Instant Field-Aligned Meshes

Wenzel Jakob[1] Marco Tarini[2,3] Daniele Panozzo[1] Olga Sorkine-Hornung[1]

[1]ETH Zurich [2]CNR-ISTI [3]Università dell'Insubria

Figure 1: *Remeshing a scanned dragon with 13 million vertices into feature-aligned isotropic triangle and quad meshes with ∼80k vertices. From left to right, for both cases: visualizations of the orientation field, position field, and the output mesh (computed in 71.1 and 67.2 seconds, respectively). For the quad case, we optimize for a quad-dominant mesh at quarter resolution and subdivide once to obtain a pure quad mesh.*

Abstract

We present a novel approach to remesh a surface into an isotropic triangular or quad-dominant mesh using a unified local smoothing operator that optimizes both the edge orientations and vertex positions in the output mesh. Our algorithm produces meshes with high isotropy while naturally aligning and snapping edges to sharp features. The method is simple to implement and parallelize, and it can process a variety of input surface representations, such as point clouds, range scans and triangle meshes. Our full pipeline executes instantly (less than a second) on meshes with hundreds of thousands of faces, enabling new types of interactive workflows. Since our algorithm avoids any global optimization, and its key steps scale linearly with input size, we are able to process extremely large meshes and point clouds, with sizes exceeding several hundred million elements. To demonstrate the robustness and effectiveness of our method, we apply it to hundreds of models of varying complexity and provide our cross-platform reference implementation in the supplemental material.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: remeshing, triangulation, quadrangulation, N-RoSy, extrinsic smoothing, point cloud, range scan

1 Introduction

Triangle and quad-dominant meshes are ubiquitously used in computer graphics and CAD applications to represent surfaces, either directly, or as the control grid for higher-order parametric or subdivision surfaces. With the introduction of T-splines [Sederberg et al. 2003] and Dyadic T-Mesh Subdivision [Kovacs et al. 2015], quad-dominant meshes with T-joints (T-meshes) now have similar properties and applications of pure quadrilateral meshes, while being more flexible and naturally supporting the flexible local refinement that is often desired in CAD applications. Meshing surfaces is a challenging problem, and a plethora of methods have been proposed in the past three decades to cope with the increasing quality and scalability requirements of modern applications [Owen 1998; Bommes et al. 2013a]. Semi-regular meshes, which have uniform element shapes and resemble regular triangle or quadrilateral grids, are of particular interest due to their structure, which is ideal for solving PDEs and for defining control grids of higher-order surfaces.

Meshing algorithms can be classified into local and global methods. The former are usually simple, robust and scalable, but due to their locality, they tend to introduce many singularities, i.e., points in the output mesh where the connectivity deviates from that of a regular lattice, and they do not usually support alignment of the output mesh with surface features. Global algorithms solve optimization problems whose size depends on the entire dataset, drastically increasing quality but sacrificing scalability, efficiency and simplicity of implementation. The most popular global approaches seamlessly parametrize the surface, regularly tessellate the parametric space, and then lift it back to 3D.

We propose *instant field-aligned meshing*, a new approach based on a local smoothing operator, which is simple to implement, robust, controllable and efficient. Similarly to field-aligned remeshing methods [Ray et al. 2006; Bommes et al. 2009], our algorithm solves two global problems: estimation of the alignment of the edges of the new mesh and element placement. In contrast to these methods, our algorithm's main steps are local to a vertex and its neighbors, which has important repercussions on its fundamental characteristics: we rely on discontinuous surface fields, whose jumps are resolved on-the-fly by our local operator. This way, we are able to sidestep the

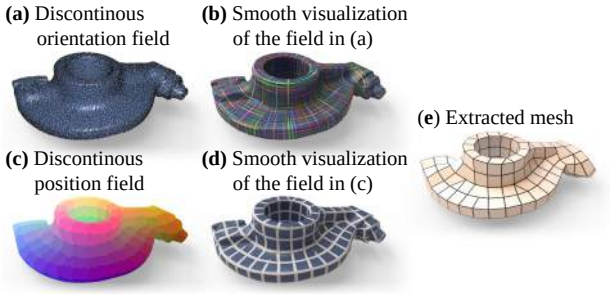


Figure 2: *The types of visualizations in this paper: our method's key steps solve for discontinuous orientation and position fields (left). In the remainder of the paper, we display them using more intuitive smooth visualizations (middle) or extracted meshes (right).*

computation of a global, continuous parametrization. Secondly, our algorithm only requires a loose definition of proximity to define neighborhoods, which allows it to process point clouds and scale to datasets exceeding several hundred million samples by adopting an unstructured multiresolution hierarchy. Our representation is designed to ensure perfect alignment to an orientation field (Figure 2), and its discontinuities are finally removed during the mesh extraction, producing a consistent triangle or quad mesh (Figure 1).

Our algorithm relies on a smoothness energy for tangent fields, which can be chosen to be either *intrinsic* or *extrinsic*. Intrinsic energies, which measure smoothness directly on the surface, have been ubiquitously used by previous works, and we can similarly employ them, producing comparable results. However, our novel extrinsic approach leads to a natural and parameter-free alignment to shape features, which is important especially for mechanical models. It also naturally snaps the edges of the output mesh to sharp features without any parameter tuning.

The running time of our algorithm scales linearly with the size of the mesh: moderately-sized datasets ($<$ 100k triangles) can be meshed in less than a second, and the largest dataset in our experiments with 372 million triangles was processed in less than 10 minutes.

We exploit this efficiency and introduce a set of interactive brush tools that can be used to control the alignment of the edges in the final mesh, their exact position on the surface, and the location and number of singularities. The effect of every stroke is visualized in real time, enabling quick design iterations. With these tools, our system combines automatic and manual meshing methods: it allows users to control the element positions in critical regions, while automatically tessellating the rest of the surface.

To validate our algorithm, we compare with both triangle and quadrilateral meshing methods, demonstrating that our method produces meshes with higher isotropy than the state of the art, while being simple, robust, controllable, fast and scalable to large datasets. The price we pay for these advantages is an increased number of singularities compared to state-of-the-art remeshing algorithms that are based on global parametrization. We provide the full source code and binaries of our interactive reference implementation in the supplemental material, as well as a collection of 232 meshes created with our algorithm and the instructions to reproduce them.

2 Related work

For the sake of brevity, we restrict our survey to the most relevant works in local and global remeshing. We refer an interested reader to [Botsch et al. 2006b] and [Bommes et al. 2013a] for complete surveys of triangle and quadrilateral meshing algorithms.

Surface reconstruction. Surfaces can be acquired using 3D scanning or stereo reconstruction; both techniques extract surface samples in the form of range maps or point clouds. The most common way to convert them into discrete surfaces is to compute a volumetric distance field [Kazhdan et al. 2006] and extract the zero-isosurface triangle mesh using Marching Cubes [Lorensen and Cline 1987]. The quality of the resulting triangulation is typically low, but the algorithm is fast and parallelizable. Our algorithm shares many similarities: it can run on large datasets, it is parallel and robust, but since it operates directly on the data without precomputing an implicit function, it is less resilient to noise. On the other hand, our method can directly produce high quality triangle and quadrilateral meshes. Spectral methods can also be used to quadrangulate point clouds via a point-based Laplacian operator [Huang et al. 2008; Zhang et al. 2010; Ling et al. 2014], though alignment to shape features requires an expensive optimization that does not scale to large input.

Local meshing. Local methods use topological operations, such as edge swap, collapse and split, to improve the quality of an existing mesh. They are widely used for both triangle [Hoppe et al. 1993; Lindstrom and Turk 2000; Surazhsky and Gotsman 2003; Pietroni et al.] and quadrilateral meshes [Daniels et al. 2008; Lai et al. 2008; Tarini et al. 2010]. Other techniques use an advancing front [Sifri et al. 2003] or centroidal Voronoi tessellations or relaxations [Turk 1992; Surazhsky et al. 2003; Alliez et al. 2005; Yan et al. 2009]. These methods struggle with precise alignment of output mesh edges to the surface shape, which can impact the approximation quality. Anisotropic Voronoi tessellations [Lévy and Liu 2010] can address some of these problems but require a costly global optimization that is many orders of magnitude slower than our method. In all of these methods, the position of singular points cannot be explicitly controlled, which can lead to smoothness problems in Catmull-Clark subdivision surfaces [Catmull and Clark 1978]; special methods have been developed to interactively move and collapse singularities [Peng et al. 2011] to deal with this problem. Our approach shares the scalability and robustness of local methods, while also providing edge alignment to shape features as well as control over the placement of singularities.

Sketch-based remeshing. Sketch-based methods enable interactive design of guiding vector fields [Zhang et al. 2006] to adjust both the shape and number of quads used; they are ideal for artists that desire complete control over the generated results [Takayama et al. 2013; Campen and Kobbelt 2014]. Our method combines some of the tools proposed by these works, allowing to restrict their usage to the regions where precise control is important, while relying on our optimization to automatically tessellate the rest of the surface.

Global parametrization. Global methods solve a global optimization that involves the entire dataset, providing a direct way to control the meshing quality and the singularities at the cost of algorithmic and implementation simplicity. They have been introduced e.g. in [Alliez et al. 2002; Gu et al. 2002; Khodakovsky et al. 2003; Marinov and Kobbelt 2006], where the mesh is parametrized to the plane, and a regular tessellation on the plane is lifted to the surface. Precise control of the edge alignment is important for quad meshing and led to the development of parametrization strategies that align with a direction field. Many variants of these methods have been proposed for triangle [Nieser et al. 2012] and quadrilateral meshing [Ray et al. 2006; Kälberer et al. 2007; Bommes et al. 2009; Ebke et al. 2014].

The design of the guiding field is an interesting and difficult subproblem on its own, introduced in [Hertzmann and Zorin 2000] and extensively studied in the past decade [Palacios and Zhang 2007; Ray et al. 2008; Lai et al. 2010; Crane et al. 2010; Knöppel et al. 2013; Panozzo et al. 2014; Diamanti et al. 2014; Jiang et al. 2015]. Global parametrization methods have also been extended to rangemaps [Pietroni et al. 2011] and point clouds [Li et al. 2011].

The majority of the global parametrization methods strives to compute a parametrization whose gradient is aligned with a set of directions that is not integrable, thus requiring complex numerical algorithms to enforce local injectivity and/or low distortion in the parametrization [Bommes et al. 2013b; Myles and Zorin 2013; Levi and Zorin 2014; Myles et al. 2014]. Integrable fields can be designed by minimizing a nonlinear energy [Diamanti et al. 2015], but this approach is not interactive and does not scale to large datasets. In addition, the extraction of a mesh from a global parametrization is a difficult task that requires the handling of many special cases [Ebke et al. 2013]. The major advantage of these methods is that they create pure quadrilateral meshes with precise control over edge alignment and singularity placement, though this comes at a significant cost in high implementation complexity and lack of scalability to large datasets due to the need to compute a globally consistent parametrization over the entire surface. Our algorithm penalizes the distortion more, introducing extra singularities to increase the uniformity of the elements, and to improve their alignment.

A radically different approach was proposed by Ray et al. [2006], who compute a pair of periodic functions without surface cuts, while introducing additional singularities to improve element quality. Our approach is related to this method, since it introduces additional singularities and does not require cutting the surface, but there are four major differences: instead of computing per-triangle parametrizations from smooth global periodic functions, we directly optimize discontinuous surface fields; our method offers precise control over both orientation and position of edges; if no constraints are specified, edges naturally snap to sharp features, and our simple algorithm optimizes the underlying energy in a way that scales linearly with the size of the input.

Integration. Quad-dominant meshes, i.e., meshes made predominantly of quadrilaterals but with a few triangles and pentagons, are commonly used for animation purposes, where they are preferred due to their flexibility in handling density changes [DeRose et al. 1998]. They are easily converted into a pure quadrilateral mesh with one step of Catmull-Clark subdivision [Catmull and Clark 1978].

Quad-dominant meshes can be created by integrating curvature lines [Alliez et al. 2003; Marinov and Kobbelt 2004], and usually contain a large number of singularities. A very interesting special case of quad-dominant meshes are T-meshes, where T-junctions allow a sharp transition in density without breaking the edge flow [Sederberg et al. 2003]. Global parametrization methods have been proposed to automatically create such meshes [Myles et al. 2010], but they share the limitations of global methods discussed earlier. Our algorithm creates quad-dominant meshes, and our singularities achieve a similar effect to T-junctions. Our user interface allows the user to interactively move singularities and collapse them, providing a high degree of control over the final result.

3 Method

Our algorithm combines ideas from local and global meshing methods: we compute a mesh that is globally aligned with a direction field using local orientation- and position-field smoothing operators. The mesh is then extracted from the fields and optionally post-processed.

