# Manifold Meshing of Point Clouds with Guaranteed Smallest Edge Length

Henriette Lipschütz* Ulrich Reitebuch† Konrad Polthier‡ Martin Skrodzki§

**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. Furthermore, the output is stable under moderate levels of noise.

Supplementary material, an extended preprint, 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 [11], rapid prototyping [5], or architecture [20]. 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 [26, 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 [18]. 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. In summary, the contributions of this paper are:

- 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.

## 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 [1], or convergence to the ground truth with increasing sampling density [16]. Some algorithms guarantee local connectedness of their output [2], while others guarantee their output to stay within the convex hull of the given input [9]. 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 [13]. 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 [14], implemented in CGAL [25]. 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 re-

*Freie Universität Berlin, Germany
†Freie Universität Berlin, Germany
‡Freie Universität Berlin, Germany
§TU Delft, The Netherlands; corresponding author: mail@ms-math-computer.science; supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 455095046.

gions 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 solver capable of solving a linear system discretized over a spatial domain [15], implemented in MeshLab [6].

Second, the scale-space approach [7], implemented in CGAL [25], 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 [8], implemented in CGAL [25], 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 [22], implemented in MeshLab [6], combines implicit MLS with robust statistics. The MLS approach [16] 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 [4], available via MeshLab [6]. 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.

Note that 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 [18], 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. In Section 5.1, we will provide a detailed comparison of our algorithm with the works listed here.

## 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 $\mathcal{C}^2$-manifold embedded into $\mathbb{R}^3$, which is assumed to be closed and of finite reach

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

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:

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

Now, for any $p, q \in \mathcal{M}$ such that $\|p - q\| < 2\rho$, the following estimation holds [3, 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 [3] gives an upper bound for the angle $\sphericalangle(T_p\mathcal{M}, T_q\mathcal{M})$ between them,

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

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

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

For a given angle $\varphi_{\max} \in \left[0, \frac{\pi}{2}\right[$, the second part of Equation (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 $\mathcal{M}' \coloneqq B_{r_{\max}}(p) \cap \mathcal{M}$ 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 (Figure 1). Furthermore, we have:

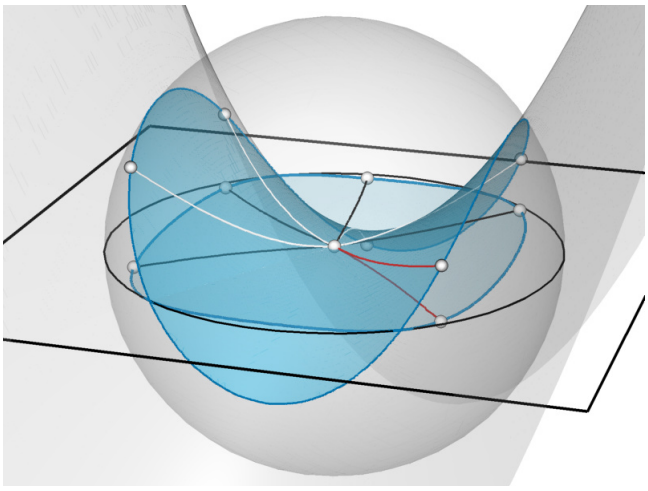


Figure 1: Illustration of Lemma 3.1. Intersection of a saddle-shaped surface with a sphere (blue). Six points on the surface are marked as well as their projections to the tangent plane belonging to the center of the sphere.

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 $\pi$ 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 $\mathcal{S}$ [3, 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 [3, 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 $\gamma$. As $\mathcal{I}$ is geodesically convex, $\gamma$ 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 $\partial R$. 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 [18]. 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 the corresponding normal $n_p$ is orthogonal to $S_p$. 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 Figure 2a. Note that following Lemma 3.1, the user has to choose the parameter $d$ with respect to the reach $\rho$ of the input, which can be estimated for point clouds [12], to ensure a manifold output. 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*.

