Differentiable surface triangulation

Marie-Julie Rakotosaona, Noam Aigerman, Niloy J Mitra, Maks Ovsjanikov, Paul Guerrero

Differentiable Surface Triangulation

MARIE-JULIE RAKOTOSAONA, LIX, École Polytechnique, France
NOAM AIGERMAN, Adobe Research, USA
NILOY J. MITRA, Adobe Research, University College London (UCL), United Kingdom
MAKS OVSJANIKOV, LIX, École Polytechnique, France
PAUL GUERRERO, Adobe Research, United Kingdom

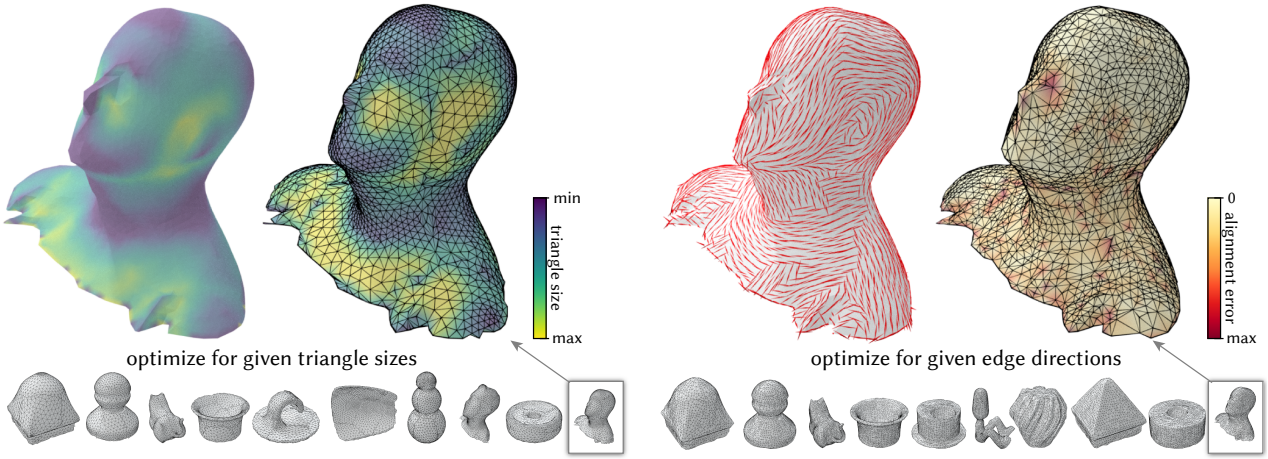

Fig. 1. We present a fully differentiable approach for optimizing triangle meshes both in 2D and on surfaces. Our approach allows to optimize the mesh using any differentiable objective function, based on vertex positions or shapes of triangles using continuous optimization techniques. Here we demonstrate meshes obtained on a surface by optimizing for (left) sizes of triangles to depend inversely on the mean absolute curvature value, and (right) alignment of triangle edges to the maximal principal curvature directions (i.e. triangle edges tend to follow the vector field). This optimization is done in a fully differentiable manner without any post-processing or combinatorial operations such as edge flips or vertex splits. Our framework is general and can be thus integrated within modern optimization and learning modules.

Triangle meshes remain the most popular data representation for surface geometry. This ubiquitous representation is essentially a hybrid one that decouples continuous *vertex locations* from the discrete topological *triangulation*. Unfortunately, the combinatorial nature of the triangulation prevents taking derivatives over the space of possible meshings of any given surface. As a result, to date, mesh processing and optimization techniques have been unable to truly take advantage of modular gradient descent components of modern optimization frameworks. In this work, we present a *differentiable surface triangulation* that enables optimization for any per-vertex or per-face differentiable objective function over the space of underlying surface triangulations. Our method builds on the result that *any* 2D triangulation can be achieved by a suitably perturbed weighted Delaunay triangulation. We translate this result into a computational algorithm by proposing a soft relaxation of the classical weighted Delaunay triangulation and optimizing over vertex weights and vertex locations. We extend the algorithm to 3D by decomposing shapes into developable sets and differentiably meshing each set with suitable boundary constraints. We demonstrate the efficacy of our method on various planar and surface meshes on a range of difficult-to-optimize objective functions. Our code can be found online: https://github.com/mrakotosaon/diff-surface-triangulation.

CCS Concepts: • **Computing methodologies** → **Shape analysis**.

Additional Key Words and Phrases: meshing, geometry processing, surface representation, neural networks

Authors' addresses: Marie-Julie Rakotosaona, LIX, École Polytechnique, France, mrakotos@lix.polytechnique.fr; Noam Aigerman, Adobe Research, USA; Niloy J. Mitra, Adobe Research, University College London (UCL), United Kingdom; Maks Ovsjanikov, LIX, École Polytechnique, France; Paul Guerrero, Adobe Research, United Kingdom.

1 INTRODUCTION

Triangle meshes are arguably the most predominant surface representation, both in geometry processing and computer graphics, as

well as in other fields such as computational geometry and topology. The popularity of triangle meshes comes from their simplicity, flexibility, and the existence of many data structures for efficient mesh navigation and manipulation [Boissonnat et al. 2000; Devillers 2002; Devillers et al. 2001; Toth et al. 2017]. Many methods have been developed to compute or modify triangulations of given surfaces or point clouds, while promoting properties such as alignment to shape features (e.g., ridges or creases), adapting sampling density to geometric detail, or triangle aspect ratio (see [Berger et al. 2017; Cazals and Giesen 2004] for an overview).

Unfortunately, as of now, no method has been proposed to enable a continuous, differentiable representation of triangulations. This is mainly due to the fact that in addition to the *continuous* spatial aspect - the position of each vertex - triangulations also have a *discrete* combinatorial component - the connectivity, i.e., the set of edges and triangles connecting the vertices. As a result, existing algorithms either optimize the mesh quality by moving the vertex locations while keeping their connectivity fixed [Nealen et al. 2006], re-mesh from scratch, or iterate between updating the vertex positions and their connectivity, e.g., [Hoppe et al. 1993; Tournois et al. 2008].

This lack of a unified differentiable representation is particularly unfortunate in light of recently-introduced gradient-based optimization frameworks such as Pytorch [Paszke et al. 2019] and TensorFlow [Abadi et al. 2016] for Machine Learning applications. These frameworks rely on the differentiablity of the pipeline and enable modular design. In absence of such a differentiable triangulation framework, current deep-learning pipelines either perform surface meshing during post-processing, or use formulations that are learned via proxies [Liao et al. 2018; Liu et al. 2020; Rakotosaona et al. 2021; Sharp and Ovsjanikov 2020], which typically do not give explicit access to the resulting triangle mesh structure.

In this work, we devise what we believe to be the first formulation for *differential triangulation*, enabling gradient-based optimization for per-face and/or per-vertex objectives, such as size and curvature alignment. Our approach is general, can be applied to manifolds represented in any explicit representation, is modular, and supports optimizing for any objective that can be expressed as a differentiable function with respect to triangle properties like size and angles.

The main technical challenge in devising a differentiable triangulation is developing a smooth representation that allows to control both the vertex positions and the (inherently-combinatorial) mesh structure, while also ensuring the resulting mesh is always a 2-manifold. Our core idea is to use the concept of a weighted Delaunay triangulation (**WDT**) [de Berg et al. 2000]. It considers a given set of vertices, along with per-vertex weights, which define a unique triangulation using a Voronoi-like partition of space.

In this paper we propose a *differentiable* weighted Delaunay triangulation (**dWDT**), by considering (arbitrary) triplets of vertices and whether they constitute a triangle in the triangulation defined by the weights and vertices. While in classic **WDT**, this existence receives a binary value, we generalize that definition by assigning inclusion scores to triangle membership, thus giving them a *soft* association. We demonstrate that this relaxation provides a unified control over both the vertices and the mesh structure, and can be used to directly optimize any (differentiable) objective function defined on the triangles. Intuitively, we define the triangle inclusion scores in terms of Voronoi diagram distances that represent how close a certain triangle is from inclusion into (or removal from) the triangulation. Represented as a continuous quantity, we can optimize triangle inclusion scores as a function of vertex positions and weights. Importantly, Memari et al. [2011] showed that, in 2D, *any* triangulation can be represented through a perturbation of a **WDT**, in other words, *any* triangulation can be reached by adjusting vertex positions and weights, and then applying a **WDT**. Therefore, our approach is both differentiable and generic, allowing to accommodate a wide range of mesh structures.

To apply our relaxation to 3D surfaces, we decompose the source into local patches, and then perform per-patch differentiable meshing with appropriate boundary constraints. For example, in Figure 1 we show triangulations obtained by optimizing for different objective functions, given the same original underlying surface models. The modular nature of our approach makes it easy to switch between target objective functions. Similarly, we can triangulate different surface representations (see Figure 9 for a triangulation of an analytic surface defined by a function).