Orientation field. In the first step (Section 3.2), we compute an N-RoSy field [Ray et al. 2008], i.e., a set of directions on the input surface to which the edges of the output mesh should align. The directions are unique up to a symmetry group that depends on the type of the generated mesh. For example, for quadrilateral meshes, we define a cross at every point of the surface, which is invariant to rotations by 90°. Differently from existing N-RoSy field design methods, which interpolate a (sparse) set of constraints, our approach is completely automatic and parameter-free: by measuring smoothness extrinsically, we obtain natural alignment to surface features. Note that this is different from existing field-aligned methods that align to curvature directions using thresholded hard constraints [Bommes et al. 2009] or global smooth constraints based on an interpolation parameter t [Knöppel et al. 2013]. If desired, such constraints are also supported by our technique, but the main difference is that we achieve parameter-free alignment to features even when no constraints are specified. If our extrinsic metric is replaced with the classically used intrinsic metric, our algorithm becomes an alternative way of minimizing the N-RoSy smoothness energies proposed in [Ray et al. 2008; Bommes et al. 2009].

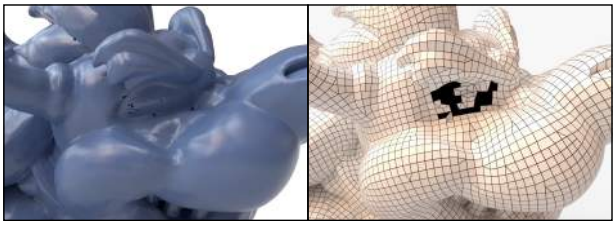


Figure 3: *Our method is robust to low quality or non-manifold input. Left: a challenging input mesh with 545 non-manifold vertices, particularly around the left eye region. Right: our method cannot mesh the eye but degrades gracefully. Global methods that assume manifold input [Ray et al. 2006; Bommes et al. 2009] cannot process such data—the implementations we tried simply crashed.*

Position field. The second step (Section 3.3) computes a local, per-vertex (u, v)-parameterization, which is discontinuous over edges (a direct visualization is shown in Figure 2(c)). We optimize the smoothness of this field under a quotient space of integer translations. The integer coordinate values in this local parameterization correspond to the vertices of the final mesh, and its gradient is exactly aligned with the previously computed orientation field. The field's discontinuous nature averts the need for a costly global optimization to assign globally consistent (u, v) coordinates. This is similar in spirit to the globally smooth periodic functions of Ray et al. [2006]; however, our formulation admits both intrinsic and extrinsic smoothness energies while being considerably faster and simpler to optimize. Similarly to the previous step, we prefer to use an extrinsic approach to smooth the parameterization, leading to natural snapping of the resulting mesh edges to sharp features.

Unified algorithm. The two steps above are deeply related, since they both minimize a smoothness energy defined up to local symmetries: integer rotations in the case of the orientation field and integer translations for the position field. They both employ nonlinear Gauss-Seidel (GS) iterations with a nested brute-force search, which solves the nonlinear symmetry-related part of the energy in a localized fashion. Both steps easily get stuck in bad local minima (Figure 8), which we avoid by either using randomization (Section 3.4) or a multiresolution hierarchy (Section 3.5). Both approaches can work with graphs extracted from a variety of geometric data formats, such as meshes or point clouds with nearest neighbor connectivity. Non-manifold input is handled gracefully: an example is shown in Figure 3, where defects in an otherwise clean mesh cause global methods to fail, while our local method can still produce high quality output away from the defects.

It is possible to detect special points on the surface called *singularities* by analyzing the integer variables defined on edges; this relationship has been studied in detail for the case of orientation fields [Ray et al. 2008]. For position fields, the integer variables induce a different kind of singularity that controls the tessellation density of the output mesh, similarly to [Ray et al. 2006]. In Section 3.6, we discuss this concept and show how position singularities can be moved or even eliminated by collapsing opposite-facing pairs.

3.1 Input and graph representation

We represent the input surface as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each vertex $i \in \mathcal{V}$ is associated with a position $\mathbf{v}_i \in \mathbb{R}^3$ and a normal direction $\mathbf{n}_i$. The set of edges, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, stores the neighborhood relations, which are defined in various ways depending on the input: for point clouds, $(i, j) \in \mathcal{E}$ if point $\mathbf{v}_j$ is in the set of the K-nearest neighbors of point $\mathbf{v}_i$, and for triangle meshes it corresponds to the usual mesh edges. We denote the neighborhood of vertex i by $\mathcal{N}(i) = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$. Every pair of neighboring vertices i, j has an associated weight w_{ij}. The weights can be chosen as uniform ($w_{ij} = 1$) or geometry-dependent [Pinkall and Polthier 1993; Belkin et al. 2009] to better adapt to irregular inputs.

Our iterative algorithm minimizes an energy defined on the vertices of $\mathcal{G}$ using local iterations that resemble the Gauss-Seidel method. The idea can be illustrated by recalling explicit Laplacian smoothing [Taubin 1995], which iteratively smooths the vertex coordinates by local iterations of the form:

$$\mathbf{v}_i \leftarrow \frac{1}{\sum_{j \in \mathcal{N}(i)} w_{ij}} \sum_{j \in \mathcal{N}(i)} w_{ij}\, \mathbf{v}_j. \qquad (1)$$

Our method relies on a similar approach to compute smooth (i.e. as constant as possible) orientation and position fields that are then used to create the final output mesh. The challenge is that both fields are subject to certain symmetry conditions that need to be taken into account in Eq. (1). We now explain both optimization steps in turn.

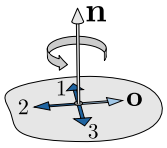
3.2 Orientation field optimization

The first step computes an s_o-RoSy field [Ray et al. 2008], which we call *orientation field*, since it guides the alignment of the edges in the final mesh. An s_o-RoSy field satisfies a *rotational symmetry* condition of degree s_o, which means that every vertex $i \in \mathcal{V}$ is associated with a set of s_o unit-length, evenly spaced tangent vectors. A value of $s_o = 1$ reduces to traditional tangent vector fields, $s_o = 4$ yields a *cross field* and $s_o = 6$ yields a hex-directional field applicable for triangular and hexagonal meshing [Nieser et al. 2012].

Symmetry group. We denote an integer rotation of a representative vector $\mathbf{o}$ about the fixed normal $\mathbf{n}$ as $\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}, k)$, and the set of all integer rotations of $\mathbf{o}$ as $\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n})$:

$$\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}, k) \coloneqq \mathrm{rot}\left(\mathbf{n}, k\,\frac{2\pi}{s_o}\right)\mathbf{o}, \quad k \in \mathbb{Z} \quad \text{and}$$
$$\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}) \coloneqq \{\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}, 0), \ldots, \mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}, s_o - 1)\},$$

where $\mathrm{rot}(\mathbf{n}, \theta)$ is the rotation matrix by angle θ about $\mathbf{n}$. The s_o-RoSy field O consists of $\mathcal{R}_{s_o}(\mathbf{o}_1, \mathbf{n}_1), \ldots, \mathcal{R}_{s_o}(\mathbf{o}_{|\mathcal{V}|}, \mathbf{n}_{|\mathcal{V}|})$, where $\mathbf{o}_i \in \mathbb{R}^3$ is a *representative direction* associated with every vertex $i \in \mathcal{V}$. Note that, being an element of the tangent space of vertex $\mathbf{v}_i$, the representative direction $\mathbf{o}_i$ could also be defined as a two-dimensional quantity in local coordinates. To describe both intrinsic and extrinsic smoothness energies in a unified manner, we prefer to work with a (redundant) representation as vectors in $\mathbb{R}^3$.



$\mathcal{R}_{s_o}(\mathbf{o}, \mathbf{n}, 1 \ldots 3)$

Intrinsic smoothness. We base our construction on the energy used in [Ray et al. 2008; Bommes et al. 2009], which we briefly summarize in the following. Ray et al. [2008] propose to measure the smoothness of an N-RoSy field as the angle difference between adjacent vectors after unfolding them to a common plane. The ambiguity induced by the rotational symmetry is explicitly encoded using integer variables k_{ij}.

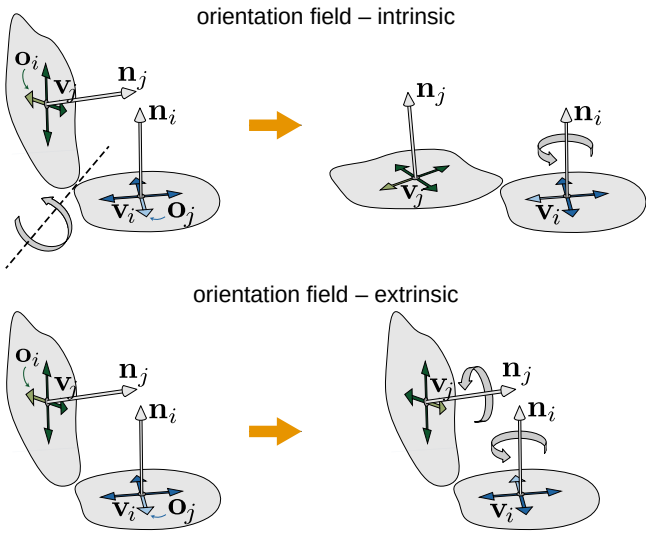


Figure 4: *Measuring smoothness of orientation RoSy fields intrinsically (top) and extrinsically (bottom). Neighboring vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ have tangent planes with normals $\mathbf{n}_i, \mathbf{n}_j$, respectively. To measure field smoothness intrinsically, the tangent plane of vertex $\mathbf{v}_j$ is first transformed by the rotation $\mathrm{rot}(\mathbf{n}_j \times \mathbf{n}_i,\ \measuredangle(\mathbf{n}_j, \mathbf{n}_i))$, so that it becomes parallel to the tangent plane of vertex $\mathbf{v}_i$, and then the difference between the associated RoSy is measured in the common 2D space by finding the integer rotation of the representative vector of j that has the smallest angle with $\mathbf{o}_i$. Extrinsic smoothness measure forgoes the transformation to a common 2D space and measures the difference directly in 3D, finding the angle-minimizing integer rotations for both RoSy representative vectors.*

Let $\mathbf{o}_{ji} \coloneqq \mathrm{rot}(\mathbf{n}_j \times \mathbf{n}_i,\ \measuredangle(\mathbf{n}_j, \mathbf{n}_i))\,\mathbf{o}_j$ be the representative vector at vertex j, rotated into the tangent plane of vertex i. The smoothness is measured as:

$$E(O, \mathbf{k}) \coloneqq \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} \measuredangle(\mathbf{o}_i,\ \mathcal{R}_{s_o}(\mathbf{o}_{ji}, \mathbf{n}_i, k_{ij}))^2, \qquad (2)$$

where k_{ij} are additional integer variables defined for each pair of neighboring vertices $(i, j) \in \mathcal{E}$, and $\mathbf{k} \in \mathbb{R}^{2|\mathcal{E}|}$ is a vector stacking the k_{ij} values. Bommes et al. [2009] proposed to minimize this smoothness energy globally, relying on a combination of Gauss-Seidel iterations, conjugate gradients and sparse Cholesky solvers with a greedy rounding strategy. Knöppel et al. [2013] later proposed a convex N-RoSy smoothness energy, which we discuss in Section 4.

Local iteration. Our approach to minimizing the same energy (Eq. (2)) is much simpler and gives comparable results. We adapt the Gauss-Seidel iteration from Eq. (1):

$$\mathbf{o}_i \leftarrow \sum_{j \in \mathcal{N}(i)} w_{ij}\, \mathcal{R}_{s_o}(\mathbf{o}_{ji}, \mathbf{n}_i, k_{ij}), \quad \mathbf{o}_i \leftarrow \mathbf{o}_i / \|\mathbf{o}_i\|,$$

where we have replaced the division by the sum of weights by a normalization. The most important change is that we search for the optimal integer variables k_{ij} *locally* whenever the vertex i is visited by the iteration. This is a much simpler problem that can be solved by a brute-force search over the (tiny) symmetry space:

$$k_{ij} \coloneqq \operatorname*{arg\,min}_{0 \le k < s_o} \measuredangle(\mathbf{o}_i,\ \mathcal{R}_{s_o}(\mathbf{o}_{ji}, \mathbf{n}_i, k)).$$

In our experiments, we observe considerable improvements in convergence when recomputing the integer variables even more frequently, i.e., after visiting each edge:

$$\begin{aligned} \mathbf{o}_i' &\leftarrow w_{ij_1}\, \mathcal{R}_{s_o}(\mathbf{o}_{j_1 i}, \mathbf{n}_i, k_{ij_1}), & \mathbf{o}_i &\leftarrow \mathbf{o}_i'/\|\mathbf{o}_i'\| \\ \mathbf{o}_i' &\leftarrow \mathbf{o}_i' \;+\; w_{ij_2}\, \mathcal{R}_{s_o}(\mathbf{o}_{j_2 i}, \mathbf{n}_i, k_{ij_2}), & \mathbf{o}_i &\leftarrow \mathbf{o}_i'/\|\mathbf{o}_i'\| \\ &\vdots \end{aligned} \tag{3}$$

where $j_1, j_2, \ldots \in \mathcal{N}(i)$.

The downside of such a purely intrinsic formulation is that it is not aware of the embedding of the surface, which is an important requirement for high-quality remeshing. To fill this gap, Bommes et al. [2009] find the regions where the curvature is high, and then force the field to align to the principal curvature directions in those regions. A curvature threshold needs to be manually tweaked to determine which areas are constrained and which remain free. In [Knöppel et al. 2013], the hard constraints were replaced by soft constraints over the entire surface, which tend to align the field to the estimated curvature directions. The hard threshold is thus replaced by another parameter that controls the weight of the soft constraints.