**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 of $\mathcal{G}$** After initialization, disk growing is performed to create further vertices and edges of $\mathcal{G}$. A vertex is added from the list of vertex candidates, connected by edges to the two vertices distance $d$ away (Figure 5a), and new vertex candidates are added based on the newly placed vertex (Figure 5d). 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 (Figures 3a and 3b).

As shown in Figure 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 bigger 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 overfolds. Given the prioritization of vertices, these can now be added to the graph $\mathcal{G}$.

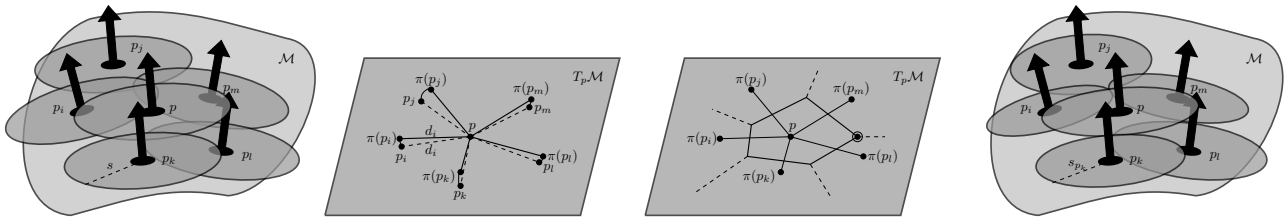
**3.6 Creating a New Vertex** Once a vertex candidate $v_c$ has been chosen, it is first determined whether there is a vertex $v$ in $\mathcal{G}$ such that $\|v_c - v\|_2 < d$. If so, the vertex candidate is discarded.

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 Figures 4a and 4b for an example of a surface with reach $\rho = 0$.

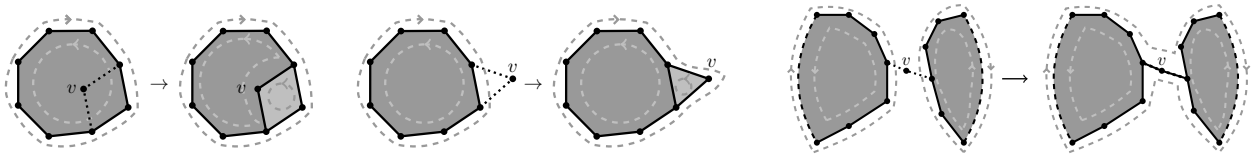
In these cases, we still want to prevent such faulty connections. Therefore, we find an approximated surface normal, which will be discussed in Section 4.1. We project $v_c$ and its prospective edges as well as all edges already existing in the vicinity of $v_c$ along this normal. For this projection, 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 (Figure 4b). 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 [18, 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




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

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




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

Figure 3: Possibilities when creating new vertices and edge connections in the graph $\mathcal{G}$: In 3a, 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 3b, 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.

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 bigger 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.

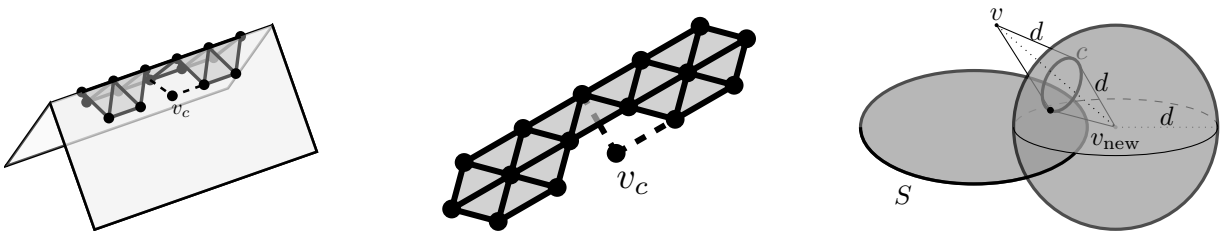
## 4 Implementation

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 [21]. Furthermore, the user has to choose the implementation-related parameter $w$ (Section 4.2).