We evaluate our method to produce 2D and 3D meshes optimized for a mix of target objective functions such as shape/size of triangles, and alignment to given vector fields, thereby highlighting that our approach is both more flexible, and can accommodate for more diverse objectives than alternative approaches.

2 RELATED WORK

Surface remeshing and triangle mesh optimization are both extremely well-studied problems in computational geometry, computer graphics, and related fields. Below we review methods most closely related to ours, and refer to recent surveys, including [Alliez et al. 2008; Cheng et al. 2016; Khan et al. 2020; Khatamian and Arabnia 2016] for a more in-depth discussion.

Simplification-based approaches. A common objective for surface remeshing is reducing the number of elements in the final mesh. As a result, especially early remeshing techniques, starting with the pioneering QEM approach [Garland and Heckbert 1997], often focused on preserving mesh quality during simplification (see [Garland and Heckbert 1997] for a survey of local methods). Such methods are typically based on edge-collapse operation followed by vertex position optimization, and have been extended both in terms of efficiency, e.g., [Hussain 2009; Ozaki et al. 2015], the use of various metrics [Ng and Low 2014] including feature preservation [Wei and Lou 2010], and even using spectral quantities [Lescoat et al. 2020] during edge collapse. However, such approaches are essentially greedy and typically do not allow to optimize mesh properties based on general structural criteria.

Local methods. A related set of methods includes approaches based on local mesh modification while aiming to improve the overall mesh quality, e.g., [Dunyach et al. 2013; Hu et al. 2016; Yue et al. 2007]. In addition to edge collapse, these local operators include edge flipping, edge splitting, and vertex translation. A prominent method in this category is real-time adaptive remeshing (RAR) [Dunyach et al. 2013], which uses an adaptive sizing function and edge flipping to optimize the mesh quality and vertex valence. This framework

was recently extended for efficient *error-bounded* remeshing [Cheng et al. 2019] through a use of a range of powerful local refinement operations. Similarly, Explicit Surface Remeshing (ESR) [Surazhsky and Gotsman 2003] is another efficient method for remeshing based on local refinement operations coupled with angle-based smoothing. The more recent Instant Meshes [Jakob et al. 2015] technique advocates using local optimization and smoothing, while aiming to optimize potentially global consistency. This results in a powerful and efficient framework, capable of handing both isotropic triangular or quad-dominant meshes. Nevertheless, as with other local techniques the topology (i.e. the connectivity between vertices) and geometry are handled separately, preventing a unified differentiable, global mesh optimization.

Delaunay and CVT-based methods. Another powerful set of remeshing methods, more closely related to our approach are based on Delaunay triangulations, and centroidal Voronoi tessellations (CVT). The former category includes approaches based on triangle refinement by flipping non-locally Delaunay (NLD) edges [Dyer et al. 2007] and defining an *intrinsic* Delaunay triangulations [Fisher et al. 2007]. Furthermore, global optimization techniques have also been used for finding optimal Delaunay triangulations [Chen and Holst 2011] under the assumption that vertex connectivity is fixed. In a different line of work, centroidal Voronoi tessellations (CVT) have been used for finding an approximately uniform vertex distributions, so that their Voronoi diagram (and thus its dual, the Delaunay triangulation) is well-shaped, e.g., [Du et al. 2003; Wang et al. 2015; Yan et al. 2009] among many others. Such methods have also been extended, for example, to explicitly penalize obtuse and sharp angles [Yan and Wonka 2015] and to *anisotropic* remeshing by embedding in an appropriate (e.g., feature or curvature-aware) space [Lévy and Bonneel 2013]. Nevertheless, the final shape of the triangulation is difficult to control using these methods, and it is not easy to combine multiple objective functions in a coherent optimization strategy.

Optimization-based approaches. Finally we also note methods based explicitly on optimizing an objective. This includes both local, e.g., [Dunyach et al. 2013; Hoppe et al. 1993] and global optimization, e.g., [Marchandise et al. 2014; Valette et al. 2008] strategies (see also Section 4.7 in [Khan et al. 2020]). Existing optimization strategies most often rely on either smoothness energies [Desbrun et al. 1999; Fu et al. 2014; Taubin 1995], use sampling [Fu and Zhou 2009] or a variant of CVT, e.g., [Yan et al. 2014] to optimize vertex positions. In both cases, while the *positions* of the vertices can be optimized, the connectivity is only defined implicitly and updated separately, typically without explicitly taking into account the optimization objective. More fundamentally, the mesh structure is purely combinatorial, preventing the use of powerful tools based on differentiability.

In contrast to these approaches, we propose a fully differentiable framework that allows to jointly optimize for both vertex positions and triangle mesh connectivity, by using a soft version of the weighted Delaunay triangulation (**WDT**). Our method is inspired by theoretical results demonstrating that in 2D any triangulation can be represented through a perturbation of a **WDT** [Memari et al. 2011]. Importantly, the same result does not hold for the standard Delaunay triangulation, and therefore optimizing over the *weights* of the **WDT** as well as the *vertex positions* allows significantly more control over the shape of the final triangulation and even allows ignoring some input points if they are deemed unnecessary (i.e., high weights) in the final triangle mesh.

Importantly, the differentiable nature of our approach allows optimizing for a range of criteria jointly, simply by formulating a single (differentiable) objective function. Furthermore, it enables optimization of both vertex positions and criteria that depend on the connectivity in a unified framework. Finally, our differentiable meshing block can be also ultimately be used as part of a larger, differentiable shape processing or design system.

Weighted Voronoi and power diagrams. Weighted Voronoi diagrams are also known as *power diagrams*, and have been researched extensively in the context of triangulations [Glickenstein 2005]. In computer graphics, they have been used for various tasks such as computing blue noise [De Goes et al. 2012], or simulating fluid dynamics [De Goes et al. 2015]. [Goes et al. 2014; Memari et al. 2011; Mullen et al. 2011] considered power diagrams in the context of formulating different triangulation *duals*. They also propose to optimize objectives on the dual. This is a different context and use-case than our differentiable formulation, which is geared towards a gradient-guided optimization of arbitrary geometric objectives on the *triangulation*.

Differentiability in computer vision. Recently, with the success of deep learning in computer vision, making common operations differentiable has started to gain research interest. In particular relaxing hard condition for deep learning purposes into soft formulations has been used for tasks such as RANSAC [Brachmann et al. 2017], rendering [Liu et al. 2019] or shape correspondence [Marin et al. 2020] among others. Similarly to these methods, we present a soft formulation of triangle existence.

3 METHOD

Let (V', T) represent a triangulation of a surface in 3D space, with $V' = (v'_1, \ldots, v'_n), v'_i \in \mathbb{R}^3$ vertices, and T its triangular faces. A common strategy for triangulating a manifold surface is to first find a 2D parameterization that maps the surface to a planar 2D domain, then sample a set of vertices $V = (v_1, \ldots, v_n), v_i \in \mathbb{R}^2$ in the 2D domain, and compute a triangulation which respects the chosen vertices. Our method relies on the ubiquitous Delaunay triangulation [Cheng et al. 2016; Delaunay et al. 1934] (**DT**), used for triangulating a given 2D vertex set. We denote it as $T = \mathbf{DT}(V)$. A Delaunay triangulation always includes all chosen vertices, and is, uniquely defined with respect to them, as long as the points are in general position. In order to gain more control over the triangulation, one can consider a *weighted* Delaunay triangulation [Aurenhammer 1987; Toth et al. 2017] $T = \mathbf{WDT}(V, W)$, where each vertex v_i has a scalar weight w_i, with $W = (w_1, \ldots, w_n)$. Traditional methods for computing a **WDT** are typically not differentiable, as the space of all possible faces is combinatorial.