Extrinsic smoothness. Curvature alignment is often desirable because it tends to lower the approximation error, but it is important to realize that the reverse implication is not generally true: good approximation does not mean that the parameterization must be aligned to curvature directions. We propose a parameter-free alternative to the above process, which entirely sidesteps curvature-related heuristics. Our extrinsic energy directly optimizes for geometric approximation error by computing distances in the embedding space, and we find that this often causes it to align with shape features. It is, however, not forced to do so and can also consider other orientations.

The main difference compared to Eq. (2) is that the neighboring orientation $\mathbf{o}_j$ is never rotated into the tangent plane of $\mathbf{v}_i$; instead, angle differences are computed directly in the 3D ambient space:

$$E_{\mathrm{e}}(O,k) \coloneqq \sum_{i\in\mathcal{V}} \sum_{j\in\mathcal{N}(i)} \measuredangle\left(\mathcal{R}_{s_o}(\mathbf{o}_i, \mathbf{n}_i, k_{ij}), \mathcal{R}_{s_o}(\mathbf{o}_j, \mathbf{n}_j, k_{ji})\right)^2 \tag{4}$$

In this formulation, both field representatives of the adjacent vertices i, j have integer rotations, expressed by k_{ij} and k_{ji}, respectively.

Similarly to the intrinsic case, we minimize this energy by local iterations akin to Eq. (3), except that each time we project the updated vector $\mathbf{o}_i$ onto the tangent plane of $\mathbf{v}_i$ before normalizing it, and, more importantly, the brute-force search now takes place on a symmetry space of s_o^2 pairs of rotations:

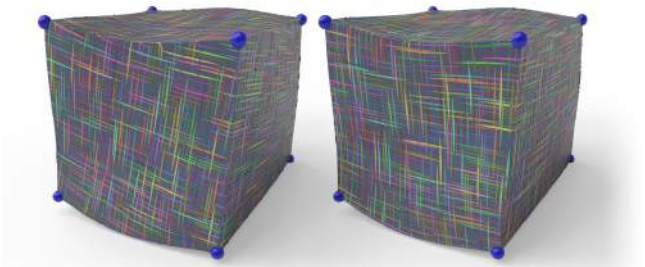
$$(k_{ij}, k_{ji}) \coloneqq \operatorname*{arg\,min}_{0\le k,l<s_o} \measuredangle(\mathcal{R}_{s_o}(\mathbf{o}_i, \mathbf{n}_i, k),\ \mathcal{R}_{s_o}(\mathbf{o}_j, \mathbf{n}_j, l))^2.$$

Refer to Figure 4 for an illustration of the intrinsic and extrinsic optimization procedures.

The fields visualized on the *Twisted Cube* in Figure 5 illustrate this behavior: not a single constraint or parameter was provided to the optimization, yet the algorithm was able to converge to a solution that is perfectly aligned with the cube's corners and curving edges.

3.3 Position field optimization

Given an orientation s_o-RoSy field O (computed with our method or with other field design algorithms), we now want to compute a local parameterization whose gradient is aligned with O. Global parameterization algorithms (e.g. [Bommes et al. 2009]) compute a single, consistent parameterization whose gradient matches the orientation field in a least-squares sense; this procedure requires cutting



(a) Intrinsic smoothness energy **(b)** Our extrinsic smoothness energy

Figure 5: ***(a)*** *Approaches based on intrinsic smoothness energies for N-RoSy fields [Ray et al. 2008; Bommes et al. 2009] by default compute a smooth field that is not aligned to geometric features; they require additional constraints to integrate this "extrinsic knowledge" into the optimization.* ***(b)*** *Our new extrinsic energy achieves the same effect naturally even when no constraints are specified.*

the surface, so that it becomes a topological disk with all the field singularities located on its boundary. This is complex to implement, not scalable to large datasets, since it relies on global mixed-integer optimization, and it is very sensitive to topological noise. On the other hand, such approach provides an ideal framework for semi-regular meshing: a regular grid of triangles or quadrilaterals can be placed in (u, v)-space and lifted onto the mesh using this parameterization. The seamless property of the parameterization ensures that the cuts are not visible. Cuts can be avoided by representing the u and v coordinates using periodic functions [Ray et al. 2006]: a consistent parametrization can then be locally extracted away from singularities and used to regularly remesh the surface. For the purpose of isotropic meshing, this approach has the major advantage of being able to introduce additional singularities that keep the area distortion low. Similarly to the previous case, an involved nonlinear optimization has to be solved over the entire mesh, preventing its use on large datasets.

Both approaches above rely on an optimization that computes a pair of parametrization functions whose gradients are similar, in a least-squares sense, to the given orientation field. We instead tackle the problem in a purely local way: instead of finding continuous parametrization functions over the entire mesh, we compute a local parameterization for each vertex and define a smoothness energy that, similarly to the orientation field optimization, measures the similarity between two local parameterizations up to integer translations. By construction, the parametrizations will be exactly aligned with the orientation field and thus discontinuous on edges. For visualization purposes only, we interpolate the discontinuous functions in a shader, which we use in all our renderings (Figure 2).

Symmetry group: Let us denote by ρ a user-defined, global scaling factor that specifies the desired edge length in the output mesh. We use an unusual, but simple and efficient encoding for the local parameterization of a vertex i: we fix the parameterization's gradient to be precisely aligned with the orientation field $\mathbf{o}_i$, strongly reducing the remaining degrees of freedom. The local parameterization of vertex i is encoded as a 2D point in its tangent plane that refers to the closest lattice point; this uniquely defines a parameterization over the entire tangent plane, up to translations by integer multiples of ρ (Figure 6).

When these local parameterizations are defined over the entire surface, we have a 2D point value for each vertex, which is defined up to a *positional symmetry* (PoSy) condition. We call these 2D valued fields *position fields*. Intuitively, they capture the fractional coordinates of the parametric position of each vertex. To be precise, they span a space that is locally invariant under translations by integer multiples of ρ along s_p symmetry axes and rotations by

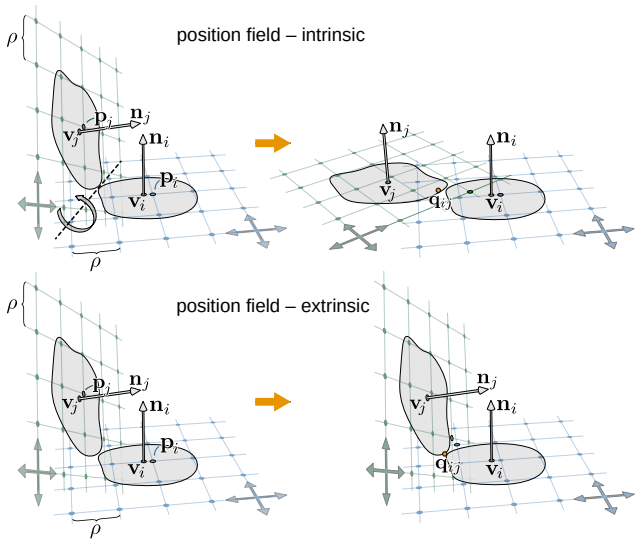


Figure 6: *Illustration of the two position field smoothness energies: in the intrinsic case, all quantities associated with $\mathbf{v}_j$ are rotated into the tangent plane of $\mathbf{v}_i$; afterwards, the closest representative position to $\mathbf{p}_i$ is determined. The rotation is omitted in the extrinsic case, and both positions are translated. The final representative positions are drawn in a darker color.*

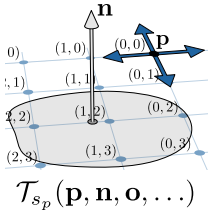
integer multiples of $2\pi/s_p$, where $s_p = m\, s_o,\ m \in \mathbb{N}$. As before, we prefer to work with three-dimensional quantities for a unified discussion of the intrinsic and extrinsic case. For this, we identify each tangential 2D position field value with a corresponding position in the 3D tangent plane, i.e., $(u_i, v_i) \mapsto \mathbf{v}_i + u_i\mathbf{a}_i + v_i\mathbf{b}_i$, where $(\mathbf{a}_i, \mathbf{b}_i)$ is a basis of the tangent space at $\mathbf{v}_i$. In Figure 2 (c), we visualize a raw field by mapping the 3D positions to RGB colors.

We define an integer translation of a position $\mathbf{p} \in \mathbb{R}^3$ and the set of all its possible integer translations as

$$\mathcal{T}_{s_p}(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}) \coloneqq \mathbf{p} + \rho \sum_{k=0}^{s_p/2-1} t_k\, \mathcal{R}_{s_p}(\mathbf{o}, \mathbf{n}, k) \quad \text{and}$$

$$\mathcal{T}_{s_p}(\mathbf{p}, \mathbf{n}, \mathbf{o}) \coloneqq \left\{ \mathcal{T}_{s_p}(\mathbf{p}, \mathbf{n}, \mathbf{o}, \mathbf{t}) \;\middle|\; \mathbf{t} \in \mathbb{Z}^{s_p/2} \right\},$$

respectively. The position field P is then given by the set of tangential lattices of the vertices, i.e. $\mathcal{T}_{s_p}(\mathbf{p}_1, \mathbf{n}_1, \mathbf{o}_1), \ldots, \mathcal{T}_{s_p}(\mathbf{p}_{|\mathcal{V}|}, \mathbf{n}_{|\mathcal{V}|}, \mathbf{o}_{|\mathcal{V}|})$, where $\mathbf{p}_i \in \mathbb{R}^3$ is a *representative position* associated with vertex i. We assume that s_p is even; the sum over $s_p/2$ terms above ensures that each translation axis is considered once (but in both directions). Similar to the orientation field optimization, we now define a smoothness energy that is invariant to the integer translation symmetry condition.

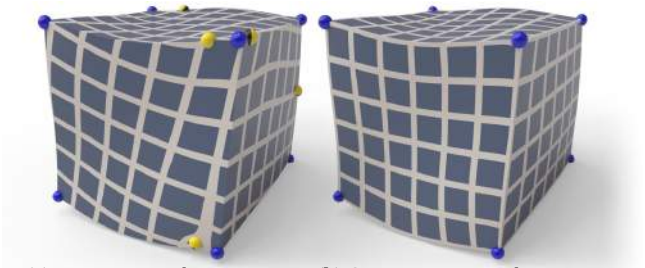


We define $\mathbf{p}_{ji}$ as the representative position of vertex j after a rotation into the tangent plane of $\mathbf{v}_i$:

$$\mathbf{p}_{ji} \coloneqq \mathrm{rot}\left(\mathbf{n}_j \times \mathbf{n}_i,\ \measuredangle(\mathbf{n}_j, \mathbf{n}_i)\right)\,(\mathbf{p}_j - \mathbf{q}_{ij}) + \mathbf{q}_{ij},$$

where $\mathbf{q}_{ij}$ is a point that lies on the intersection of the two tangent planes (its computation is discussed in the appendix). We can then measure the intrinsic smoothness with the following energy:

$$E(P, \mathbf{t}) \coloneqq \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} \|\mathbf{p}_i - \mathcal{T}_{s_p}(\mathbf{p}_{ji}, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij})\|_2^2, \qquad (5)$$



(a) Intrinsic smoothness energy **(b)** Our extrinsic smoothness energy

Figure 7: *Our extrinsic energy naturally* snaps *the integer isolines to sharp features (b), while the intrinsic version is unaware of them.*

which penalizes disagreements in the representative parametric positions, where the integer variables $\mathbf{t}_{ij} \in \mathbb{Z}^{s_p/2}$, defined per edge $(i, j) \in \mathcal{E}$, remove ambiguities due to the underlying symmetry.

By definition, this energy is *intrinsic* and thus unaware of the geometric embedding. Hence, the integer isolines of the resulting parameterization are unlikely to snap to the shape's features (see Figure 7(a)). To remedy this, one could first locate the features and then explicitly add constraints that force the isolines to pass there [Bommes et al. 2009]. Instead, similarly to our solution for orientation fields, we modify our energy to make it *extrinsic*, obtaining natural snapping of the integer lines to geometric features:

$$E_{\mathrm{e}}(P, \mathbf{t}) \coloneqq \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} \left\|\mathcal{T}_{s_p}(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij}) - \mathcal{T}_{s_p}(\mathbf{p}_j, \mathbf{n}_j, \mathbf{o}_j, \mathbf{t}_{ji})\right\|_2^2, \qquad (6)$$

where $\mathbf{t}_{ij}, \mathbf{t}_{ji} \in \mathbb{Z}^{s_p/2}$ are $s_p/2$-dimensional integer variables. The bottom part of Figure 6 illustrates this change. Note how the lowest energy configuration is reached when both vertices lie exactly on the intersection between the tangent planes. This subtle change has a major impact on the results, as demonstrated in Figure 7.