**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 (Figure 5c). 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 $\left\|\overline{n}_{b_j}\right\|_2$ 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

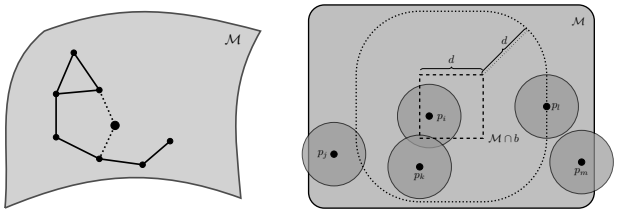



(a) 3D view of a tent-like feature, a geometry with reach $\rho = 0$.

(b) Projection of the graph along an approximated surface normal.

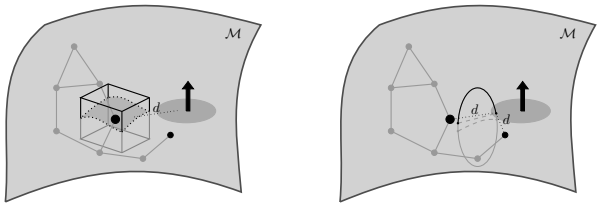
(c) Intersection of circle $c$ with splat $S$ gives next vertex candidate.

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




(a) Adding a new vertex.

(b) Collecting splats, top view.




(c) Collecting normals.

(d) Finding candidates.

Figure 5: Update steps of the algorithm.

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 Figure 4a. To compute an approximation efficiently, we take a finite sampling $\mathcal{S}_F \subset \mathbb{S}^2$ and derive the box normal $n_{b_j}$ as

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

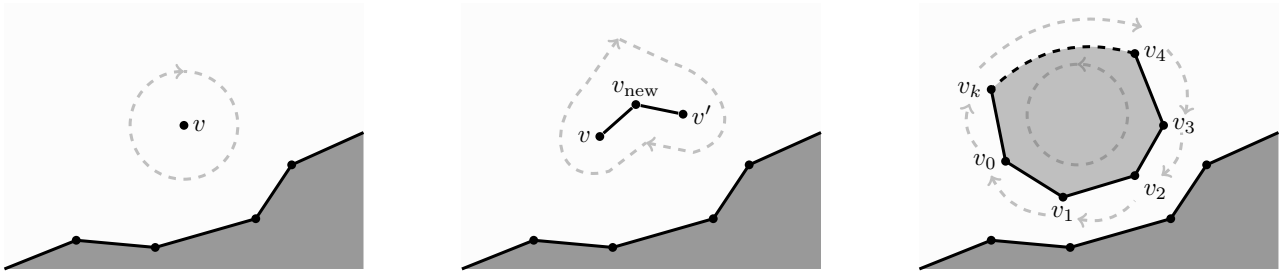
That is, we search for the unit normal that maximizes the smallest scalar product with all point normals associated to the box $b_j$. For each box $b_j$, the scalar product $\langle n_{p_i}, n_{b_j} \rangle$ is ideally strictly positive for all points $p_i \in \mathcal{B}_j$, even in those cases where 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 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 (Figure 5b).

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




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

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

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.

**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 (Figure 6a). 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 [18].

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.

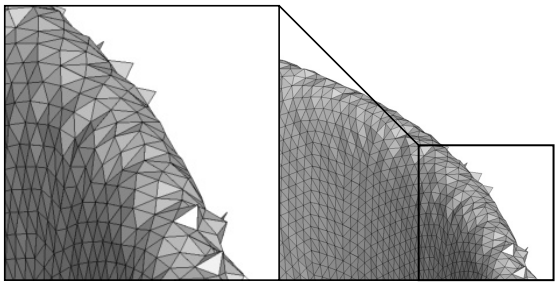
Furthermore, we experienced that setting the window size to $w = 0$ immediately creates noticeable negative effects on the result of the algorithm. In this case, our algorithm defaults to the *breadth-first* strategy of [18] and thus creates visible seams on the geometry (Figures 7a to 7d). Starting 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, 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 (Figure 8a). 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.