We propose a differentiable weighted Delaunay triangulation $\mathbf{dWDT}(V, W)$ that is differentiable with respect to both the vertex positions V and the weights W. In conjunction with a parameterization m that defines a bijective and piecewise differentiable mapping m from a surface in 3D to a 2D parameter space, **dWDT** enables a

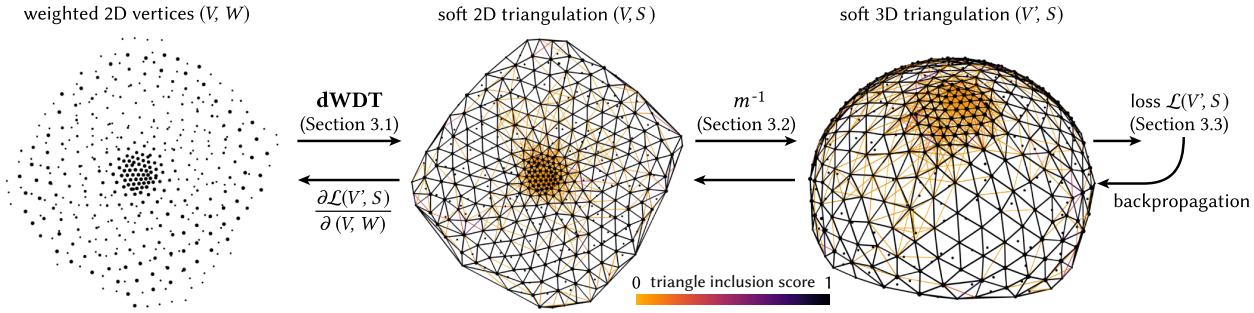

Fig. 2. **Overview of our approach.** We propose a differentiable weighted Delaunay Triangulation (**dWDT**) to create a soft triangulation from a set of 2D vertices V with associated associated weights W (shown as marker size). In the soft triangulation, triangles have inclusion scores S of being part of the triangulation. We illustrate triangle inclusion scores as edge colors (using the largest inclusion score of the two adjacent faces) and only show triangles with inclusion score > 0.001. The 2D vertices V are lifted to form a soft 3D triangulation on the manifold's surface using a fixed mapping m^{-1}. Since the pipeline is fully differentiable, we can propagate gradients of any differentiable loss on the 3D triangulation back to the vertex positions V and weights W. Note that by choosing appropriate weights W, our network can ignore points and produce a triangulation over a subset of points, if desired.

differentiable pipeline for triangulating 3D domains. We describe our differentiable triangulation approach in two parts (see Figure 2). First, in Section 3.1, we describe the differentiable weighted 2D Delaunay triangulation **dWDT**. In this part, we first focus on the definition of **DT** and the existence of triangles, w.r.t. vertex positions and weights, and then replace the binary triangle existence function with a smooth *triangle inclusion score*, again defined w.r.t the vertex positions and weights, in a way that naturally follows from the definition of **DT**. This yields a *soft* and differentiable notion of a triangulation that can easily be generalized to a weighted Delaunay triangulation. Then, in Section 3.2, we describe a parameterization m that maps between a manifold surface and our 2D Delaunay triangulation to obtain **dWDT** on 3D surfaces, before describing the losses and optimization setup in Section 3.3.

3.1 Differentiable Weighted Delaunay Triangulation

Assume we are given a set of vertices $V = \{v_1, ..., v_{|V|}\}$ with $v_j \in \mathbb{R}^2$. Consider the set of all possible triangles defined over these vertices, i.e., all possible triplets of vertices:

$$T^* = \left\{ (v_j, v_k, v_l) \,|v_j, v_k, v_l \in V \right\}. \tag{1}$$

Any triangulation of the vertices V is a subset of all possible triangles $T \subset T^*$ on V, and we can consider the triangulation's *existence* function $e : T^* \rightarrow \{0, 1\}$, defined for any triplet $t_i \in T^*$ as

$$e_i = \begin{cases} 1 & t \in T \\ 0 & t \notin T. \end{cases} \tag{2}$$

From this perspective, the binary and discrete existence function is the cause of the combinatorial nature of the triangulation problem. Hence, our main goal is to define a *smooth* formulation in which this function is differentiable as to enable gradient-based optimization. We achieve this by extending **WDT** to the smooth setting.

Towards gaining intuition into **WDT**, let us first consider the classic, non-weighted *Delaunay triangulation* $\mathbf{DT}(V)$ of a given set of vertices V. This triangulation is defined by considering each possible triangle $t \in T^*$ and deeming it as part of the triangulation $\mathbf{DT}(V)$ if and only if its circumcenter is the shared vertex of the three Voronoi cells centered at the triangle's vertices (see Figure 3 for an illustration). The Voronoi cell of vertex v_j is defined as the set of points in $\mathbb{R}^2$ closer to v_j than to any other vertex $v_k \in V$.

Said differently, each pair of vertices (v_j, v_k) divides the 2D plane into two half-spaces: the set of points closer to v_j, denoted as $H_{j<k}$, and the set of points closer to v_k, denoted as $H_{k<j}$. The Voronoi cell a_j centered at v_j is defined as the intersection of half-spaces $a_j := \bigcap_{k \neq j} H_{j<k}$. The triangle circumcenter is the intersection point of the three half-space boundaries between the three vertex pairs that define its edges. Hence, we can define the existence function of the Delaunay triangulation for a triplet of vertices, $t_i = (v_j, v_k, v_l)$ with circumcenter c_i as

$$e_i = \begin{cases} 1 & \text{if } c_i = a_j \cap a_k \cap a_l \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Parameterizing Triangle Existence with respect to V. We are interested in how the triangulation T changes as the vertex positions are changed - namely, we aim to analyze the range of vertex positions that do not change its membership function e_i of a triangle t_i.

For any triangle $t_i = (v_j, v_k, v_l)$, we consider the three *reduced* Voronoi cells $a_{j|i}, a_{k|i}, a_{l|i}$ respectively around the triangle's vertices v_j, v_k, v_l, where we define a reduced Voronoi cell $a_{j|i}$ centered at the triangle vertex v_j as the Voronoi cell created by ignoring the two other vertices of the triangle, v_k and v_l (see Figure 3). The triangle t_i is part of the triangulation T as long is its circumcenter c_i remains inside the reduced Voronoi cells around its vertices. Similarly, t_i is not part of T as long as its circumcenter remains outside its three reduced Voronoi cells. Note that, by construction, the circumcenter simultaneously enters or exits the three reduced Voronoi cells. Thus, we can re-formulate the triangle existence e_i as:

$$e_i = \begin{cases} 1 & \text{if } c_i \in a_{x|i} \text{ for any } x \in \{j, k, l\} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Continuous Triangle Inclusion Scores. We now turn to making **DT** differentiable by *relaxing* the binary existence function e_i defined

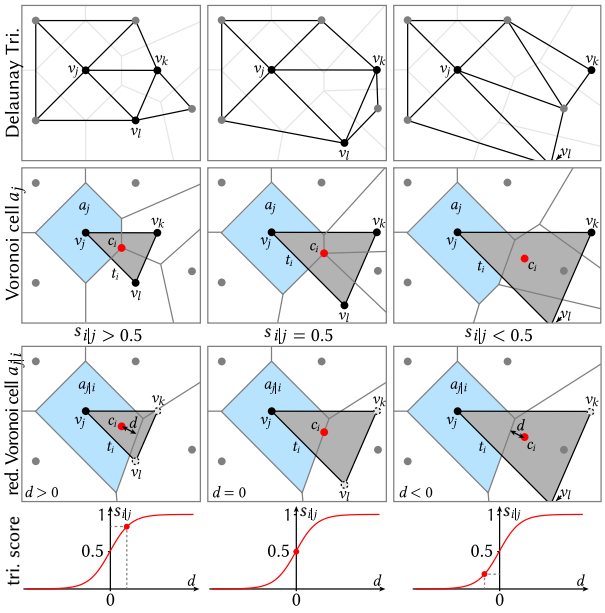

Fig. 3. **Triangle inclusion score and reduced Voronoi cell.** A reduced Voronoi cell for a given triangle t_i at a vertex v_j is constructed from the point set that excludes the two other vertices of the triangle. A triangle t_i exists in the Delaunay triangulation, as long as its circumcenter c_i remains inside the reduced Voronoi cell. We base triangle inclusion scores $s_{i|j}$ on the signed distance from c_i to the boundary of the reduced Voronoi cell.

in Equation (4) into a continuous inclusion score function, s_i, denoting the inclusion score a triangle $t_i \in T^*$ to exist as a member of the triangulation T, defined with respect to vertex positions. The inclusion scores are based on the signed distance of the triangle circumcenter to the boundary of the reduced Voronoi cells at the triangle vertices: considering a single vertex v_j of the triangle t_i, and its reduced Voronoi cell $a_{j|i}$, the inclusion score is defined as:

$$s_{i|j} := \sigma(\alpha\, d(c_i, a_{j|i})), \tag{5}$$

where d is the signed distance (positive inside, negative outside) from a point c_i to the boundary of a reduced Voronoi cell $a_{j|i}$, and α is a scaling factor for the width of the Sigmoid σ (we use $\alpha = 1000$ in all experiments). The Sigmoid gives a smooth transition from an inclusion score close to 1 inside the reduced Voronoi cell to an inclusion score close to 0 outside, with an inclusion score 0.5 if the circumcenter lies on the boundary of the reduced Voronoi cell, i.e., exactly when the discrete triangle membership changes. The triangle inclusion score s_i can then be defined as the average over the three inclusion scores at its vertices v_j, v_k, and v_l:

$$s_i = \frac{1}{3}(s_{i|j} + s_{i|k} + s_{i|l}). \tag{6}$$

Note that since the circumcenter simultaneously enters/exits the reduced Voronoi cells around each vertex, all three inclusion scores equal 0.5 at a discrete membership transition.