Our minimization strategy is similar to the algorithm used for orientation fields: we iteratively visit each edge, finding the integer variables locally via brute-force search each time:

$$\begin{aligned}
&\mathbf{p}_i' \leftarrow w_{ij_1}\, \mathcal{T}_{s_p}(\mathbf{p}_{j_1 i}, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij_1}), && \mathbf{p}_i \leftarrow \mathbf{p}_i' / \textstyle\sum_{k=1}^{1} w_{ij_k} \\
&\mathbf{p}_i' \leftarrow \mathbf{p}_i' + w_{ij_2}\, \mathcal{T}_{s_p}(\mathbf{p}_{j_2 i}, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}_{ij_2}), && \mathbf{p}_i \leftarrow \mathbf{p}_i' / \textstyle\sum_{k=1}^{2} w_{ij_k} \\
&\quad\vdots && \quad\vdots \\
&\mathbf{p}_i \leftarrow \mathbf{round}(\mathbf{n}_i, \mathbf{o}_i, \mathbf{p}_i, \mathbf{v}_i) && (7)
\end{aligned}$$

The final lattice rounding operation performs an integer translation to determine the position representative that is the closest to the vertex position $\mathbf{v}_i$ (a definition is provided in the appendix).

The exhaustive search for the optimal position integer variables $\mathbf{t}_{ij}$ may initially appear daunting, since the space of possible integer translations is infinitely large. We would need to search two regular lattices, living in different tangent spaces, for the two points with the smallest pairwise distance. Formally, this problem is equivalent to the *shortest vector problem* (SVP) in an s_p-dimensional lattice, which does not have an efficient solution [Micciancio 2001].

However, our problem is more specific than the SVP, because we are only interested in points that are also close to $\mathbf{v}_i$ and $\mathbf{v}_j$ and their corresponding tangent planes, which permits a very localized search. In the appendix, we describe a simple computation of an intermediate position $\mathbf{q}_{ij}$ that minimizes the distance to $\mathbf{v}_i$ and $\mathbf{v}_j$ subject to being constrained to lie in both tangent planes (see

Figure 8: *On its own, the basic nonlinear Gauss-Seidel iteration introduced in Section 3.2 can locally smooth the orientation and position fields, though it soon gets stuck in local minima that are far from the desired solution. By combining the iteration with a simple multiresolution hierarchy, this problem is avoided.*

Figure 6, where $\mathbf{q}_{ij}$ is highlighted in orange). With this position in hand, we again search for the best pair of integer jumps

$$(\mathbf{t}_{ij}, \mathbf{t}_{ji}) \coloneqq \underset{\mathbf{t}\in Q_{ij},\ \mathbf{m}\in Q_{ji}}{\arg\min} \|\mathcal{T}_{s_p}(\mathbf{p}_i, \mathbf{n}_i, \mathbf{o}_i, \mathbf{t}) - \mathcal{T}_{s_p}(\mathbf{p}_j, \mathbf{n}_j, \mathbf{o}_j, \mathbf{m})\|,$$

where the sets Q_{ij} and Q_{ji} only contain translations that yield positions near $\mathbf{q}_{ij}$. In our experiments, we found it sufficient to only consider the s_p immediate neighbors of $\mathbf{q}_{ij}$ on each lattice, and thus we search over a total of s_p^2 possible combinations.

3.4 Randomized GS

As noted before, the GS iterations tend to get stuck in local minima. An extremely simple approach to avoid this is to traverse the graph $\mathcal{G}$ using a different random permutation of the vertices in each iteration. While this does not address any of the other issues of the GS method with regards to slow convergence, it does lead to a particularly short and instructive algorithm. We provide a script in the supplemental material, which implements the field smoothing steps in 100 lines of Python code. However, all of the results shown in this paper are generated using the approach described in the next section.

3.5 Multiresolution hierarchy

We use a simple multiresolution hierarchy to improve convergence and to allow the algorithm to move out of local minima. Starting with the full-resolution input graph $\mathcal{G}$, we perform approximately $\log_2 |\mathcal{V}|$ coarsening steps to create the hierarchy by collapsing adjacent vertices until the entire graph is reduced to a single "super-vertex" per connected component.

Traditional multi-grid approaches on unstructured triangle mesh use mesh decimation [Aksoylu et al. 2005] to build the hierarchy levels. However, similarly to [Botsch et al. 2006a], we do not require the levels of the hierarchy to be a consistent triangulation, and we can thus use a much simpler clustering strategy. In particular, we use the following scheme:

1. Preprocessing: assign a dual area A_i to each vertex i (uniform $A_i = 1$, or Voronoi area when the input is a mesh).
2. Repeat the following phases until a fixed point is reached:
   (a) For each pair of neighboring vertices i, j, assign a score
   $$S_{ij} \coloneqq \langle \mathbf{n}_i, \mathbf{n}_j \rangle \min\left(A_i/A_j,\ A_j/A_i\right).$$
   (b) Traverse S_{ij} in decreasing order and collapse vertices i and j into a new vertex v if neither has been involved in a collapse operation thus far. The new vertex is assigned an area of $A_v = A_i + A_j$, and its other properties are given by area-weighted averages of the vertices that are merged together.

Each iteration reduces the number of vertices approximately by half. Figure 9 shows an illustration of the progressively coarser color-coded hierarchy levels in the *Botijo* dataset. We keep track of the

Figure 9: *Our multiresolution hierarchy is created by merging adjacent vertices in the input graph in approximately* $\log_2 |\mathcal{V}|$ *phases until only a single "super-vertex" is left per connected component.*

relations between hierarchy levels to be able to propagate position and orientation fields up and down between different resolutions. Figure 8 shows the impact of the hierarchy on convergence. In all our results, including the right side of Figure 8, we run six GS iterations per level, beginning at the coarsest version of the graph and finishing at the original input mesh. The initial orientation and position fields are initialized using uniformly distributed random tangential vectors and positions within the bounding box of the mesh. After completing each level, we simply copy the solution to the next finer level. The inset in Figure 20 shows the decreasing smoothness energy during this process.

We experimented with different methods for assigning normals to vertices in the finest resolution graph: averaging the angle-weighted normals of adjacent faces produces generally good results but over-smoothes normals in crease regions, which reduces alignment to hard features (e.g. in CAD models). We flag crease edges based a dihedral angle threshold. Any vertex that is adjacent to a crease edge is (arbitrarily) assigned the normal of an adjacent face rather than their average.

Intuitively, the space of possible solutions at the finest level is exceedingly large, and our energy functions E are non-convex and contain many local minima. On the other hand, a coarse graph with just a few vertices has a low-dimensional solution space whose global minimum is easier to find for a local, iterative method. Our algorithm is therefore based on the hypothesis that the global minima of two adjacent hierarchy levels are closely related, so that a converged solution at one level is an excellent starting point for optimization of a finer graph after one level of refinement. We currently offer no formal proof or guarantees of this property and demonstrate the success of our method in practical experiments.

3.6 Singularities

The integer variables k_{ij} and $\mathbf{t}_{ij}$ are a byproduct of our local iterations. They represent the transformations required to locally factor out the symmetries of the two fields: integer rotations for orientation fields and integer translations for position fields. Consider any simple cycle around a vertex in $\mathcal{G}$: usually, the composition of all transformations over the cycle is an identity transformation. If this is

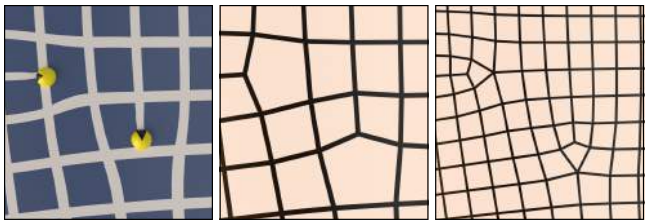


Figure 10: *Position singularities such as the yellow markers (left) naturally arise in our position field optimization; they allow our method to adapt to the available space and create T-junctions in the output mesh to maximize isotropy. The singularities create pentagons or triangles in the output mesh (middle), which can be converted into pairs of valence 3 and 5 vertices after a subdivision step (right).*

not the case, the cycle contains a *singularity* of the respective field, which introduces an irregularity in the output mesh.

Orientation field singularities. In an orientation field, the definition of a singularity is equivalent (up to being defined on a general graph) to the familiar notion of singularity of discrete N-RoSy fields on piecewise linear meshes [Ray et al. 2008]. An N-RoSy singularity corresponds to an irregular vertex in the final meshing, i.e. a vertex with more or less than four (in quad meshes) or six (in triangle meshes) incident edges.

Position field singularities. In a position field, a singularity corresponds to the residual integer translation over a cycle, and it is defined only if no orientation singularity is present. Position field singularities cannot affect the orientation of the parametrization, which is entirely controlled by the orientation field. Therefore, they intuitively prescribe insertions or removals of a parametric isoline in the region inside the cycle. In analogy to the orientation singularities, a position singularity locally disrupts the regularity of the output mesh (Figure 10): in triangle meshes, it simply corresponds to an irregular vertex, and in quad meshes, it can be equivalently meshed as a T-junction or a polygonal element (a triangle or a pentagon). A position singularity splits or merges two parallel edge loops (i.e. sequences of consecutive edges that make no turn), preserving their alignment. These singularities are used by our algorithm to enforce the uniformity of elements without deviating from the prescribed orientation field, in contrast to global parametrization methods, which avoid these singularities by sacrificing isotropy and field alignment. It is not surprising that the number of position singularities depends on the resolution of the output mesh.

As an optional post-process step, we convert our quad-dominant meshes into pure quad meshes with a step of Catmull-Clark subdivision; each position singularity becomes a pair of irregular vertices of valence 3 and 5 in the final mesh (Figure 10).

Detecting singularities. When the input graph $\mathcal{G}$ is a triangle mesh, it is simple to detect orientation and position singularities using the already computed integer variables by adding k_{ij} or $\mathbf{t}_{ij}$ over the edges of each triangle and checking whether the result is a multiple of s_o or non-zero, respectively. We use this simple definition in the next section to implement editing operations that move and collapse singularities. Detecting and manipulating singularities for point clouds is more involved; we leave this as a future work.

3.7 Editing operations

Our method supports intuitive editing operations to influence orientation and position values via constraints that are passed to the corresponding optimization stages. Another type of editing operation makes targeted modifications to the integer variables k_{ij} and $\mathbf{t}_{ij}$ to move and collapse singularities. The supplemental video contains demonstrations of all tools in action.

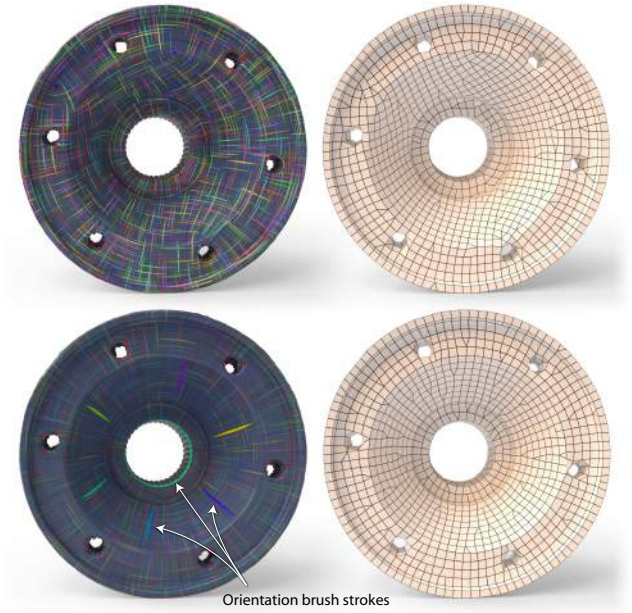


Figure 11: *The orientation brush forces the orientation field to align with the direction of a stroke drawn by the user. This is an intuitive way of influencing the edge flow and placement of singularities.*

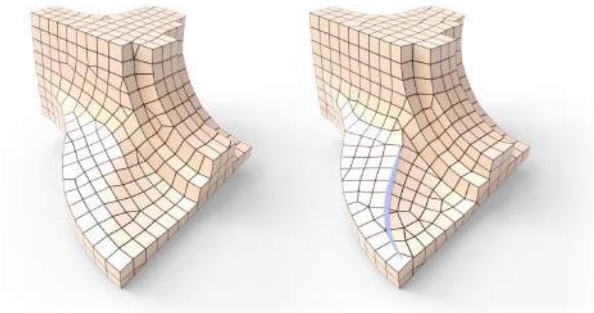


Figure 12: *The edge brush forces an iso-parameter line of the position field to pass along a stroke drawn by the user. This causes a corresponding chain of edges to be created in the output mesh.*

Orientation and position constraints. The orientation (Figure 11) and position brushes (Figure 12) enforce field constraints during the GS iteration: when computing an orientation field value at vertex i, $\mathbf{o}_i$ is updated to a value that is linearly interpolated between the originally computed result and a constraint, and then projected onto the tangent space and normalized; for this, we search the symmetry space once more and interpolate the most similar directions. The interpolation weight determines the strength of the constraint. Constraints are also propagated up in the multiresolution hierarchy and interpolated whenever two vertices with constraints are merged.

The position brush works similarly: we constrain one of two possible dimensions by projecting the computed $\mathbf{p}_i$ onto the plane spanned by the stroke direction and the normal, and adjust it to an interpolated value between its current position and the projection. When processing meshes with boundaries, our implementation can automatically assign matching edge brush constraints so that boundary edges in the output mesh follow the input.