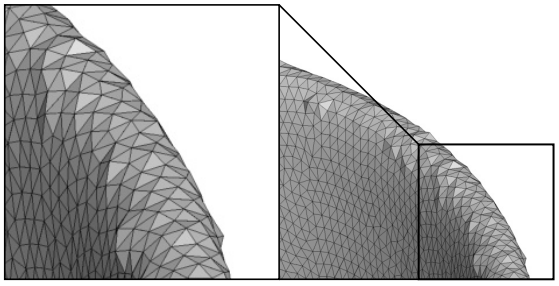
An additional benefit is that for smaller splat sizes, there are less splats registered per box, which speeds up the algorithm. However, the individual splats have to have sizes sufficient to cover the underlying geometry. To find the specific splat size $s_p$ for each point $p \in \mathcal{P}$, we use the box data structure (Figure 2d). 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 (Figure 2b). 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

(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).

Figure 7: Visible seams on the *Bottle Shampoo*. Figures 7a to 7d show experimental results obtained by inserting new vertices via pure breadth-first-growing. The seams appear as regions having a high number of border edges compared to all other regions on the surface. Figures 7e and 7f show results obtained by prioritizing new vertex candidates. For better visibility, the target edge length $d$ was chosen as 1.



(a) Result with uniform splat sizes.



(b) Result with individual splat sizes.

Figure 8: Running our algorithm on the *Bowl Chinese* from [13] with 8a using a global splat size and 8b using a local spat size. The letter reduces artifacts in high-curvature regions such as the rim of the bowl.

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 (Figure 2c). 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 (Figure 8b).

**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 [23, Chapter 2.4].

If a vertex still has correct priority, checking for conflict with existing vertices and performing the projection check from Figure 4b 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 $v_{\text{new}}$ is created, we compute new vertex candidates having $v_{\text{new}}$ as parent vertex. Therefore, we need the set of all splats intersecting the ball of radius $d$ centered at $v_{\text{new}}$. This is a subset of the set of those splats associated to the box containing $v_{\text{new}}$. To efficiently access potential second parent vertices, we maintain for each splat $S$ a list of all vertices within distance $d$ to $S$ (Figures 5b and 5c).

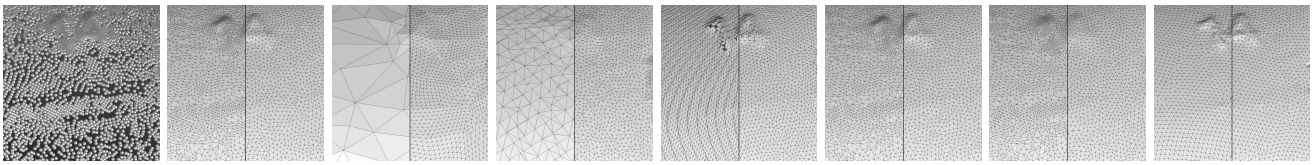
## 5 Experiments

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 [19, 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



Input. Adv. Front. Poisson. Poisson MG. RIMLS. Scale Space. Voronoi. Ours.

Figure 9: Qualitative comparison of named algorithms without remeshing (left) and with meshing (right).

*scale-invariant (smooth)* conditioning quality measure discussed by Shewchuk [24, Table 3]. Based on the local, triangle-based measure $Q_t$, further following [19], 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 $Q_{\text{RMS}}$ as

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

See Section 3 of [24] 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 $E_{\text{avg}}$ as well as the corresponding root mean square deviation $E_{\text{RMS}}$, 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 [13]. Here, we concentrate on high-resolution scans obtained by an *OKIO 5M* scanning device, resulting in 330k to 2,000k 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.

**5.1 Experimental Comparison for Point Cloud Meshing** From the algorithms listed in Section 2, Poisson [14], advancing front [7], and scale space [8] are run with the standard parameters as implemented in [10] except for the cleaning steps, which were unnecessary because of the high-quality input. Multigrid Poisson [15] and Voronoi reconstruction [4] are run with the standard parameters as implemented in [17]. RIMLS [22] is run with the standard parameters from [6], 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 [6]. This filter repeatedly applies edge flip, collapse, relax, and refine operations. We run three iterations with a target edge length of 0.2 in absolute world units for the input [13].