For each triangle t_i, we store the triangle inclusion score s_i and the three inclusion scores $s_{i|j}$, $s_{i|k}$ and $s_{i|l}$ defined for its three vertices, yielding a *soft* 2D triangulation (V, S) with inclusion scores S. We store the inclusion scores $s_{i|j}$ in addition to the triangle inclusion scores s_i, since most losses that we use are defined on vertices where using a triangle's vertex inclusion scores is more convenient. We can, subsequently, convert this soft triangulation into a discrete 2D triangulation (V, T), by selecting all triangles where $s_i > 0.5$. This gives us the same results as the discrete **DT**, the final triangulation is guaranteed to be manifold.

Since the number of all possible triangles T^* grows cubically with the vertex count, we reduce the number of triangles under consideration by observing that vertices in the triangles of a Delaunay triangulation are typically within the k-nearest neighbors of each other, for some small k (we use $k = 80$ in all experiments). Thus, at each Voronoi cell a_j, we only consider triangles that are within the k-nearest neighbors of v_j and set all other triangle inclusion score implicitly to 0. Note that since the 2D vertices V change positions during optimization, we recompute nearest neighbours after each iteration of our algorithm.

Weighted Delaunay triangulation. Our relaxed formulation of the Delaunay triangulation can naturally be extended to the *weighted* Delaunay triangulation **WDT**, where weights are associated to each vertex. The weights allow shifting the boundary between the two half-spaces $H_{j<k}$ and $H_{k<j}$, by the relative weights w_j and w_k of the two vertices. The weighted half space $H_{j<k}$ is defined as the set of points $x \in \mathbb{R}^2$ where

$$\|x - v_j\|_2^2 - w_j^2 \leq \|x - v_k\|_2^2 - w_k^2 \tag{7}$$

so that a larger weight pushes the boundary away from the vertex. This allows generalizing the definition of the Voronoi cell to a weighted Voronoi cell. As a result, the existence function Equation (4) and the inclusion score $s_{i|j}$ in Equation (5) can be used as-is, with the modified definition of the half-planes, and considering the *weighted* circumcenter of the triangle. This makes the inclusion scores S of the soft triangulation (V, S) a function of both the vertex position V and their weights W.

Thus, weights enable further control over the resulting triangulation, by enabling modifications the Voronoi cells (and therefore, the triangulation itself). In fact, note that it is even possible for a vertex to be excluded from a **WDT** (i.e., not be part of any triangle), if the weight difference to any other vertex is so large that the boundary line between the two half-spaces shifts past one of the vertices - a property not possible with classical Delaunay triangulation. We will make use of this property to allow our method to ignore vertices deemed unnecessary, hence producing triangulations with a reduced number of vertices. In the following, we consider the weighted triangle circumcenters, denoted by c_i, and the weighted Voronoi cells, denoted by a_i.

3.2 3D Surface Parameterization

So far we have defined differentiable triangulations of 2D sets of vertices. In order to apply our **dWDT** on a 3D surface $\mathcal{M}$, we reduce the problem to a set of 2D (triangulation) problems.

First, we construct a bijective piecewise differentiable mapping m between the manifold and the 2D plane, i.e., a 2D parameterization. Next, we elaborate on the computation of this parameterization. As a pre-process, since we are not concerned with the original triangulation but only the underlying surface it represents, we initially remesh input models using isotropic explicit remeshing [Cignoni et al. 2008] to yield meshes constituting between 3.5 − 4.5K triangles. We normalize each model to unit area. We then decompose the manifold into a set of separate patches $\{\mathcal{P}_1, \mathcal{P}_2, \dots\}$ that can be individually parameterized with less distortion than the whole shape. Individual patches are found with a spectral clustering approach [Ng et al. 2002], using the adjacency matrix for affinity. We used 10 patches in all experiments.

Then, we construct a low-distortion mapping m between the surface of a patch $\mathcal{P}_i$ and the 2D plane using Least-Squares Conformal Maps [Lévy et al. 2002] (LSCM). To lower the distortion of the mapping for patches that are far from developable, we first measure the distortion as the deviation of the local scale factor from the global average. Patches with high distortion are cut along the shortest geodesic between the area of maximum distortion and any existing boundary. This process is repeated until the maximum distortion of all patches, measured as the ratio between local scale and global average is above 15% and the mapping is bijective. Finally, we normalize the 2D parametrization of each patch to have equal average edge length. We compute this mapping once, as a preprocess, and reuse it in all steps of the optimization.

Differentiable 3D Surface Triangulation. Given the mapping m, we can pull back the computed 2D triangulation (V, T) to a part of the 3D surface $\mathcal{P}_i$ using the inverse mapping m^{-1}. Thus, our differentiable triangulation of a 3D surface patch is defined as:

$$(V', S) := \big((m^{-1}(v_1), \dots, m^{-1}(v_n)),\ \mathbf{dWDT}(V, W)\big), \tag{8}$$

which gives us the soft 3D triangulation (V', S) that consists of a set of 3D vertices V' and triangle inclusion scores S. Note that the inclusion scores are differentiable functions of the 2D vertices V and their weights W, and that we can obtain a manifold discrete mesh at any time by selecting all triangles with inclusion scores > 0.5. Since the mapping m is piecewise differentiable, any loss $\mathcal{L}$ can be applied directly to the 3D vertices V' and triangle inclusion scores S, allowing gradients to propagate back to the parameters V and W that define V' and S. We highlight that similarly to Leaky ReLU activations, the piecewise differentiability does not significantly impact optimization. We discuss the losses we use in our experiments in Section 3.3.

Boundary preservation. Special care must be taken to preserve the boundary of each patch, so that putting the patches back together does not result in gaps or overlaps. We use a two-part strategy to ensure pieces fit back together. First, we define a loss that repels vertices from the boundary of a patch, which we describe in Section 3.3. Second, we perform a post-processing step that cuts the 2D mesh (V, S) along the 2D boundary, based on a triangle flipping strategy along the boundary. Namely, we use the simple strategy described in [Sharp and Crane 2020] between consecutive boundary points of the optimized patches. The boundary between patches is therefore kept fixed before and after the optimization step.

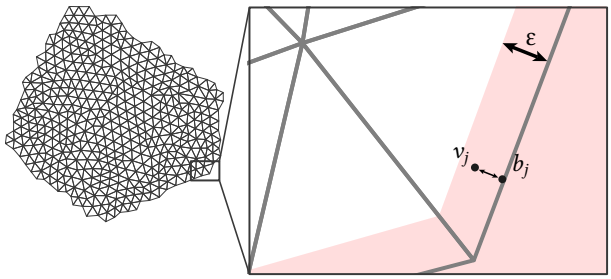

Fig. 4. **Boundary repulsion loss.** The repulsion loss is non-zero below a (signed) distance ϵ from the boundary. Non-boundary vertices inside the red region are pushed towards the center of the patch.

3.3 Losses and Optimization

Our differentiable triangulation allows us to optimize a triangular mesh on a surface in 3D using any differentiable loss defined on the 3D vertex positions V and triangle inclusion scores S. We experiment with several different losses, combinations of which are useful for both traditional applications, as well as novel ones, as we experimentally show in Section 4.

The *triangle size loss* $\mathcal{L}_s$ encourages triangles to have a specified area:

$$\mathcal{L}_s(V', S) := \frac{1}{\sum_{i,j} s_{i|j}} \sum_{i,j} s_{i|j} \left(0.5\, \|(v'_k - v'_j) \times (v'_l - v'_j)\|_2 - A(v_j)\right)^2, \tag{9}$$

where v'_j, v'_k, and v'_l are the 3D vertices of triangle t_i, and $A(v_j)$ is the target area at vertex v_j, where A is defined as a continuous function over the 3D surface. This loss allows us, for example, to coarsen a triangulation, when used in conjunction with other losses. Note that the size of the triangles is not constrained by the initial number of vertices - due to the **WDT** our optimized result can contain fewer vertices than the initial triangulation.

The *boundary repulsion loss* $\mathcal{L}_b$ encourages vertices to stay inside the 2D boundary of the patch $\mathcal{P}$ during the optimization:

$$\mathcal{L}_b(V, \mathcal{P}) := \frac{1}{|V|} \sum_j e^{\epsilon - \min\left(\epsilon,\ (v_j - b_j) n_j^b\right)}, \tag{10}$$

where b_j is the point on the boundary closest to the vertex v_j and n_j^b is the 2D boundary normal at that point (pointing inward). The repulsion loss is non-zero below a (signed) distance ϵ from the boundary as we show in Figure 4. We set ϵ to 0.01 in our experiments. Note that we do not use our triangle inclusion scores in this loss, since we want all vertices to remain inside the boundary, irrespective of inclusion scores. We note that since $\mathcal{L}_b$ has a local effect and does not rely on global properties of the patch, patches can be non-convex.

The *angle loss* $\mathcal{L}_a$ encourages triangles to be equilateral.

$$\mathcal{L}_a(V', S, \mathcal{P}) := \frac{1}{\sum_{i,j} s_{i|j}} \sum_{i,j} s_{i|j} \left|\cos(\angle_j) - \cos(\pi/3)\right|, \tag{11}$$

where $\angle_j$ is the corner angle of triangle t_i including vertex v_j. Note that this loss can be modified to produce isosceles triangles.