Singularity attractor. The singularity attractor (Figure 13) enables interactive movement of both orientation and position singularities. We currently only provide this operation for triangle mesh

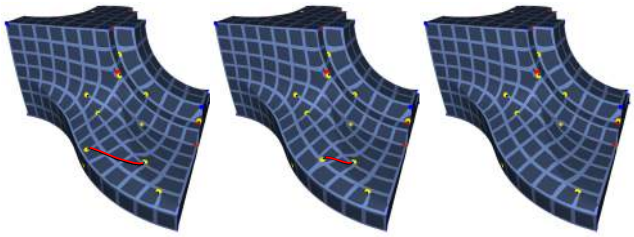


Figure 13: *Example application of the singularity attractor to move and collapse a pair of position singularities of opposite sign.*

inputs, as it requires a decomposition of the input graph into simple cycles. When using this operation, the discrete search over symmetry spaces is deactivated and the integer variables are globally frozen to their last computed value. The optimization continues, but only the fractional part is optimized from then on. Since no other brush strokes can be applied afterwards, this step should come last in any meshing workflow.

Recall that we can detect the presence of singularities simply by adding the associated integer values over the edges of a triangle in the input mesh. The operation implemented by the attractor then moves a singularity of either type from a triangle T_1 to an adjacent triangle T_2 by a suitable modification to the rotation index or integer shifts associated with their shared edge; longer movements are done by repeating this operation. Between each step, the GS iteration is executed at the finest hierarchy level to update the continuous part of the solution, and the singularity moves accordingly. We observed that stretching the computed field in this way can occasionally introduce additional singularities.

3.8 Mesh extraction

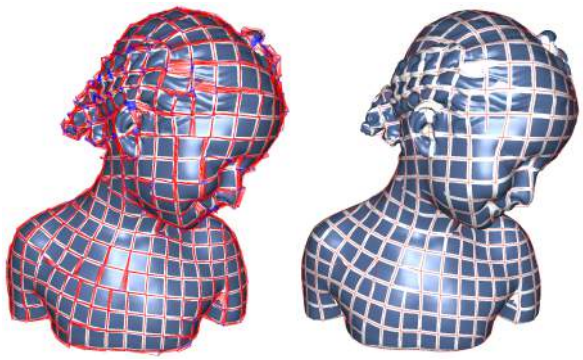
Existing mesh extraction algorithms [Ebke et al. 2013] are specialized to surfaces and difficult to extend to our general graph representation. In this section, we propose a simple and efficient way of turning the computed fields into a mesh, which works for 4- and 6-PoSy fields and can handle point clouds. We assume that the input mesh resolution is denser than the desired output resolution (otherwise, we appropriately subdivide the input where necessary).

Recall that each vertex i in the input graph is associated with a three-dimensional representative position $\mathbf{p}_i$. Due to the last rounding step in the position optimization (Eq. (7)), this is the closest representative position to the vertex $\mathbf{v}_i$. This 3D point can also be understood as the approximate position of a vertex to be placed into the output mesh. Every edge $(i,j) \in \mathcal{E}$ of the input mesh is associated with an $s_p/2$-dimensional vector of integer values $\mathbf{t}_{ij}$, which expresses how many integer translations are necessary to bring vertices $\mathbf{p}_i$ and $\mathbf{p}_j$ as close together as possible.

Many of these integer values $\mathbf{t}_{ij}$ are zero in the intrinsic case, or opposite in sign, i.e. $\mathbf{t}_{ij} = -\mathbf{t}_{ji}$, in the extrinsic case, which means that both vertices reference the same position up to small differences in the fractional part. Among the nonzero integer variables, unit vector-valued $\mathbf{t}_{ij}$ are also of particular interest: they state that the representative positions $\mathbf{p}_i$ and $\mathbf{p}_j$ differ by a single integer translation, i.e. they approximate an edge of the output mesh.

Figure 14 shows a visualization of the information conveyed by the integer variables. Each variable with $\mathbf{t}_{ij} = -\mathbf{t}_{ji}$ is drawn as a blue edge between the associated representative positions $\mathbf{p}_i$ and $\mathbf{p}_j$, whereas unit vector-valued $\mathbf{t}_{ij}$ are shown as red edges. This visualization closely resembles the desired output mesh, where vertices appear as small blue clusters connected by groups of red edges.

To extract the mesh, we begin by creating an undirected graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, where the set of vertices $\mathcal{V}'$ is the set of representative



(a) Input graph **(b)** Processed graph

Figure 14: *The position field and integer jumps generated by the second stage of our optimization pipeline are already in a form that is very close to the final output mesh:* ***(a)*** *shows a visualization of the position field, where zero integer jumps are connected by a blue line, and red lines indicate unit jumps. We collapse clusters of blue links into the vertices of the output mesh, shown in* ***(b)****, and extract polygons from the resulting graph.*

positions $\mathbf{p}_1, \ldots, \mathbf{p}_{|\mathcal{V}|}$ of the input graph, and two vertices (i,j) are connected if the associated integer variable is an integer unit vector (this corresponds to the red subgraph in Figure 14(a)). Let $\mathcal{C}$ denote the set of all edges (i,j) with $\mathbf{t}_{ij} = -\mathbf{t}_{ji}$ (these are the blue edges). Our algorithm then takes edges from $\mathcal{C}$ and collapses the associated vertices in $\mathcal{G}'$ until $\mathcal{C}$ is empty.

Even such a simple criterion can already produce a passable output graph, though it collapses vertices too eagerly near orientation and position singularities due to the conflicting information recorded in the integer variables at such positions. We avoid this behavior by visiting the edges $(i,j) \in \mathcal{C}$ in increasing order of distance between their representative positions $\|\mathbf{p}_i - \mathbf{p}_j\|$ and only collapsing the associated vertices in $\mathcal{G}'$ if they are not already connected by an edge in $\mathcal{G}'$: an already existing edge indicates a conflict among integer variables regarding whether or not the vertices should be merged, and we conservatively choose to keep them. The intuition behind this simple heuristic is that short edges represent a high degree of confidence that two vertices in $\mathcal{G}'$ in fact correspond to the same vertex of the output mesh. By visiting them in this order, we can postpone conflicting merge operations until the very end, at which point they can easily be identified as such and ignored.

Following the collapse operations, each remaining vertex in $\mathcal{G}'$ is the result of merging multiple representative positions from the input graph. To keep notation simple, let us focus on one vertex $\mathbf{v}'_k \in \mathcal{V}'$ which corresponds to a cluster of representative positions $\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_n} \in \mathcal{V}$. We assign the output vertex a position value and a normal direction as a weighted combination of the representative positions and normals of the input mesh, respectively:

$$(\mathbf{v}'_k, \mathbf{n}'_k) \leftarrow \frac{1}{\sum_l^n \alpha_{i_l}} \sum_{l=1}^{n} \alpha_{i_l} (\mathbf{p}_{i_l}, \mathbf{n}_{i_l}),$$

where $\alpha_i := \exp\left(-\beta^{-2}\|\mathbf{p}_i - \mathbf{v}_i\|^2\right)$ and $\beta = \rho/3$ in our experiments. The rationale of this expression is to give more weight to vertices in the input mesh that are close to their associated representative position. Occasionally, the graph extraction step produces a merged vertex with very low n, i.e., few corresponding vertices in the input mesh, which leads to spurious faces in the output. We remove them by discarding vertices with $n < \bar{n}/10$, where $\bar{n}$ is the average value of n over all vertices in $\mathcal{G}'$.

The processed graph $\mathcal{G}'$ describes both position and adjacency information of the output mesh. A final step entails detecting the actual

Figure 15: *Screen captures of a point cloud meshing session conducted using our interactive implementation*

faces, which can be implemented using a simple greedy algorithm that walks along oriented edges and removes them from the graph. This can sometimes produce non-manifold output in tricky situations such as extremely coarse meshing of very detailed or high-genus input. Note that in such situations, existing global methods might also introduce folds that either cause severe distortion or holes. If manifold output is required, we simply drop offending edges or vertices, producing similar artifacts.

4 Results

We implemented our algorithm in C++, relying on the language's template mechanism to easily instantiate our optimization kernels for different symmetry groups. We use the Eigen library for linear algebra operations, and Intel Threading Building Blocks to facilitate parallelization of all components. We follow a standard approach to parallelize the nonlinear Gauss-Seidel iterations to achieve interactive performance even on larger models: a greedy graph coloring preprocess produces a graph labeling using approximately 5-8 colors. Vertices of the same color share no edges and can be safely visited in a sequence of (independent) parallel phases. We also experimented with Jacobi-style iterations that require no such provisions, but preferred the better convergence of our modified Gauss-Seidel method. Our algorithm requires no special numerical precautions and works well in single precision arithmetic.

The extraction phase is also executed in parallel; to efficiently merge clusters of vertices of the input mesh into the vertices of the output mesh, worker threads pop edges (i, j) to be merged from a parallel priority queue. The merging operation itself is implemented using a lock-free disjoint set data structure that supports concurrent operations, implemented using atomic compare and exchange operations [Anderson and Woll 1991].

We developed an interactive proof-of-concept meshing application that is demonstrated in the accompanying video (see also Figure 15). As the user interacts with our tool, a preview of the computed orientation and position fields is rendered using OpenGL. The orientation field visualization is generated by tracing flow lines on the mesh using a simple explicit integrator. The position field visualization is implemented as a pixel shader that maps fractional field values into a procedural texture of a regular lattice. To render point clouds, we simply draw circular disks with a constant value of $\mathbf{o}_i$ and $\mathbf{p}_i$ per element. For triangles, we interpolate the field values over triangles to provide a more pleasing visualization that is continuous almost everywhere (exceptions are edges of triangles containing singularities). To interpolate field values, we must once more remove ambiguities due to the underlying symmetry groups. This is done in a geometry shader, which implements the search over the associated quotient spaces and generates new triangles whose parameters can be interpolated linearly by the GPU rasterizer.

We provide the full source code of our method and cross-platform binaries in the supplemental material. Additionally, we include all mesh files that are needed to reproduce our comparisons, as well as instructions for reproducing them with our binary. Experiments were conducted on a 12-Core 2.7 GHz Intel Xeon E5 workstation with 64 GB of memory, except for the large *St. Matthew* mesh, where we used a 16-core Intel Xeon machine with 256 GB memory. The statistics for all of our comparisons are in Table 1. In these comparisons, we always used extrinsic energies and we only control the density of the output to match the results of the other algorithm, or to achieve a sufficiently high resolution for our method. We subdivide all quad-dominant meshes with one step of Catmull-Clark subdivision to make the results comparable between methods.

Triangle meshing. We compare our triangle meshing results obtained with a 6-RoSy [Nieser et al. 2012] and 6-PoSy with [Surazhsky et al. 2003] and [Pietroni et al.] in Figure 16. The former uses local operations to obtain a uniform distribution of the samples; our method outperforms it both in terms of isotropy and regularity, using 5k singularities instead of 20k (Figure 16, top). [Pietroni et al.] uses a global parametrization constructed with an iterative simplification method and produces a more regular mesh than ours (Figure 16, bottom), but with a much higher angular distortion (standard deviation of 17° compared with 5° for us).

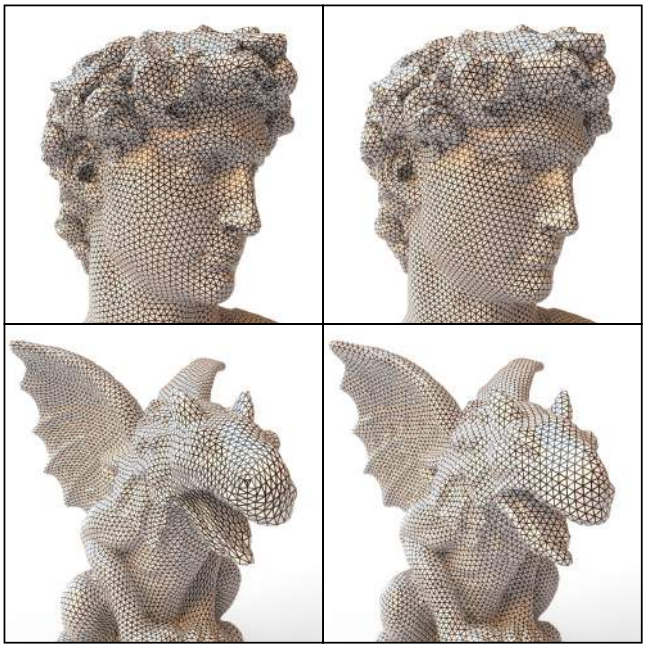


Figure 16: *Triangle meshing comparison: [Surazhsky et al. 2003] (top left), [Pietroni et al.] (bottom left) and our results on the right. Note that our meshes are better aligned to the shape features.*

Quad-dominant meshing. By combining a 4-RoSy with a 4-PoSy, our algorithm produces quad-dominant meshes with pentagons and triangles in the vicinity of position singularities. We observe that our method achieves a much higher regularity (148 singularities vs. 436) compared to [Alliez et al. 2003] (Figure 17). Our elements are also more uniform, with a normalized area standard deviation of 0.17 instead of 0.62 (measured on the *Pig* dataset). PGP [Ray et al. 2006] generates meshes with good alignment and similar numbers of singularities, but introduces noticeable errors because vertices of the output mesh do not snap to geometric features. Our extrinsic position smoothing considerably improves on this.