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. We chose the *Bottle Shampoo* and the *Bowl Chinese* because of their features, as explored in Figures 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 $E_{\text{avg}}$ (see supplementary material 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 $Q_{\text{avg}}$ 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

| Algorithm | $\lvert\mathcal{T}\rvert$ | $E_{\text{avg}}$ | $E_{\text{RMS}}$ | $Q_{\text{avg}}$ | $Q_{\text{RMS}}$ |
|---|---|---|---|---|---|
| Adv. Front | 1.209,546 | 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 1: *Bottle Shampoo* (604,903 input points).

| Algorithm | $\lvert\mathcal{T}\rvert$ | $E_{\text{avg}}$ | $E_{\text{RMS}}$ | $Q_{\text{avg}}$ | $Q_{\text{RMS}}$ |
|---|---|---|---|---|---|
| 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 2: *Bowl Chinese* (606,320 input points).

| Algorithm | $\lvert\mathcal{T}\rvert$ | $E_{\text{avg}}$ | $E_{\text{RMS}}$ | $Q_{\text{avg}}$ | $Q_{\text{RMS}}$ |
|---|---|---|---|---|---|
| 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 3: *Cloth Duck* (1,018,891 input points).

| Algorithm | $\lvert\mathcal{T}\rvert$ | $E_{\text{avg}}$ | $E_{\text{RMS}}$ | $Q_{\text{avg}}$ | $Q_{\text{RMS}}$ |
|---|---|---|---|---|---|
| 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** |

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

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 $Q_{\text{RMS}}$ 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 models, the lowest deviation $Q_{\text{RMS}}$ is at most 0.6% better than ours, cf. supplementary material.

Regarding the second metric, note that by construction, all edges produced by our algorithm are of length $\geq 0.2$. 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 $E_{\text{avg}}$. 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 $E_{\text{RMS}}$, 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 (Figure 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 Figure 12 for a corresponding set of plots for the *Bottle Shampoo* and find histograms for the other models in the supplementary material. 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 Figure 11, where the competitors are reported including the remeshing time. All experiments were run on a machine with an Intel® Core™ i7-5600U CPU 2.60GHz with four cores and 16GB of RAM. Five of the models did not fit the RAM of this comparison

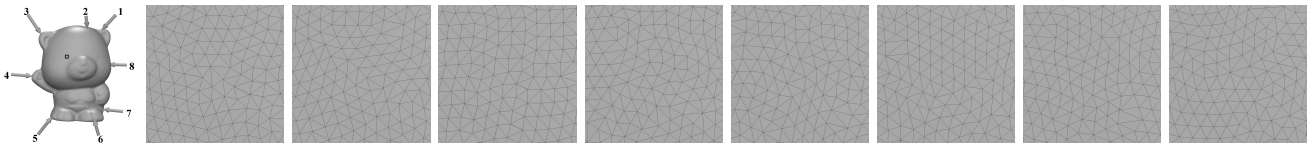


Figure 10: Results with various starting vertices. From left to right: The *Toy Bear* with placements positions for starting vertex pairs, close up for pairs 1 to 8, showing the area at one eye emphasized in the first image.

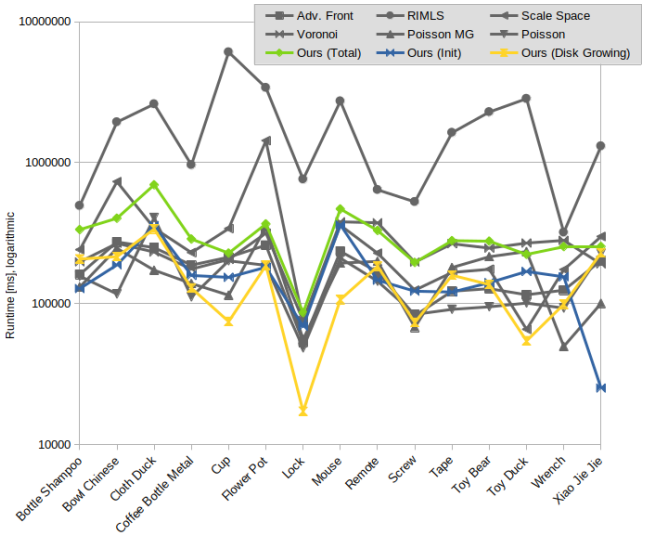
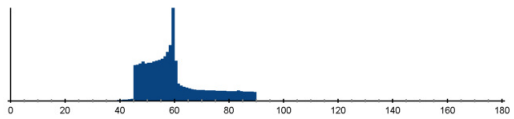
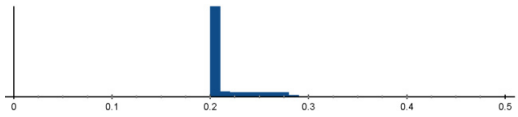