The *curvature alignment loss* $\mathcal{L}_c$ encourages two edges per vertex to align to the two directions of the minimum principal curvature vector field C. We define it as,

$$\mathcal{L}_c(V', S, \mathcal{P}, C) := \frac{-1}{\sum_{i,j} s_{i|j}} \sum_j \Big(\\ \mathrm{LSE}(\cup_{i \in \mathcal{N}_j} \{C(v_j) \cdot h_{jk}\, s_{i|j}, C(v_j) \cdot h_{jl}\, s_{i|j}\}) \\ + \mathrm{LSE}(\cup_{i \in \mathcal{N}_j} \{-C(v_j) \cdot h_{jk}\, s_{i|j}, -C(v_j) \cdot h_{jl}\, s_{i|j}\}) \Big) \\ \text{with } h_{jm} = (v'_j - v'_m)/\|v'_j - v'_m\|_2, \tag{12}$$

where $\mathcal{N}_j$ are the triangles adjacent to v'_j, and v'_j, v'_k, v'_l are the 3D vertices of triangle t_i. LSE denotes the smooth maximum function LogSumExp over the weighted alignment scores of all edges adjacent to vertex v'_j, where each triangle contributes two edges corresponding to h_{jk} and h_{jl}. Intuitively, we want to maximize the alignment of the best-aligned edge in a star of each vertex, for both the positive and negative target guidance direction $C(v_j)$, which is the principal curvature field evaluated at v_j.

Optimization. Given a loss $\mathcal{L}$, as a sum of a selection of the terms above, we optimize the 3D mesh M, parameterized by the 2D vertex positions V and vertex weights W. Since our framework is completely differentiable, we use the Adam [Kingma and Ba 2015] optimizer. We initialize all vertex weights with random values and use the mapping of the input mesh vertices to 2D as the initial 3D vertex positions. We use a learning rate of 0.0001 in all experiments. Please refer to the supplementary video for evolving triangulations over optimization iterations.

4 RESULTS

We next describe experiments that highlight the key advantage of our method - differentiability, which enables plugging in and mixing any combination of differentiable losses, circumventing the need to design a specialized optimization method for each loss combination. Practically, the experiments show the efficacy of our method, and its ability to produce superior results than state-of-the-art methods that are specifically tailored to those specific applications. Code of our method is available at anonymous.code.

4.1 Customized Triangulation

Most triangulation tasks are formulated via user-provided requirements that are imposed on the resulting triangulation, such as desired triangle sizes or edge alignment. We employ our differentiable losses in two common scenarios, shown in Figures 1 and 5. We evaluate our method in both scenarios on 140 randomly selected meshes (among those with genus 10 or less) sampled from Thingi10k [Zhou and Jacobson 2016].

(i) Triangle size. We first optimize the triangulation to match a given distribution of triangle sizes, represented as a scalar field over the surface. We chose to assign sizes that are the reciprocal of the mean absolute curvature value, so that high curvature regions receive a finer tessellation than lower-curvature regions. We sum the losses $\mathcal{L}_s$, $\mathcal{L}_b$, and $\mathcal{L}_a$ with weights 0.5, 500, and 10^7, respectively, in order to scale each loss to the same range. Qualitative results are shown in the left half of Figure 5.

As evaluation metric, we take the absolute difference between the resulting triangle size and the target size distribution. Since we are interested in the distribution of relative triangle sizes rather than the absolute sizes, we normalize the triangle sizes per model to have zero mean and unit standard deviation. To compare our triangle sizes to the continuous target size distribution, triangle size at each vertex is defined as the average size of all adjacent triangles. In Figure 5, normalized triangle sizes are shown as colors while the numbers below each result show the RMSE over all vertices.

We compare our method to the remeshing method of Loseille [2017], a state-of-the-art method for remeshing that can be guided by a given triangle size field, and show a qualitative comparison on a subset of shapes in Figure 5. In most cases our method can reproduce the target size distribution more accurately. Note, for example, the size distribution on the top of the pawn, on the heads, on the rim of the hat and on the cat's hind.

(ii) Vector-field alignment. In our second scenario, we optimize 3D meshes with the loss $\mathcal{L}_c$ that encourages edges to align with a given vector field. We chose to use minimum principal curvature directions to encourage meshes which edges that adhere to ridge lines and geometric features. At the same time, we emphasize that any other user-prescribed field could be used as well. We minimize the loss $\mathcal{L}_c$ combined with the boundary repulsion loss $\mathcal{L}_b$ with weights of 1 and 500, respectively. Qualitative results are shown in the right half of Figure 5.

As evaluation metric, we take the absolute angular difference between both the positive and negative prescribed curvature direction at each vertex and the best-aligned edge. We compare with Instant Meshes [Jakob et al. 2015], a method specialized to creating feature-aligned equilateral triangulations. Since Instant Meshes is designed to align to sharp features and is not well defined near flat regions or umbilical points, we weight the per vertex-alignment error using the following term:

$$w_j = \frac{|k_1^j - k_2^j|}{0.5 * (|k_1^j| + |k_2^j|)}, \tag{13}$$

where k_1^j and k_2^j are the signed principal curvature magnitudes at vertex j. Intuitively, this term reduces the influence of regions that are nearly flat or umbilical, so as to not penalize the baseline in those regions unfairly. In Figure 5, edge alignment errors are shown as colors while the numbers below each result show the RMSE over all vertices.

Our general-purpose triangulation achieves significantly better alignment, as can be seen by the significantly lower color-coded and average error on all models. While the baseline method of [Jakob et al. 2015] generates triangles that are very close to equilateral, the alignment with the curvature directions suffers, as can be seen on the lower part of the pawn, where none of the edges align well with the curvature directions. Similarly, for the cylindrical hat, our method generates edge-loops "hugging" the cylinder, while Instant Meshes does not present such edge-loops. On more organic models, such as the cat and human, lack of alignment is even more evident, e.g., on the human's brow.