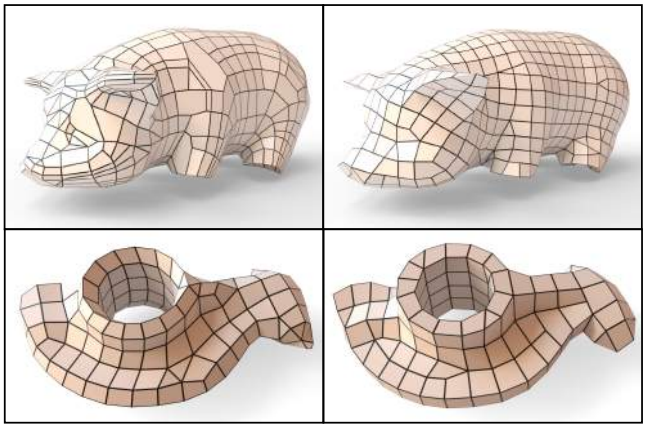


Figure 17: *Quad-dominant meshing comparison: Top left: [Alliez et al. 2003] and our result (right), which is more uniform and contains fewer irregular vertices. Bottom: [Ray et al. 2006] (left) and our result, which snaps to geometric features of the input mesh.*

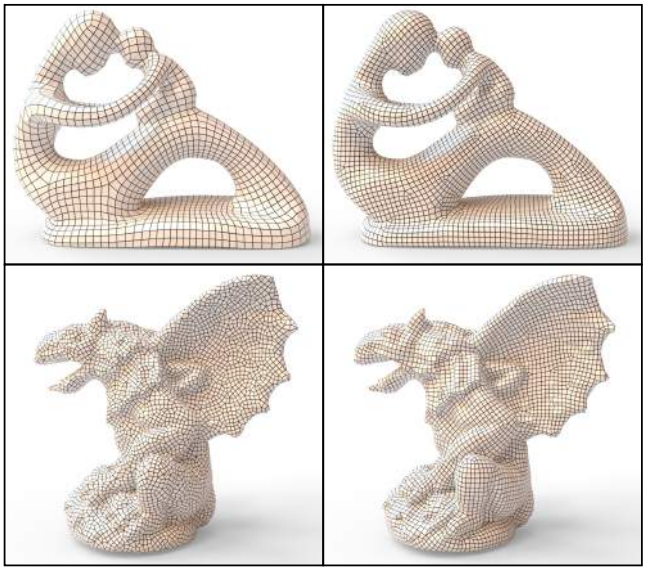


Figure 18: *Quadrilateral meshing comparisons: [Bommes et al. 2009] (top left), [Tarini et al. 2010] (bottom left) and ours (right).*

Quadrilateral meshing. Our results are less regular than global parametrization algorithms due to the higher number of singularities that we use to obtain a perfect alignment to the orientation field and a highly uniform mesh. We compare with [Bommes et al . 2009] in Figure 18 and with [Bommes et al. 2013b] in Figure 19. In both cases, our uniformity is higher, but we introduce more singularities (for example, 208 additional singularities are introduced in Figure Figure 18, top).

On the other hand, compared to the robust and local quadrangulation method proposed in [Tarini et al. 2010], our approach is significantly superior: we obtain higher isotropy and regularity, while naturally aligning to shape features (Figure 18, bottom row).

Our approach is easily generalizable to different field symmetries: in Figure 20, we compare 4-PoSy fields computed starting from a 2-RoSy and a 4-RoSy. It is interesting to observe that when starting from a 2-RoSy, we obtain a consistent assignment of colors in the parametric lines.

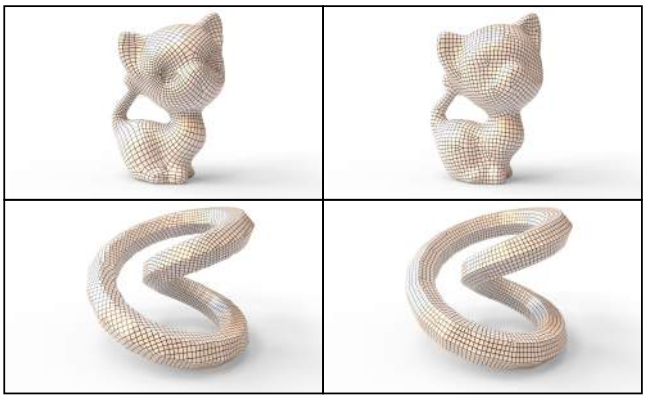


Figure 19: *Quadrilateral meshing comparison continued: [Bommes et al. 2013b] (left), and ours (right). Our method introduces additional singularities to improve the isotropy of the output mesh. On the knot mesh, our parameter-free extrinsic formulation achieves better alignment to shape features.*

Figure 20: *The Owl dataset (left), a 4-PoSy computed using a 2-RoSy (middle) and a 4-RoSy (right). Bottom left: visualization of our extrinsic energy function for the result on the right. The steps through the hierarchy lead to a visible staircase pattern.*

Point cloud meshing. We show an example of a quad-dominant mesh created directly from a point cloud in Figure 21.

Note that since we do not extract a volumetric scalar field, we are not able to automatically fill the regions that are not covered by the point cloud, and those become holes in the final mesh. We use $K = 10$ nearest neighboring points to establish the adjacency information.

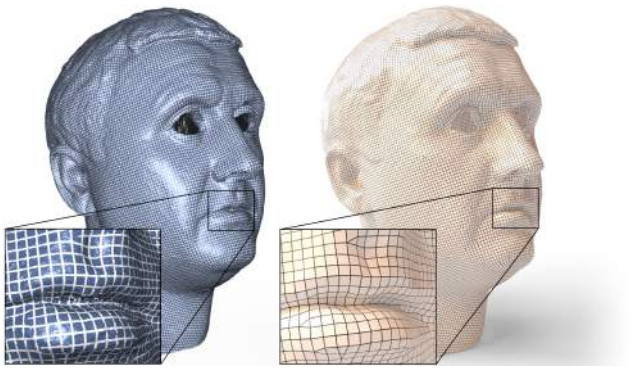


Figure 21: *A large point cloud with 4.9 million points (left) is directly meshed into a high-quality isotropic quad-dominant mesh (right) in 34.4 seconds by our algorithm.*

Custom metrics. It is straightforward to use custom metrics due to the local nature of our iteration. Figure 22 shows examples with a varying scale, created by adapting the parameters of the smoothing iterations when visiting vertices.

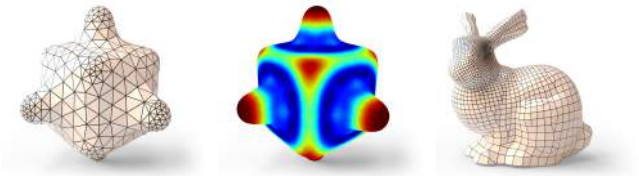


Figure 22: *Custom metrics are easily integrated into our method by changing the parameters of the local smoothing iterations. Left: adaptive resolution in a triangle mesh based on the mean curvature (middle). Right: quad-dominant mesh with varying scale based on the distance to a chosen point on the mesh.*

Relation to convex energies. Convex smoothness energies have been proposed to interpolate N-RoSy fields [Knöppel et al. 2013; Diamanti et al. 2014]; these energies are attractive because their global minimum can be found by solving sparse linear systems. The original formulation by Knöppel et al. involves two major modifications: first, field values are mapped through a nonlinear change of variables, eliminating the need for period jumps. Second, unit norm constraints are removed to ensure that the energy remains stable under refinement. Both of these changes are also simple to realize in the intrinsic version of our method.

We implemented these changes and compared the smoothness energy by Ray et al. [2008] against Knöppel et al.'s convex energy, using the same intrinsic position field and extraction pipeline for both. On average, the convex energy produced meshes with a marginally better distribution of face angles with a standard deviation of 9.55° compared to 9.70° for Ray et al.—a 1.5% relative improvement. However, this came at a cost of 18.4% additional orientation singularities. When used for remeshing, the output of the convex formulation led to considerably higher spatial distortion with a standard deviation 0.183 in normalized face areas compared to 0.168 for Ray et al.—a relative deterioration by 9.2%. The number of position singularities also increased by 20.3% for the convex energy.

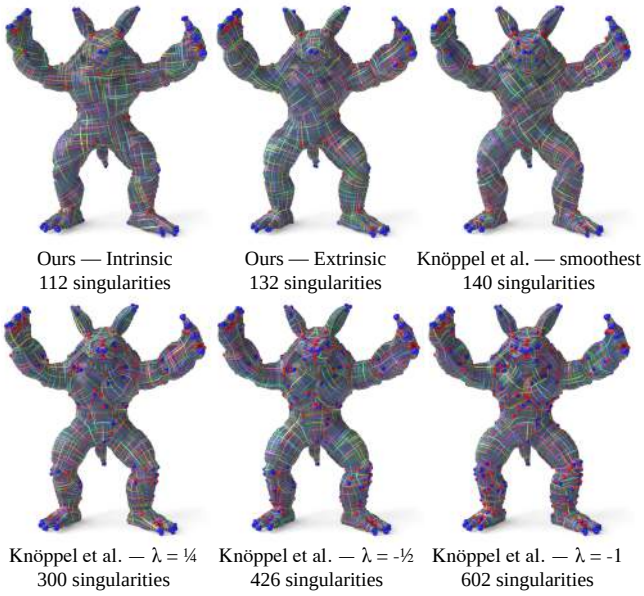


Figure 23: *N-RoSy smoothness: our method generates smoother fields with fewer singularities than recently proposed convex approaches. This even holds when comparing our extrinsic energy to the smoothest unaligned energy of Knöppel et al. [2013].*

The reader might find these results surprising, since the energy by Knöppel et al. was shown to be globally optimal. However, to achieve this optimality, the method must perform a nonlinear transformation of the orientation field, which fundamentally changes the interpretation of smoothness. We conclude that the original non-convex energy is in fact a superior description of field smoothness for the purpose of remeshing.

Figure 23 shows a visualization of fields generated using different energies. In this figure, the fields corresponding to the convex energy were created using the reference implementation by Knöppel et al.

Scalability and robustness. A major advantage of the simplicity and locality of our operations is their scalability and robustness. Our algorithm meshes the 372 million triangle 3D scan in Figure 24 using only 9 minutes and 18 seconds. On all the meshes in the global parametrization benchmark proposed by [Myles et al. 2014], our algorithm completes in less than a second. Note that this benchmark has been designed for non-rounded global parametrization, which is not suitable for quadrangulation, since the grid does not close up on the seams. Our method is the first that can automatically triangulate and quadrangulate such a large database of challenging datasets. We showcase a few results in Figure 25, and attach all 116 models, triangulated and quadrangulated, in the additional material, together with the breakdown of the timings for each step of our algorithm and the instruction to reproduce all results with the provided binary.

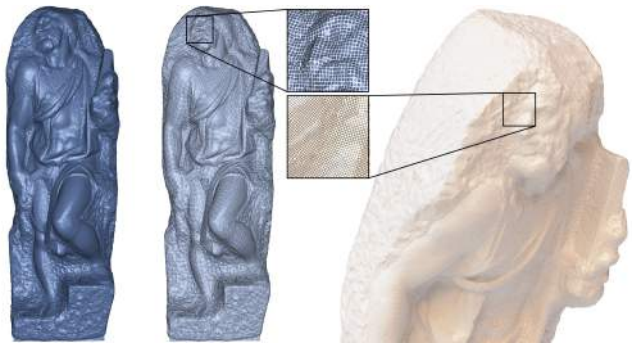


Figure 24: *Our method scales to extremely large datasets, such as the 372M triangle St. Matthew statue acquired by the Digital Michelangelo project [Levoy et al. 2000]. The middle column shows a visualization of the position field, and the right is the final quadrilateral mesh. The entire process takes 9 minutes and 18 seconds.*

Figure 25: *A selection of the 116 models we triangulated and quadrangulated with our algorithm. The wireframe can be seen by zooming in or looking at the mesh files attached to the submission.*

Asymptotic running time. We compared the running time of several meshing techniques on a progressive version of the Lucy statue sampled at regular intervals (Figure 26), running all implementations with default parameters except for a fixed target edge length. For PGP [Ray et al. 2006], we used the implementation in Graphite,

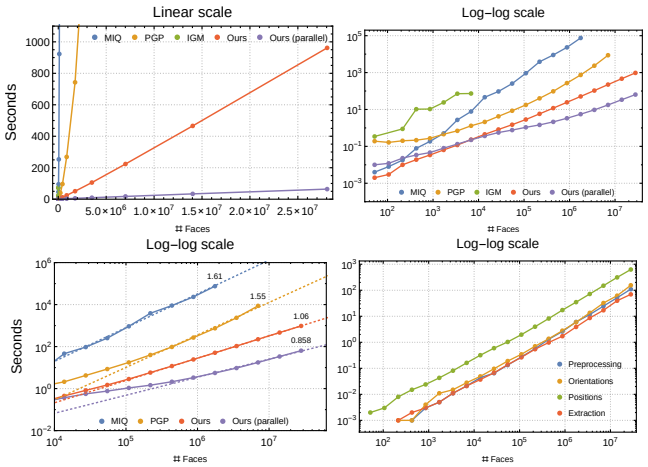


Figure 26: *Top row: Running time comparison against several global methods for a progressive input mesh (`lucy`) sampled at regular intervals between 52 and 28M faces. Top: linear and log-log plots of the running time. Bottom left: Fits obtained by linear regression. The indicated slopes provide an approximation of the polynomial order of complexity. The parallel implementation of our method benefits from larger input sizes, hence the sub-linear behavior. Bottom right: running time of our method's individual components for the same input.*

and for MIQ [Bommes et al. 2009], we used the function `miq` in libigl, which builds on the CoMISo solver by Bommes et al. (for fairness, we only report the time spent in CoMISo). The IGM results [Bommes et al. 2013b] were directly provided by the authors and ran on a MacBook Pro (i7, 2.6 GHz, 16GB RAM). The running time of this method is exponential in the number of singularities, and the authors stopped the CPLEX optimizer after 100 seconds, corresponding to about 200 singularities. We compare against both serial and parallel implementations of our technique. The plots illustrate significant superlinear growth in computation time in currently used global methods, compared to linear growth using our approach.

