Valiente S, Granier X. 3D-ARD: A 3D-Acquired research dataset. 2020.

[4] Wiersma R, Nasikun A, Eisemann E, Hildebrandt K. A Fast Geometric Multigrid Method for Curved Surfaces. SIGGRAPH 2023 2023;41(4):1–11.

[5] Lipschütz H, Skrodzki M, Reitebuch U, Polthier K. Single-sized spheres on surfaces (S4). Comput Aided Geom Design 2021;85:101971.

[6] Lipschütz H, Reitebuch U, Polthier K, Skrodzki M. Manifold meshing of point clouds with guaranteed smallest edge length. In: Proceedings of the 2024 international meshing roundtable. IMR, SIAM; 2024, p. 1–13.

[7] Amenta N, Choi S, Kolluri RK. The power crust. In: Proceedings of the sixth ACM symposium on solid modeling and applications. SMA '01, New York, NY, USA: Association for Computing Machinery; 2001, p. 249–66.

[8] Levin D. The approximation power of moving least-squares. Math Comp 1998;67(224):1517–31.

[9] Bernardini F, Mittleman J, Rushmeier H, Silva C, Taubin G. The ball-pivoting algorithm for surface reconstruction. IEEE Trans Vis Comput Graphics 1999;5(4):349–59.

[10] Edelsbrunner H, Kirkpatrick D, Seidel R. On the shape of a set of points in the plane. IEEE Trans Inform Theory 1983;29(4):551–9.

[11] Huang Z, Wen Y, Wang Z, Ren J, Jia K. Surface Reconstruction from Point Clouds: A Survey and a Benchmark. 2022, arXiv preprint arXiv:2205.02413.

[12] Kazhdan M, Bolitho M, Hoppe H. Poisson surface reconstruction. In: Sheffer A, Polthier K, editors. Symposium on geometry processing. The Eurographics Association; 2006, p. 61–70.

[13] The CGAL Project. CGAL User and Reference Manual. 5.6. CGAL Editorial Board; 2023.

[14] Kazhdan M, Hoppe H. An adaptive multi-grid solver for applications in computer graphics. Comput Graph Forum 2019;38(1):138–50.

[15] Cignoni P, Callieri M, Corsini M, Dellepiane M, Ganovelli F, Ranzuglia G. MeshLab: an Open-Source Mesh Processing Tool. In: Scarano V, Chiara RD, Erra U, editors. Eurographics Italian chapter conference. The Eurographics Association; 2008, p. 129–36.

[16] Cohen-Steiner D, Da F. A greedy delaunay-based surface reconstruction algorithm. Vis Comput 2004;20:4–16.

[17] Digne J, Morel J-M, Souzani C-M, Lartigue C. Scale space meshing of raw data point sets. Comput Graph Forum 2011;30(6):1630–42.

[18] Öztireli C, Guennebaud G, Gross M. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. In: Proceedings of eurographics 2009, Comput Graph Forum In: Proceedings of eurographics 2009, 2009;28(2).493–501,

[19] Boltcheva D, Lévy B. Surface reconstruction by computing restricted voronoi cells in parallel. Computer-Aided Des 2017;90:123–34.

[20] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Mesh optimization. In: Proceedings of the 20th annual conference on computer graphics and interactive techniques. SIGGRAPH '93, New York, NY, USA: Association for Computing Machinery; 1993, p. 19–26.

[21] Sieger D, Botsch M. The Polygon Mesh Processing Library. 2023, http://dx.doi.org/10.5281/zenodo.10866532, Version: 3.0.0, https://github.com/pmp-library/pmp-library.

[22] Botsch M, Kobbelt L. A remeshing approach to multiresolution modeling. In: Proceedings of the 2004 eurographics/ACM SIGGRAPH symposium on geometry processing, vol. 71, 2004, p. 189–96.

[23] Surazhsky V, Gotsman C. Explicit Surface Remeshing. In: Kobbelt L, Schroeder P, Hoppe H, editors. Eurographics symposium on geometry processing. The Eurographics Association; 2003.

[24] Cignoni P, Rocchini C, Scopigno R. Metro: measuring error on simplified surfaces. In: Computer graphics forum, vol. 17, (2):Blackwell Publishers; 1998, p. 167–74.

[25] Skrodzki M, Zimmermann E. A large-scale evaluation of shape-aware neighborhood weights and neighborhood sizes. Computer-Aided Des 2021;141:103107.

[26] Yadav SK, Reitebuch U, Skrodzki M, Zimmermann E, Polthier K. Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics. Comput Graph 2018.

[27] Yadav SK, Skrodzki M, Zimmermann E, Polthier K. Surface denoising based on normal filtering in a robust statistics framework. In: Proceedings of the forum "math-for-industry" 2018: big data analysis, AI, fintech, math in finances and economics, vol. 35, Springer Nature; 2022, p. 103–32.

[28] Boissonnat J-D, Lieutier A, Wintraecken M. The reach, metric distortion, geodesic convexity and the variation of tangent spaces. J Appl Comput Topol 2019;3:29–58.

[29] Huang H, Wu S, Cohen-Or D, Gong M, Zhang H, Li G, Chen B. L1-medial skeleton of point cloud. ACM Trans Graph 2013;32(4):1–8.

[30] Mitra NJ, Nguyen AT, Guibas LJ. Estimating surface normals in noisy point cloud data. In: SCG '03. 2003, p. 322–8.

[31] Lipschütz H, Reitebuch U, Skrodzki M. Supplementary material and CAD models. 2025, https://ms-math-computer.science/projects/guaranteed-smallest-edge-length-manifold-meshing.html.

[32] Sedgewick R, Wayne K. Algorithms. Addison-Wesley Professional; 2011.

[33] Ma M, Yu X, Lei N, Si H, Gu X. Guaranteed quality isotropic surface remeshing based on uniformization. Procedia Eng 2017;203:297–309.

[34] Shewchuk JR. What is a good linear element? Interpolation, conditioning, and quality measures. In: Chrisochoides N, editor. Proceedings of the 11th international meshing roundtable, IMR 2002, ithaca, new york, USA, September 15-18, 2002. 2002, p. 115–26.

[35] Giraudot S. Surface reconstruction from point clouds. In: CGAL user and reference manual. CGAL Editorial Board; 2022, p. 5.5.1.

[36] Levy B. Geogram. 2023, https://github.com/BrunoLevy/geogram.

[37] Foskey M, Lin MC, Manocha D. Efficient computation of a simplified medial axis. In: Proceedings of the eighth ACM symposium on solid modeling and applications. 2003, p. 96–107.

[38] Jacobson A. Common 3D Test Models. 2025, https://github.com/alecjacobson/common-3d-test-models.

[39] Mineo C, Pierce SG, Summan R. Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction. J Comput Des Eng 2019;6(1):81–91.