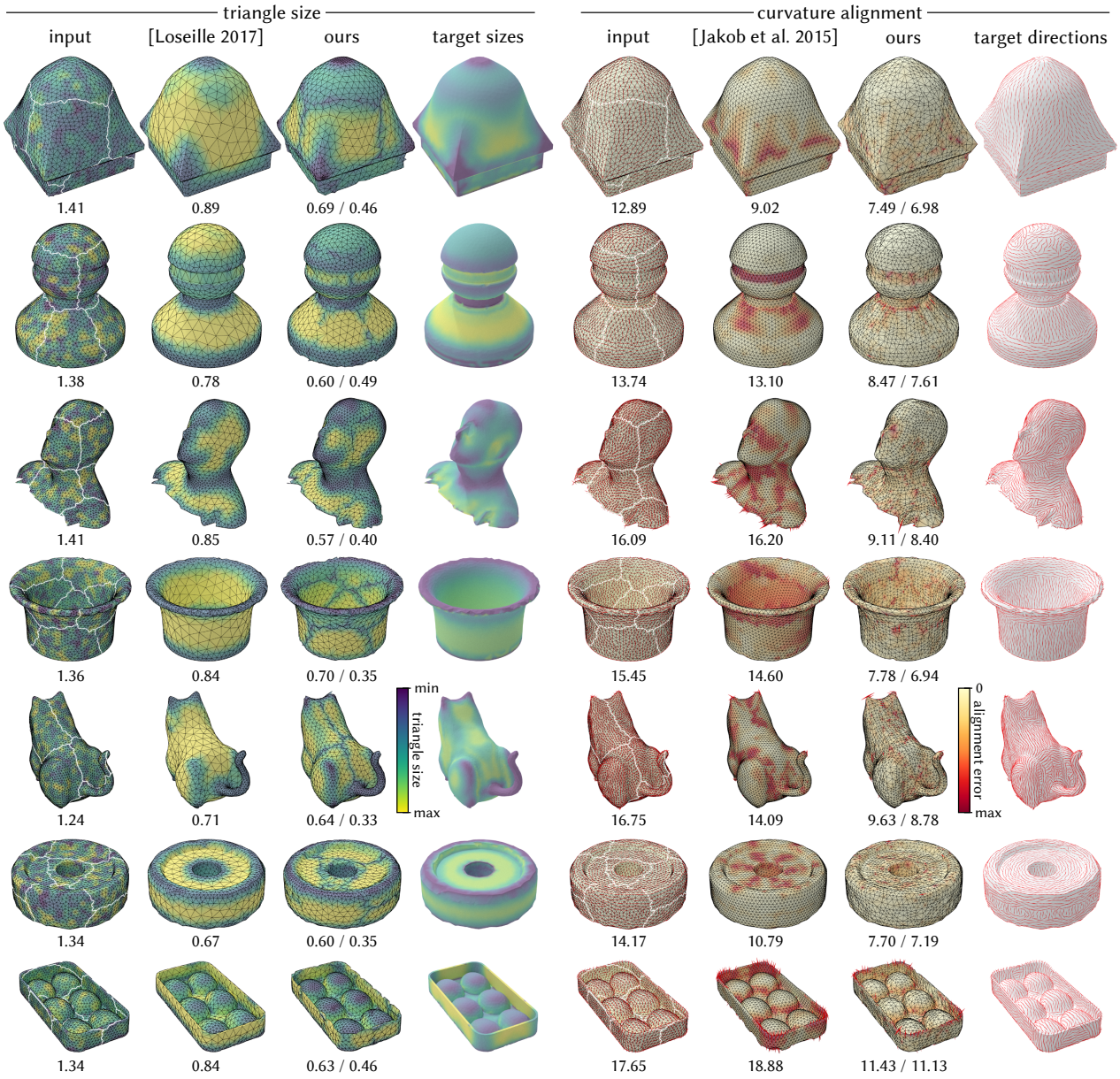

Fig. 5. **Qualitative Results.** We show two applications of our approach. In the left half of the figure we optimize for given target triangle sizes, and compare with a state-of-the-art remeshing method [Loseille 2017] (triangles are colored according to size). In the right half, we optimize for edges that are aligned to the principal curvature directions and compare with Instant Meshes [Jakob et al. 2015] (colors illustrate alignment errors). Boundaries of the patch decomposition are shown as white lines on the input meshes. The average error is given below each result - for our method we give the error with / without faces adjacent to a patch boundary. Note that our differentiable triangulation more accurately satisfies the target triangle sizes or edge directions.

Quantitative evaluation. We further evaluate our method on our complete dataset of 140 meshes taken from Thingi10k [Zhou and Jacobson 2016]. The quantitative results in Table 1 show the RMSE of the metrics described above over all vertices and all shapes in the dataset. Since the vertices at the boundary of our patches cannot fully be optimized with our approach, we provide errors computed both with and without the vertices at the patch boundaries. In both cases and in both applications, our method approximates the correct

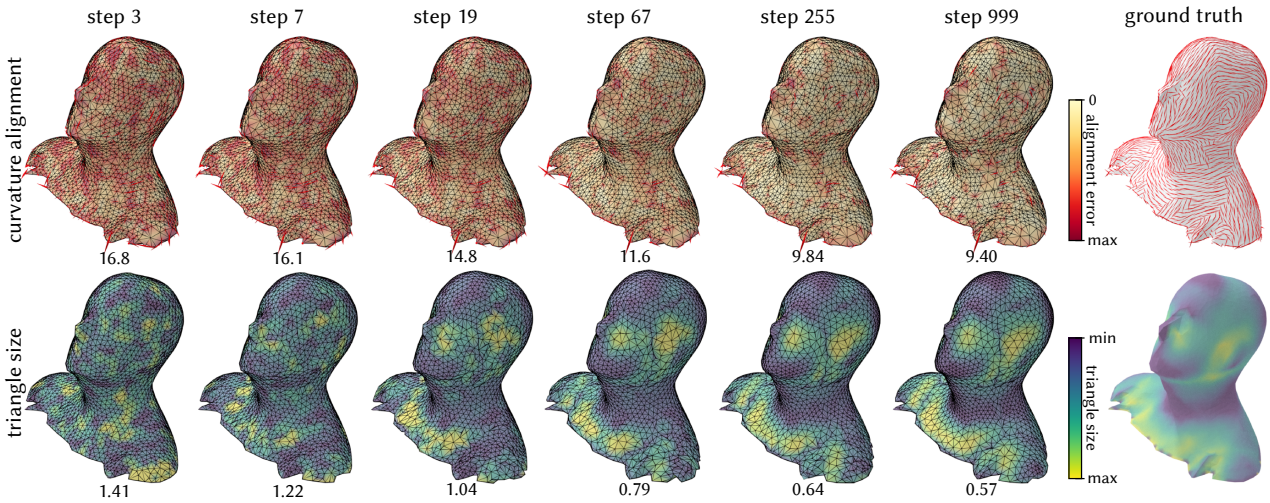

Fig. 6. **Optimization steps.** We show our results at different optimization steps for curvature alignment (top row), and triangle size (bottom row).

triangle sizes and edges directions significantly better than the state-of-the-art methods [Loseille 2017] and [Jakob et al. 2015].

Table 1. **Quantitative results.** We compare both the triangle size and curvature alignment applications to state of the art remeshing methods. For our results, we provide values computed both with and without the boundary triangles.

input mesh	[Loseille 2017]	ours w/o bound.	ours
1.320	0.865	0.499	0.686

triangle size

input mesh	[Jakob et al. 2015]	ours w/o bound	ours
14.043	11.850	7.919	8.4617

curvature alignment

4.2 Optimization

Choice of optimizer. We evaluate the effect of different optimization methods, comparing ADAM, LBFGS, and Simulated Annealing. In Figure 7 we show results on a 2D triangulation example where we optimize for both triangle sizes, and alignment to a custom vector field. We run each optimizer for 1000 steps and observe that while LBFGS can achieve better performances on some patches, ADAM produces good results more consistently, and hence we opted to use it in all our experiments. We use Simulated Annealing (SA) with the discrete mesh representation instead of our formulation as SA does not handle gradients. After computing the non differentiable weighted Delaunay triangulation, we minimize the discrete version of our losses: for instance we align existing edges to the curvature vector field and fit the area of existing triangles to the target area function. Both gradient-based methods perform significantly better than the non-gradient based method, Simulated Annealing, suggesting that our search space is typically too complex to allow for a more random search strategy that is not guided by gradients. We included the comparison to the non-gradient-based simulated annealing to show gradient-based methods are more apt for this problem; however, putting performance aside, we note that simulated annealing cannot accomplish the main goal of our work, which is to devise a triangulation module that can be used within differentiable optimization frameworks (e.g., PyTorch [Paszke et al. 2019]).

Optimization process. In Figure 6 we show the evolution of the triangulation through the optimization steps. The gradual change shows that indeed our differential triangulation enables gradient-based optimization which smoothly decreases the energy towards a local minimum. Please refer to the supplemental video for more detailed visualizations of the optimization process.

4.3 Loss blending

As an important advantage, our method naturally enables blending and interpolating the relative weights placed on different loss terms, such as triangle size and adherence to equilateral triangles. We show the plot of energies with respect to such a blending in Figure 8 using the aggregated loss term defined as,

$$\mathcal{L}(V', P) := t \times \mathcal{L}_a + (1 - t) \times \mathcal{L}_s \tag{14}$$

with t being the blending weight. We evaluate over 7 values of the weight on a subset of 7 models from our dataset. This allows us to easily trade off between characteristics for the triangulation. Note that this was not previously possible for specialized methods targeted towards individual tasks.

4.4 Method of vertex initialization

We compare our method with an alternative vertex initialization technique in Table 2. Given fixed per-patch boundary vertices, we uniformly sample the remaining vertices on the 3D surface using

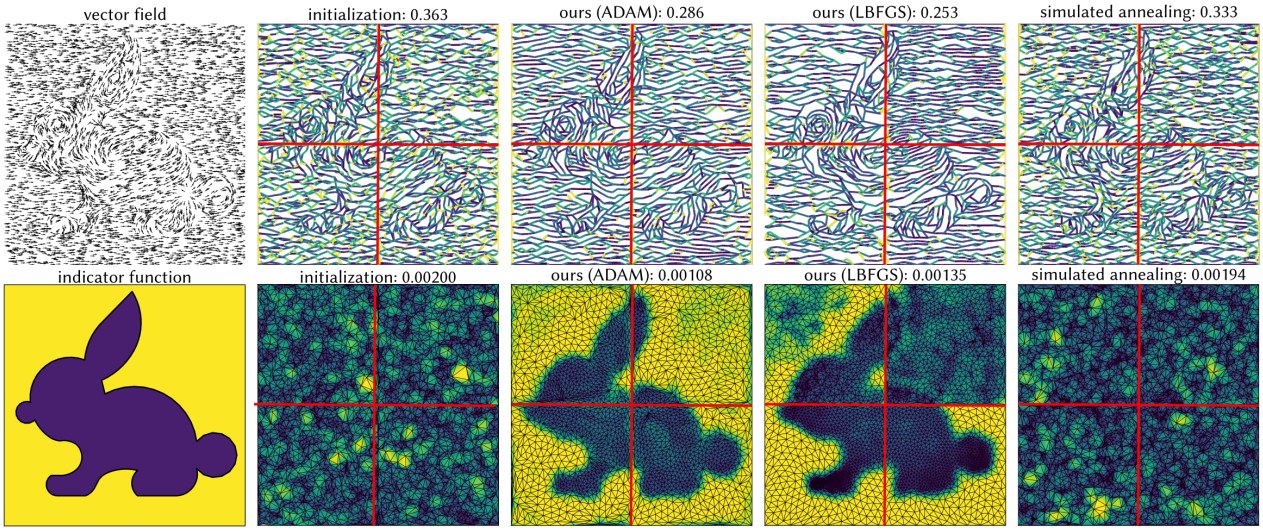

Fig. 7. **Comparing different optimizers.** We compare ADAM, LBFGS and Simulated Annealing on a 2D mesh. We start from a 2D mesh with random vertices. In the top row, we optimize edges to align with a given vector field. The best-aligned edges are color-coded according to the alignment error (blue is lowest error, yellow largest error). The average alignment error is shown at the top. In the bottom row, we optimize triangle areas to align with a given size field. Vertices are color-coded according to the average neighboring triangle area (blue are smaller triangles, yellow larger triangles), with RMSE shown at the top. Note how the two gradient-based optimizers ADAM and LBFGS perform significantly better than the gradient-less simulated annealing.