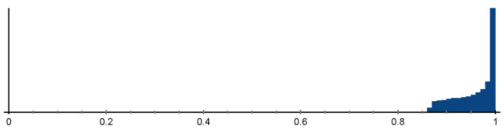

Figure 11: Log of the run time of the algorithms on several models. Ours is additionally split into initialization and disk growing.




(a) Angle distribution, target=60°.




(b) Edge lengths distribution, target=0.2.




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

Figure 12: Distributions of the *Bottle Shampoo* as obtained by our algorithm (without remeshing).

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.

**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. As illustrated in Figure 10, 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 edge length varies from 0.2158 to 0.2159, as does the average quality: from 0.9499 to 0.9504. In all cases, $E_{\text{RMS}}$ is equal to 13.5 while $Q_{\text{RMS}}$ 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

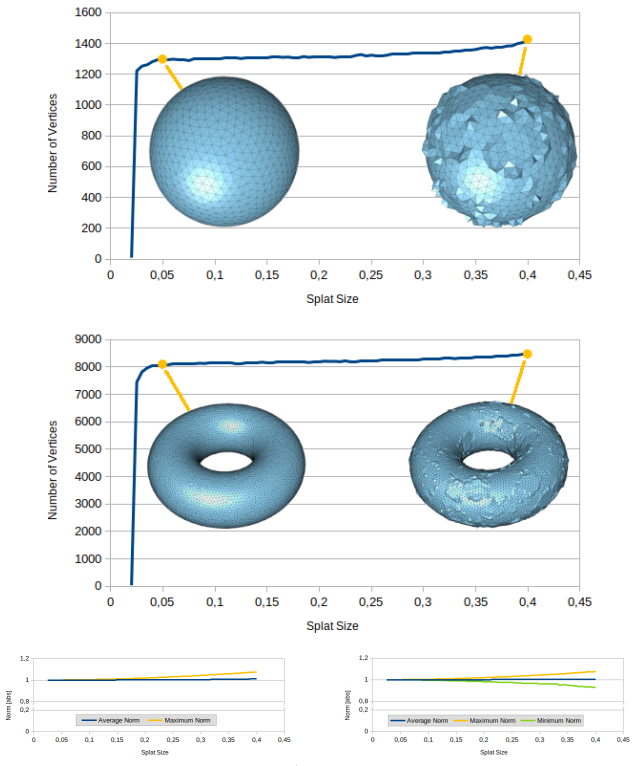



Figure 13: Measuring the reconstruction quality.

are stable until larger splat sizes are reached, which causes visible distortion in the reconstructed models (Figure 13). 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 [13] 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 $\nu$. This moves the noisy points within a bound of $\pm\nu$ around the ground truth. To

| Name \ $\nu$ | 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 5: Noise levels $\nu$ for which the reconstruction is (✓) or is not (✗) manifold.

measure the quality of the output, for each level $\nu$ 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 $2\nu$ between 60% to 100% of $d$. For the user, this experiment suggests that for a geometry with known or estimated noise level $\nu$, choosing $d \geq 2\nu$ yields the best results.

## 6 Conclusion and Future Work

We have presented a surface reconstruction algorithm that produces triangulations with edge lengths close to uniformity, oriented at a user-chosen target edge length. In experiments with real-world models, the algorithm can compete with several established state-of-the-art methods in both quantitative and qualitative aspects. Furthermore, our implementation proved to be competitive in a timing comparison. Additionally, different experiments run on a variety of scan data show the robustness of the algorithm to both variations in input parameters and noise in the input geometries.