5 Concluding remarks

We proposed a novel algorithm for isotropic or scale-varying, semi-regular meshing that is robust, scalable and controllable. We demonstrated its practical utility by meshing hundreds of input models, some of which exceed several hundred million elements.

The three steps of our algorithm have been designed to work together, but each one can be used independently: The orientation field optimization can be incorporated into any field-aligned quadrangulation pipeline, enhancing it by providing a parameter-free method that computes shape-aligned fields. Similarly, the position field optimization can process arbitrary orientation fields generated with other methods. These two stages are similar, simple and robust: we provide a self-contained implementation in 100 lines of Python code in addition to an optimized, parallel C++ reference implementation. Finally, our mesh extraction algorithm could be adapted to work on global parameterizations, providing a simple and robust solution for this challenging task. The locality of our approach makes it impervious to non-manifold input (Figure 3).

Considerable flexibility with regards to the input graph enables unified treatment of point clouds, range scans and triangle meshes, for which fields can either be discretized on vertices or faces.

Our algorithm has one major limitation: we introduce more singularities than global parametrization methods. Singularities can be collapsed with our interactive brushes, and we would like to investigate further applications of these brushes to automatically reduce the number of singularities.

Our approach builds upon existing state-of-the-art global parametrization methods [Ray et al. 2006; Bommes et al. 2009], tackling the same problems with a purely local optimization approach that is radically different from existing techniques. We believe that its properties, in particular its linear running time with respect to the size of the input, will make it very appealing to both researchers and practitioners. Many interesting directions for future works are possible, including new types of interactive strokes and the generalization of our approach to semi-regular volumetric meshes.

To support and foster research in this area, we will release our optimized parallel implementation of the algorithms presented in this paper together with all our datasets, comparisons and results.

Acknowledgements

We thank Christian Schüller for 3D-scanning the meshes shown in Figures 1 and 20, Keenan Crane for helpful advice on visualizations of N-RoSy fields, and Olesya Jakob for designing the RoSy/PoSy illustrations shown in Figures 4 and 6. Emily Whiting provided voice narration for the video. This work was supported in part by the ERC Starting Grant iModel (StG-2012-306877) and the EU FP7 project ICT FET Harvest4D (http://www.harvest4d.org/, G.A. no. 323567). Wenzel Jakob was supported by an ETH/Marie Curie fellowship.

References

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. 2005. Multilevel solvers for unstructured surface meshes. *SIAM J. Sci. Comput. 26*, 4 (Apr.).

ALLIEZ, P., MEYER, M., AND DESBRUN, M. 2002. Interactive geometry remeshing. *ACM Trans. Graph. 21*, 3.

ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Trans. Graph. 22*, 3.

ALLIEZ, P., DE VERDIÉRE, E. C., DEVILLERS, O., AND ISENBURG, M. 2005. Centroidal Voronoi diagrams for isotropic surface remeshing. *Graphical Models 67*, 3.

ANDERSON, R. J., AND WOLL, H. 1991. Wait-free parallel algorithms for the union-find problem. In *Proc. STOC*.

BELKIN, M., SUN, J., AND WANG, Y. 2009. Constructing laplace operator from point clouds in rd. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3.

BOMMES, D., LÉVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., AND ZORIN, D. 2013. Quad-mesh generation and processing: A survey. *Comput. Graph. Forum 32*.

BOMMES, D., CAMPEN, M., EBKE, H.-C., ALLIEZ, P., AND KOBBELT, L. 2013. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph. 32*, 4.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. In *In Proc. Symposium of Geometry Processing 2006*.

BOTSCH, M., PAULY, M., ROSSL, C., BISCHOFF, S., AND KOBBELT, L. 2006. Geometric modeling based on triangle meshes. In *ACM SIGGRAPH 2006 Courses*.

Dataset	Algorithm	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{V}_{\text{out}}\rvert$	$S(\text{RoSy})$	Area			Angles				
					σ [0]	Min [1]	Max [1]	σ [0]	Min [60/90]	$\overline{\text{Min}(\square)}$ [60/90]	Max [60/90]	$\overline{\text{Max}(\square)}$ [60/90]
Armadillo	[Pietroni et al.]	20002	21387/**20780**	69(N/A)/**1388(264)**	0.14/**0.11**	0.14/**0.33**	2.79/**1.76**	17.95/**4.37**	1.95/**17.98**	43.74/**56.44**	175.40/**135.87**	78.46/**63.79**
Bunny	[Pietroni et al.]	34834	4502/**4452**	48(N/A)/**315(82)**	0.13/**0.11**	0.38/**0.39**	2.59/**1.72**	16.64/**4.61**	11.03/**19.89**	45.28/**56.23**	157.03/**116.52**	77.79/**64.04**
Gargoyle	[Pietroni et al.]	24992	14582/**14171**	41(N/A)/**1241(491)**	0.13/**0.12**	0.13/**0.36**	1.91/**1.72**	17.00/**5.09**	7.81/**13.43**	44.15/**55.84**	162.59/**152.13**	78.33/**64.44**
Omotondo	[Pietroni et al.]	50002	4862/**4754**	14(N/A)/**413(133)**	0.10/**0.12**	0.43/**0.43**	1.41/**1.70**	11.60/**5.22**	26.16/**25.13**	48.39/**55.80**	106.18/**124.52**	72.72/**64.66**
Bunny	[Alliez et al. 2005]	99999	10020/**10037**	3263(N/A)/**515(102)**	0.11/**0.10**	0.13/**0.30**	4.85/**1.68**	16.14/**4.10**	4.35/**19.01**	44.62/**57.00**	152.32/**135.41**	78.06/**63.23**
MaxPlanck	[Alliez et al. 2005]	23609	100001/**99627**	33268(N/A)/**1534(44)**	0.09/**0.06**	0.00/**0.23**	1.57/**1.95**	11.58/**2.18**	0.00/**22.92**	50.49/**58.45**	180.00/**128.20**	73.19/**61.61**
Rotor	[Alliez et al. 2005]	2400	10210/**9969**	3555(N/A)/**528(52)**	0.17/**0.10**	0.00/**0.32**	2.81/**1.64**	24.84/**4.06**	0.00/**22.06**	36.70/**56.87**	180.00/**116.48**	88.49/**63.30**
David	[Surazhsky et al. 2003]	99977	99980/**99053**	20881(N/A)/**5482(2156)**	0.39/**0.09**	0.02/**0.15**	134.94/**1.96**	9.34/**4.22**	0.92/**5.74**	52.87/**56.99**	167.27/**167.73**	68.63/**63.22**
Fandisk	[Surazhsky et al. 2003]	7229	5051/**4962**	711(N/A)/**401(33)**	0.18/**0.12**	0.21/**0.33**	2.51/**1.64**	8.77/**5.16**	19.44/**18.10**	52.76/**56.01**	118.86/**116.56**	68.68/**64.21**
Rocker Arm	[Surazhsky et al. 2003]	42747	10044/**9990**	2816(N/A)/**533(74)**	0.41/**0.10**	0.04/**0.33**	2.60/**1.60**	10.57/**3.66**	10.41/**27.76**	50.95/**57.01**	154.64/**111.42**	70.47/**63.11**
David	[Alliez et al. 2003]	24085	42871/**41817**	10309(N/A)/**2708(1826)**	0.75/**0.20**	0.01/**0.05**	12.30/**2.60**	24.54/**12.31**	4.40/**3.78**	64.16/**79.26**	178.55/**180.00**	115.76/**100.34**
Pig	[Alliez et al. 2003]	1843	3341/**3406**	436(N/A)/**148(40)**	0.62/**0.17**	0.05/**0.27**	5.02/**1.74**	18.84/**9.06**	9.54/**8.40**	72.04/**82.71**	171.10/**180.00**	107.48/**97.44**
Fandisk	[Marinov and Kobbelt 2006]	7229	754/**2208**	59(N/A)/**108(30)**	0.60/**0.18**	0.26/**0.27**	4.83/**1.84**	19.59/**8.85**	27.16/**19.20**	73.70/**83.46**	161.84/**180.00**	107.79/**97.07**
Fandisk d.	[Marinov and Kobbelt 2006]	7229	1968/**5658**	104(N/A)/**162(30)**	0.54/**0.14**	0.22/**0.37**	5.34/**1.81**	14.08/**6.51**	14.02/**24.67**	79.30/**85.27**	180.00/**180.00**	101.63/**94.98**
Rocker Arm	[Marinov and Kobbelt 2006]	10044	1644/**6562**	117(N/A)/**232(44)**	0.43/**0.15**	0.14/**0.24**	3.40/**1.94**	16.64/**7.43**	28.59/**2.44**	75.51/**83.99**	165.61/**179.99**	105.33/**96.24**
Fandisk	[Ray et al. 2006]	7229	2614/**2386**	110(N/A)/**110(30)**	0.23/**0.18**	0.07/**0.22**	1.86/**1.75**	8.92/**8.21**	34.28/**19.35**	82.78/**83.78**	159.86/**180.00**	97.50/**96.64**
Hand	[Ray et al. 2006]	4242	1082/**1010**	66(N/A)/**51(40)**	0.32/**0.17**	0.08/**0.30**	2.07/**1.65**	10.39/**9.25**	38.14/**29.30**	80.79/**82.27**	147.02/**180.00**	99.76/**97.67**
Hand	[Ray et al. 2006]	10044	1078/**978**	59(N/A)/**74(38)**	0.28/**0.20**	0.02/**0.28**	1.93/**1.71**	12.39/**12.62**	31.00/**26.81**	78.75/**79.39**	152.68/**180.00**	101.88/**101.01**
Hand-highres	[Ray et al. 2006]	4242	10452/**9990**	311(N/A)/**182(40)**	0.19/**0.11**	0.01/**0.21**	2.00/**1.66**	8.12/**6.89**	0.55/**20.28**	83.91/**85.56**	173.57/**180.00**	96.44/**94.61**
Bunny	[Tarini et al. 2010]	34834	11020/**10960**	3438(N/A)/**350(40)**	0.22/**0.15**	0.60/**0.27**	4.23/**1.75**	16.30/**7.86**	24.25/**17.86**	73.70/**83.43**	178.48/**180.00**	109.77/**96.71**
Gargoyle	[Tarini et al. 2010]	24992	11104/**10950**	4283(N/A)/**659(194)**	0.15/**0.19**	0.52/**0.11**	4.24/**1.96**	17.49/**10.97**	1.00/**17.05**	72.22/**80.19**	179.10/**180.00**	110.74/**99.24**
Omotondo	[Tarini et al. 2010]	50002	10049/**10054**	3902(N/A)/**367(70)**	0.21/**0.16**	0.53/**0.19**	2.55/**1.88**	17.13/**8.50**	40.69/**12.95**	72.33/**82.81**	163.30/**180.00**	111.67/**97.47**
Rampant	[Tarini et al. 2010]	49873	10019/**9950**	3749(N/A)/**455(130)**	0.28/**0.17**	0.28/**0.15**	3.64/**1.93**	17.92/**8.93**	12.22/**9.18**	71.76/**82.39**	178.81/**180.00**	111.83/**97.63**
Beetle	[Bommes et al. 2009]	17908	4063/**7928**	6(6)/**286(41)**	0.17/**0.17**	0.22/**0.17**	2.37/**2.30**	8.79/**8.89**	40.80/**19.16**	82.80/**83.36**	172.74/**180.00**	97.27/**96.97**
Fandisk	[Bommes et al. 2009]	7229	766/**1502**	30(30)/**67(30)**	0.42/**0.18**	0.39/**0.24**	3.36/**1.75**	7.54/**8.64**	48.76/**20.20**	83.18/**83.61**	149.55/**179.97**	96.41/**96.72**
Fertility	[Bommes et al. 2009]	13971	3351/**6812**	48(48)/**256(46)**	0.26/**0.15**	0.49/**0.36**	3.30/**1.73**	8.35/**7.52**	31.30/**38.62**	82.08/**83.72**	146.17/**165.17**	97.65/**96.47**
RockerArm	[Bommes et al. 2009]	20776	1127/**2230**	36(36)/**123(36)**	0.30/**0.17**	0.28/**0.39**	2.50/**1.91**	6.88/**9.02**	49.16/**34.76**	83.63/**82.27**	147.98/**180.00**	96.42/**98.10**
RockerArm d.	[Bommes et al. 2009]	20776	9413/**18780**	36(36)/**419(42)**	0.20/**0.12**	0.29/**0.32**	2.17/**1.96**	7.91/**5.91**	29.81/**18.46**	83.79/**85.53**	169.71/**180.00**	96.31/**94.71**
Buddha	[Bommes et al. 2013b]	99370	5366/**5312**	107(107)/**328(94)**	0.28/**0.20**	0.15/**0.13**	2.99/**1.82**	13.39/**10.76**	11.85/**20.59**	77.06/**80.11**	173.79/**180.00**	102.68/**99.87**
Fandisk	[Bommes et al. 2013b]	7229	822/**3296**	30(30)/**117(30)**	0.42/**0.14**	0.13/**0.27**	2.41/**1.60**	13.37/**7.05**	24.51/**33.66**	78.89/**84.53**	172.86/**180.00**	101.22/**95.75**
Feline	[Bommes et al. 2013b]	9998	1047/**12554**	109(109)/**592(112)**	0.34/**0.16**	0.09/**0.17**	3.01/**1.97**	18.64/**8.51**	18.67/**7.12**	70.41/**82.76**	167.59/**179.99**	108.44/**97.31**
Hand	[Bommes et al. 2013b]	4242	182/**1094**	39(39)/**43(36)**	0.31/**0.18**	0.24/**0.28**	1.78/**1.69**	12.91/**8.09**	19.11/**19.00**	77.88/**83.50**	160.00/**160.40**	102.17/**96.50**
Knot	[Bommes et al. 2013b]*	240	5133/**5108**	17(N/A)/**184(4)**	0.29/**0.15**	0.47/**0.21**	2.13/**1.66**	6.09/**8.06**	55.22/**20.57**	84.00/**84.23**	137.02/**180.00**	95.40/**96.32**
Kitten	[Bommes et al. 2013b]*	50000	3607/**3594**	62(N/A)/**127(44)**	0.50/**0.16**	0.01/**0.27**	5.46/**2.01**	8.51/**7.44**	17.08/**28.94**	81.73/**83.51**	172.38/**158.65**	98.01/**96.63**