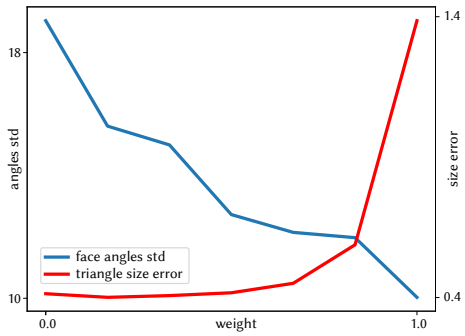

Fig. 8. **Loss blending.** We blend the triangle sizing loss $\mathcal{L}_s$ and the equilateral triangle loss $\mathcal{L}_a$ on 7 models of the dataset. We show the error in triangle distribution and the standard deviation of face angles. Note that given a triangulation, the average angle value is 60°. We observe that we can combine the two losses to obtain a trade off between the desired properties.

Table 2. **Vertex initialization methods.** We compare an initialization based on remeshed 3D vertices to an initialization based on a uniform distribution over the 3D surface. The uniform initialization has significantly higher initial error, but our method can still decrease the loss significantly.

init. method	input mesh	ours
remeshed model vertices	1.354	0.634
uniform	1.429	0.791

triangle size

init. method	input mesh	ours
remeshed model vertices	13.809	9.037
uniform	19.119	11.028

curvature alignment

rejection sampling. We evaluate both initialization methods on the same subset of shapes from Section 4.3. We observe that the alternative initialization method produces initial triangulations with higher errors. While our method is not completely insensitive to the initialization strategy, it can significantly decrease the loss in both cases.

4.5 Runtime and memory

We show the average runtime and maximum memory usage of our method for multiple values of k in Table 3 and multiple vertices count per patch in Table 4. The runtime is for a typical optimization with 1000 iterations. Both time and memory are linear in the number of vertices and cubic in the number of neighbors k. In our experiments, we typically optimize for 1000 steps for the curvature alignment task and 1500 steps for the triangle size task.

Table 3. **Runtime and memory usage w.r.t. k.** We report the average runtime for an optimization with 1K iterations and maximum memory usage per patch.

k	70	80	90
runtime (sec)	132	185	252
memory (GB)	7.6	9.47	13.2

Table 4. **Runtime and memory usage w.r.t. number of vertices per patch.** We report the average runtime for an optimization with 1K iterations and maximum memory usage per patches of varying number of vertices. Time and memory are linear in the number of vertices. Note that we can adjust the size of our patches as needed to avoid memory limitations.

n vertices	500	700	900	1100
runtime (sec)	266	364	477	581
memory (GB)	9.0	12.6	16.3	21.1

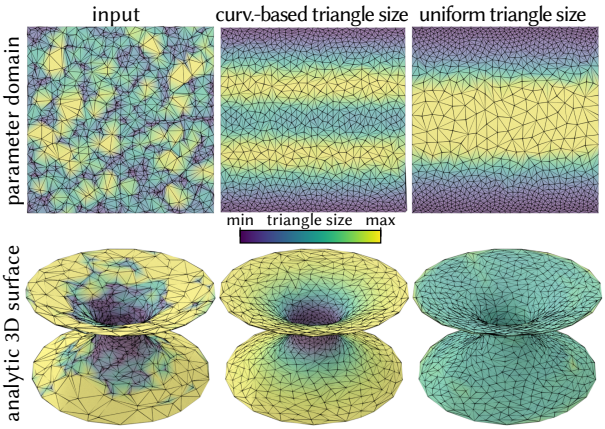

Fig. 9. **Analytic surfaces.** Our method can triangulate surfaces given in any representation: here we triangulate an analytic surface (a catenoid), with the parameteric domain shown on the top row, and the 3D surface on the bottom row. Starting from randomly distributed vertices (left), our approach successfully triangulates the analytic surface with curvature-based triangle sizes (middle) and equal triangle sizes (right).

4.6 Analytic Surfaces

Our differentiable triangulation method can be applied to any kind of 3D surface, as long as bijective piecewise differentiable parameterization of the surface is available. In Figure 9, we experiment with an analytically defined 3D surface, a catenoid [Dierkes et al. 2010]. This surface is defined as a function over a 2D parameter domain (thus, the analytical function itself is our mapping m). We start with randomly distributed vertices in the parameter domain and optimize for either triangle sizes based on the curvature magnitude that we compute analytically or for equal-sized triangles in the 3D domain. Curvature values were computed analytically from the surface definition. We can see that our approach successfully optimizes these objectives on the analytic surface.

4.7 Discussion on performance

We observe that our method presents an overhead compared to task-specific methods in terms of running time and the size of processed meshes. But with this overhead, our method buys generality and differentiability. We can minimize multiple different objectives (like the objectives of [Loseille 2017], [Jakob et al. 2015]), or can easily combine multiple objectives, without modifying our pipeline and our method can be used as a component in a differentiable framework. Our numerical results are on par with specifically tailored remeshing methods. Note that several recent learning-based methods work with even smaller point counts (1024 points in PointNet [Qi et al. 2017], 2250 Edges in MeshCNN [Hanocka et al. 2019], 2048 points in [Luo and Hu 2021]), but these restrictions are quickly decreasing with improvements in GPU hardware and methodological improvements.

5 CONCLUSION, LIMITATIONS & FUTURE WORK

The framework presented in this paper is the first, to the best of our knowledge, to enable approaching surface triangulation from a differentiable point of view. As shown in the experiments, differentiability enables a generic and flexible framework, which can handle various geometric losses, along with their combinations, while taking advantage of modern optimization frameworks. We believe it is the first step towards a black-box, differentiable triangulation module in deep learning frameworks such as PyTorch and TensorFlow where it can be immensely helpful in devising a trainable pipeline, e.g., learning to triangulate models based on deformation sequences.

Our method has two main limitations, the first of which is that the surface needs to be segmented into patches before triangulating. The boundaries of these patches do not participate in the optimization and hence some visible artifacts exist across boundaries. Nevertheless, we note that as shown in the experiments, even with this limitation, our approach achieves significantly better results than the state of the art. A possible solution for this would be to repeat the meshing by iteratively selecting different patches and reparameterizing until convergence.

The second limitation of our method is that it cannot yet handle a large number of points (e.g., 100k+), or large patches, as we need to compute the inclusion scores over a the large space of possible triangles. As future work, we plan to consider a multiscale approach for tackling this issue.

We are excited about the possibilities our approach opens up. For one, since our method can work with surfaces represented in any explicit format (as we show in Figure 9), we wish to explore triangulating surfaces in other representations, such as NURBS, or neural representations such as AtlasNet [Groueix et al. 2018; Morreale et al. 2021]. Extending our method further to point clouds and recovering not only an optimized triangulation but also the topological structure (i.e. the connectivity that defines a surface) could be immensely important for future applications. As an immediate application, we wish to harness differentiabilty to *train* a network to directly output

vertex weights and displacements for any given surface in a single forward pass, and thus avoid test-time optimization.

ACKNOWLEDGMENTS

Parts of this work were supported by the KAUST OSR Award No. CRG-2017-3426, the ERC Starting Grant No. 758800 (EXPROTEA), the ANR AI Chair AIGRETTE, gifts from Adobe, the UCL AI Centre and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant PRIME No. 956585.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. 2008. Recent advances in remeshing of surfaces. *Shape analysis and structuring* (2008), 53–82.

Franz Aurenhammer. 1987. Power diagrams: properties, algorithms and applications. *SIAM J. Comput.* 16, 1 (1987), 78–96.

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, Vol. 36. 301–329.

Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. 2000. Triangulations in CGAL. In *Proc. SoCG*. 11–18.

Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. 2017. Dsac-differentiable ransac for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6684–6692.

Frédéric Cazals and Joachim Giesen. 2004. *Delaunay triangulation based surface reconstruction: ideas and algorithms*. Ph.D. Dissertation. INRIA.