The algorithm has the potential to be extended in several directions. First, introducing more than two starting vertices can be used for feature preservation: Placing starting vertices in high-curvature regions ensures that sampled features persist under the reconstruction. Second, the algorithm can be run on a mesh as input and be used for remeshing rather than surface reconstruction. In this application, the input mesh will serve as the surface where to place new vertices, replacing the splat approximation. Finally, starting with a closed surface, the next question is whether the sphere packing approach can be extended into the interior of the geometry. Thereby, a sphere packing could not only provide a surface mesh but also a tetrahedral volume mesh. These extensions are left as future work.

## References


[1] N. Amenta, S. Choi, and R. K. Kolluri. The Power Crust. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, SMA '01, pages 249—-266, New York, NY, USA, 2001. Association for Computing Machinery.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[3] J.-D. Boissonnat, A. Lieutier, and M. Wintraecken. The reach, metric distortion, geodesic convexity and the variation of tangent spaces. *Journal of Applied and Computational Topology*, 3:29–58, 2019.

[4] D. Boltcheva and B. Lévy. Surface reconstruction by computing restricted voronoi cells in parallel. *Computer-Aided Design*, 90:123–134, 2017.

[5] R. Chaudhari, P. K. Loharkar, and A. Ingle. Medical Applications of Rapid Prototyping Technology. In *Recent Advances in Industrial Production*, pages 241–250. Springer, 2022.

[6] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In V. Scarano, R. D. Chiara, and U. Erra, editors, *Eurographics Italian Chapter Conference*, pages 129–136. The Eurographics Association, 2008.

[7] D. Cohen-Steiner and F. Da. A greedy Delaunay-based surface reconstruction algorithm. *The Visual Computer*, 20:4–16, 2004.

[8] J. Digne, J.-M. Morel, C.-M. Souzani, and C. Lartigue. Scale Space Meshing of Raw Data Point Sets. *Computer Graphics Forum*, 30(6):1630–1642, 2011.

[9] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[10] S. Giraudot. Surface Reconstruction from Point Clouds. In *CGAL User and Reference Manual*, page 5.5.1. CGAL Editorial Board, 2022.

[11] R. H. Helle and H. G. Lemu. A case study on use of 3d scanning for reverse engineering and quality control. *Materials Today: Proceedings*, 45:5255–5262, 2021.

[12] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1, 2013.

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

[14] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson Surface Reconstruction. In A. Sheffer and K. Polthier, editors, *Symposium on Geometry Processing*, pages 61–70. The Eurographics Association, 2006.

[15] M. Kazhdan and H. Hoppe. An Adaptive Multi-Grid Solver for Applications in Computer Graphics. *Computer Graphics Forum*, 38(1):138–150, 2019.

[16] D. Levin. The approximation power of moving least-squares. *Mathematics of computation*, 67(224):1517–1531, 1998.

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

[18] H. Lipschütz, M. Skrodzki, U. Reitebuch, and K. Polthier. Single-sized spheres on surfaces (S4). *Computer Aided Geometric Design*, 85:101971, 2021.

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

[20] N. Mellado, Q. Marcadet, L. Espinasse, P. Mora, B. Dutailly, S. Tournon-Valiente, and X. Granier. 3D-ARD: A 3D-Acquired Research Dataset, June 2020.

[21] N. J. Mitra, A. T. Nguyen, and L. J. Guibas. Estimating surface normals in noisy point cloud data. In *SCG '03*, pages 322–328, 2003.

[22] C. Öztireli, G. Guennebaud, and M. Gross. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. *Computer Graphics Forum*, 28(2):493–501, 2009.

[23] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

[24] J. R. Shewchuk. What is a good linear element? Interpolation, conditioning, and quality measures. In N. Chrisochoides, editor, *Proceedings of the 11th International Meshing Roundtable, IMR 2002, Ithaca, New York, USA, September 15-18, 2002*, pages 115–126, 2002.

[25] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023.

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