Long Chen and Michael Holst. 2011. Efficient mesh optimization schemes based on optimal Delaunay triangulations. *Computer Methods in Applied Mechanics and Engineering* 200, 9-12 (2011), 967–984.

S.-W Cheng, Tamal Dey, and J.R. Shewchuk. 2016. *Delaunay mesh generation*. 1–386 pages.

Xiao-Xiang Cheng, Xiao-Ming Fu, Chi Zhang, and Shuangming Chai. 2019. Practical error-bounded remeshing by adaptive refinement. *Computers & Graphics* 82 (2019), 163–173.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. Meshlab: an open-source mesh processing tool.. In *Eurographics Italian chapter conference*, Vol. 2008. Salerno, Italy, 129–136.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. 2000. *Computational Geometry: Algorithms and Applications*. Springer-Verlag. 367 pages.

Fernando De Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–11.

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.* 34, 4 (2015), 50–1.

Boris Delaunay et al. 1934. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7, 793-800 (1934), 1–2.

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 317–324.

Olivier Devillers. 2002. The delaunay hierarchy. *International Journal of Foundations of Computer Science* 13, 02 (2002), 163–180.

Olivier Devillers, Sylvain Pion, and Monique Teillaud. 2001. Walking in a triangulation. In *Proc. SoCG*. 106–114.

Ulrich Dierkes, Stefan Hildebrandt, and Friedrich Sauvigny. 2010. Minimal surfaces. In *Minimal Surfaces*. Springer, 53–90.

Qiang Du, Max D Gunzburger, and Lili Ju. 2003. Constrained centroidal Voronoi tessellations for surfaces. *SIAM Journal on Scientific Computing* 24, 5 (2003), 1488–1506.

Marion Dunyach, David Vanderhaeghe, Loïc Barthe, and Mario Botsch. 2013. Adaptive remeshing for real-time mesh deformation. In *Eurographics 2013*. The Eurographics Association.

Ramsay Dyer, Hao Zhang, and Torsten Möller. 2007. Delaunay mesh construction. In *Proceedings of the fifth Eurographics symposium on Geometry processing*. 273–282.

Matthew Fisher, Boris Springborn, Peter Schröder, and Alexander I Bobenko. 2007. An algorithm for the construction of intrinsic Delaunay triangulations with applications to digital geometry processing. *Computing* 81, 2-3 (2007), 199–213.

Xiao-Ming Fu, Yang Liu, John Snyder, and Baining Guo. 2014. Anisotropic Simplicial Meshing Using Local Convex Functions. *ACM Trans. Graph.* 33, 6, Article 182 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661235

Yan Fu and Bingfeng Zhou. 2009. Direct sampling on surfaces for high quality remeshing. *Computer Aided Geometric Design* 26, 6 (2009), 711 – 723. https://doi.org/10.1016/j.cagd.2009.03.007 Solid and Physical Modeling 2008.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 209–216.

David Glickenstein. 2005. Geometric triangulations and discrete Laplacians on manifolds. *arXiv preprint math/0508188* (2005).

Fernando de Goes, Pooran Memari, Patrick Mullen, and Mathieu Desbrun. 2014. Weighted triangulations for geometry processing. *ACM Transactions on Graphics (TOG)* 33, 3 (2014), 1–13.

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 216–224.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 19–26.

Kaimo Hu, Dong-Ming Yan, David Bommes, Pierre Alliez, and Bedrich Benes. 2016. Error-bounded and feature preserving surface remeshing with minimal angle improvement. *IEEE transactions on visualization and computer graphics* 23, 12 (2016), 2560–2573.

Muhammad Hussain. 2009. Efficient simplification methods for generating high quality LODs of 3D meshes. *Journal of Computer Science and Technology* 24, 3 (2009), 604–604.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Transactions on Graphics* 34, 6 (2015), 189.

Dawar Khan, Alexander Plopski, Yuichiro Fujimoto, Masayuki Kanbara, Gul Jabeen, Yongjie Zhang, Xiaopeng Zhang, and Hirokazu Kato. 2020. Surface remeshing: A systematic literature review of methods and research directions. *IEEE Transactions on Visualization and Computer Graphics* (2020).

A. Khatamian and Hamid Arabnia. 2016. Survey on 3D Surface Reconstruction. *Journal of Information Processing Systems* 12 (01 2016), 338–357. https://doi.org/10.3745/JIPS.01.0010

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. 2020. Spectral Mesh Simplification. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 315–324.

Bruno Lévy and Nicolas Bonneel. 2013. Variational anisotropic surface meshing with Voronoi parallel linear enumeration. In *Proceedings of the 21st international meshing roundtable*. Springer, 349–366.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371.

Yiyi Liao, Simon Donne, and Andreas Geiger. 2018. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2916–2925.

Minghua Liu, Xiaoshuai Zhang, and Hao Su. 2020. Meshing Point Clouds with Predicted Intrinsic-Extrinsic Ratio Guidance. In *European Conference on Computer Vision*. Springer, 68–84.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7708–7717.

Adrien Loseille. 2017. Unstructured mesh generation and adaptation. In *Handbook of Numerical Analysis*. Vol. 18. Elsevier, 263–302.

Shitong Luo and Wei Hu. 2021. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2837–2845.

Emilie Marchandise, Jean-François Remacle, and Christophe Geuzaine. 2014. Optimal parametrizations for surface remeshing. *Engineering with Computers* 30, 3 (2014), 383–402.

Riccardo Marin, Marie-Julie Rakotosaona, Simone Melzi, and Maks Ovsjanikov. 2020. Correspondence learning via linearly-invariant embedding. *Advances in Neural Information Processing Systems* 33 (2020).

Pooran Memari, Patrick Mullen, and Mathieu Desbrun. 2011. Parametrization of generalized primal-dual triangulations. In *Proceedings of the 20th International Meshing Roundtable*. Springer, 237–253.

Luca Morreale, Noam Aigerman, Vladimir Kim, and Niloy J. Mitra. 2021. Neural Surface Maps. arXiv:2103.16942 [cs.CV]

Patrick Mullen, Pooran Memari, Fernando de Goes, and Mathieu Desbrun. 2011. HOT: Hodge-optimized triangulations. In *ACM SIGGRAPH 2011 papers*. 1–12.

Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2006. Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. 381–389.

Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. 2002. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 2 (2002), 849–856.

Kok-Why Ng and Zhi-Wen Low. 2014. Simplification of 3d triangular mesh for level of detail computation. In *2014 11th International Conference on Computer Graphics, Imaging and Visualization*. IEEE, 11–16.

Hiromu Ozaki, Fumihito Kyota, and Takashi Kanai. 2015. Out-of-core framework for QEM-based mesh simplification. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization*. 87–96.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

Marie-Julie Rakotosaona, Paul Guerrero, Noam Aigerman, Niloy J Mitra, and Maks Ovsjanikov. 2021. Learning Delaunay Surface Elements for Mesh Reconstruction. *CVPR* (2021).

Nicholas Sharp and Keenan Crane. 2020. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.

Nicholas Sharp and Maks Ovsjanikov. 2020. Pointtrinet: Learned triangulation of 3d point sets. In *European Conference on Computer Vision*. Springer, 762–778.

Vitaly Surazhsky and Craig Gotsman. 2003. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 20–30.

Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 351–358.

Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. 2017. *Handbook of discrete and computational geometry*. CRC press.

Jane Tournois, Pierre Alliez, and Olivier Devillers. 2008. Interleaving Delaunay refinement and optimization for 2D triangle mesh generation. In *Proceedings of the 16th international meshing roundtable*. Springer, 83–101.

Sébastien Valette, Jean Marc Chassery, and Rémy Prost. 2008. Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (2008), 369–381.

Xiaoning Wang, Xiang Ying, Yong-Jin Liu, Shi-Qing Xin, Wenping Wang, Xianfeng Gu, Wolfgang Mueller-Wittig, and Ying He. 2015. Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes. *Computer-Aided Design* 58 (2015), 51–61.

Jin Wei and Yu Lou. 2010. Feature preserving mesh simplification using feature sensitive metric. *Journal of Computer Science and Technology* 25, 3 (2010), 595–605.

Dong-Ming Yan, Guanbo Bao, Xiaopeng Zhang, and Peter Wonka. 2014. Low-resolution remeshing using the localized restricted Voronoi diagram. *IEEE transactions on visualization and computer graphics* 20, 10 (2014), 1418–1427.

Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. 2009. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. In *Computer graphics forum*, Vol. 28. Wiley Online Library, 1445–1454.

Dong-Ming Yan and Peter Wonka. 2015. Non-obtuse remeshing with centroidal Voronoi tessellation. *IEEE transactions on visualization and computer graphics* 22, 9 (2015), 2136–2144.

Weining Yue, Qingwei Guo, Jie Zhang, and Guoping Wang. 2007. 3D triangular mesh optimization in geometry processing for CAD. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. 23–33.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).