Table 1: *Statistics of our experiments. The results obtained with our method are highlighted in bold. From left to right: number of vertices in the input and in the output; number of singularities (in parenthesis the number of orientation field singularities, when applicable); standard deviation, mean and max areas, all normalized over the average area; standard deviation, min, average of the min per element, max, average of the max per element angles. All models used to generate this table are attached to the submission as additional material. They were used by the authors for images in the corresponding submissions, except for the last two results which are marked with a "*". These two models were kindly created by the authors for this comparison using a default set of parameters.*

CAMPEN, M., AND KOBBELT, L. 2014. Dual strip weaving: Interactive design of quad layouts using elastica strips. *ACM Trans. Graph. 33*, 6.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6.

CRANE, K., DESBRUN, M., AND SCHRÖDER, P. 2010. Trivial connections on discrete surfaces. *Comput. Graph. Forum 29*, 5.

DANIELS, J., SILVA, C. T., SHEPHERD, J., AND COHEN, E. 2008. Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, 5.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *Proc. ACM SIGGRAPH*.

DIAMANTI, O., VAXMAN, A., PANOZZO, D., AND SORKINE-HORNUNG, O. 2014. Designing N-PolyVector fields with complex polynomials. *Comput. Graph. Forum 33*, 5.

DIAMANTI, O., VAXMAN, A., PANOZZO, D., AND SORKINE-HORNUNG, O. 2015. Integrable PolyVector fields. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 34*, 4.

EBKE, H.-C., BOMMES, D., CAMPEN, M., AND KOBBELT, L. 2013. QEx: Robust quad mesh extraction. *ACM Trans. Graph.*.

EBKE, H.-C., CAMPEN, M., BOMMES, D., AND KOBBELT, L. 2014. Level-of-detail quad meshing. *ACM Trans. Graph. 33*, 6.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proc. ACM SIGGRAPH*.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proc. ACM SIGGRAPH*.

HUANG, J., ZHANG, M., MA, J., LIU, X., KOBBELT, L., AND BAO, H. 2008. Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph. 27*, 5.

JIANG, T., FANG, X., HUANG, J., BAO, H., TONG, Y., AND DESBRUN, M. 2015. Frame field generation through metric customization. *ACM Trans. Graph. 34*, 4.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. QuadCover – surface parameterization using branched coverings. *Comput. Graph. Forum 26*, 3.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. Eurographics Symposium on Geometry Processing*.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. *ACM Trans. Graph. 22*, 3.

KNÖPPEL, F., CRANE, K., PINKALL, U., AND SCHRÖDER, P. 2013. Globally optimal direction fields. *ACM Trans. Graph.*.

KOVACS, D., BISCEGLIO, J., AND ZORIN, D. 2015. Dyadic t-mesh subdivision. *ACM Trans. Graph. 34*, 4 (July).

LAI, Y.-K., KOBBELT, L., AND HU, S.-M. 2008. An incremental approach to feature aligned quad dominant remeshing. In *Proc. ACM Symposium on Solid and Physical Modeling*.

LAI, Y.-K., JIN, M., XIE, X., HE, Y., PALACIOS, J., ZHANG, E., HU, S.-M., AND GU, X. 2010. Metric-driven rosy field design and remeshing. *IEEE TVCG 16*, 1.

LEVI, Z., AND ZORIN, D. 2014. Strict minimizers for geometric optimization. *ACM Trans. Graph. 33*, 6.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S.,

DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital Michelangelo project: 3D scanning of large statues. In *Proc. ACM SIGGRAPH*.

LÉVY, B., AND LIU, Y. 2010. Lp centroidal voronoi tessellation and its applications. *ACM Trans. Graph. 29*, 4 (July).

LI, E. R., LÉVY, B., ZHANG, X., CHE, W.-J., DONG, W., AND PAUL, J.-C. 2011. Meshless quadrangulation by global parameterization. *Computers and Graphics*.

LINDSTROM, P., AND TURK, G. 2000. Image-driven simplification. *ACM Trans. Graph. 19*, 3.

LING, R., HUANG, J., JÜTTLER, B., SUN, F., BAO, H., AND WANG, W. 2014. Spectral quadrangulation with feature curve alignment and element size control. *ACM Trans. Graph. 34*, 1.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. ACM SIGGRAPH*.

MARINOV, M., AND KOBBELT, L. 2004. Direct anisotropic quad-dominant remeshing. In *Proc. Pacific Graphics*.

MARINOV, M., AND KOBBELT, L. 2006. A robust two-step procedure for quad-dominant remeshing. *Computer Graphics Forum*.

MICCIANCIO, D. 2001. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing 30*, 6 (Mar.). Preliminary version in FOCS 1998.

MYLES, A., AND ZORIN, D. 2013. Controlled-distortion constrained global parametrization. *ACM Trans. Graph. 32*, 4.

MYLES, A., PIETRONI, N., KOVACS, D., AND ZORIN, D. 2010. Feature-aligned t-meshes. *ACM Trans. Graph. 29*, 4.

MYLES, A., PIETRONI, N., AND ZORIN, D. 2014. Robust field-aligned global parametrization. *ACM Trans. Graph. 33*, 4.

NIESER, M., PALACIOS, J., POLTHIER, K., AND ZHANG, E. 2012. Hexagonal global parameterization of arbitrary surfaces. *IEEE Trans. Visualization and Computer Graphics 18*, 6.

OWEN, S. J. 1998. A survey of unstructured mesh generation technology. In *Proc. IMR*.

PALACIOS, J., AND ZHANG, E. 2007. Rotational symmetry field design on surfaces. *ACM Trans. Graph. 26*, 3.

PANOZZO, D., PUPPO, E., TARINI, M., AND SORKINE-HORNUNG, O. 2014. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Trans. Graph. 33*, 4.

PENG, C.-H., ZHANG, E., KOBAYASHI, Y., AND WONKA, P. 2011. Connectivity editing for quadrilateral meshes. *ACM Trans. Graph. 30*, 6.

PIETRONI, N., TARINI, M., AND CIGNONI, P. Almost isometric mesh parameterization through abstract domains. *IEEE Trans. Visualization and Computer Graphics 16*, 4.

PIETRONI, N., TARINI, M., SORKINE, O., AND ZORIN, D. 2011. Global parametrization of range image sets. *ACM Trans. Graph.*.

PINKALL, U., AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experiment. Math. 2*, 1.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Trans. Graph.*.

RAY, N., VALLET, B., LI, W. C., AND LÉVY, B. 2008. N-symmetry direction field design. *ACM Trans. Graph. 27*, 2.

SEDERBERG, T. W., ZHENG, J., BAKENOV, A., AND NASRI, A. 2003. T-splines and T-NURCCs. *ACM Trans. Graph. 22*, 3.

SIFRI, O., SHEFFER, A., AND GOTSMAN, C. 2003. Geodesic-based surface remeshing. In *Proc. Intnl. Meshing Roundtable*.

SURAZHSKY, V., AND GOTSMAN, C. 2003. Explicit surface remeshing. In *Proc. Symposium on Geometry Processing*.

SURAZHSKY, V., ALLIEZ, P., AND GOTSMAN, C. 2003. Isotropic remeshing of surfaces: A local parameterization approach. In *Proc. International Meshing Roundtable*.

TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., AND SORKINE-HORNUNG, O. 2013. Sketch-based generation and editing of quad meshes. *ACM Trans. Graph. 32*, 4.

TARINI, M., PIETRONI, N., CIGNONI, P., PANOZZO, D., AND PUPPO, E. 2010. Practical quad mesh simplification. *Comput. Graph. Forum 29*, 2.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. ACM SIGGRAPH*.

TURK, G. 1992. Re-tiling polygonal surfaces. In *Proc. ACM SIGGRAPH*.

YAN, D.-M., LÉVY, B., LIU, Y., SUN, F., AND WANG, W. 2009. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Comput. Graph. Forum 28*, 5.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2006. Vector field design on surfaces. *ACM Trans. Graph. 25*, 4 (Oct.).

ZHANG, M., HUANG, J., LIU, X., AND BAO, H. 2010. A wave-based anisotropic quadrangulation method. *ACM Trans. Graph.*.

A Appendix

Lattice rounding operation. Given a regular grid with orientation $\mathbf{o}$, position $\mathbf{p}$ and normal $\mathbf{n}$, the following operation rounds a position $\mathbf{p}'$ to the nearest lattice point:

$$\mathbf{round}_4(\mathbf{n},\mathbf{o},\mathbf{p},\mathbf{p}') \coloneqq \mathbf{p} + \rho\left[\mathbf{o}\lfloor\gamma_1+1/2\rfloor + \mathbf{o}'\lfloor\gamma_2+1/2\rfloor\right],$$

where $\mathbf{o}' \coloneqq \mathrm{rot}(\mathbf{n}, 2\pi/s_p)\mathbf{o}$, $\gamma_1 \coloneqq \rho^{-1}\langle\mathbf{p}'-\mathbf{p},\mathbf{o}\rangle$ and $\gamma_2 \coloneqq \rho^{-1}\langle\mathbf{p}'-\mathbf{p},\mathbf{o}'\rangle$ and ρ is the target edge length. For the $s_p = 6$ case, the rounding operation first unwarps the lattice:

$$\begin{aligned}\mathbf{round}_6(\mathbf{n},\mathbf{o},\mathbf{p},\mathbf{p}') \coloneqq \quad & \mathbf{p} + \rho\,\mathbf{o}\,\lfloor(\ 4\gamma_1 - 2\gamma_2)/3 + 1/2\rfloor + \\ & \rho\,\mathbf{o}'\lfloor(-2\gamma_1 + 4\gamma_2)/3 + 1/2\rfloor.\end{aligned}$$

Intermediate position. We define a position $\mathbf{q}_{ij}$ that minimizes the distance to vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ while being located in their respective tangent planes, i.e.:

$$\begin{aligned}&\text{minimize } \ \|\mathbf{v}_i - \mathbf{q}_{ij}\|_2^2 + \|\mathbf{v}_j - \mathbf{q}_{ij}\|_2^2 \\ &\text{subject to } \ \langle\mathbf{n}_i,\mathbf{q}_{ij}\rangle = \langle\mathbf{n}_i,\mathbf{v}_i\rangle \ \text{ and } \ \langle\mathbf{n}_j,\mathbf{q}_{ij}\rangle = \langle\mathbf{n}_j,\mathbf{v}_j\rangle.\end{aligned}$$

This constrained least-squares problem has a simple solution:

$$\mathbf{q}_{ij} = \tfrac{1}{2}(\mathbf{v}_i + \mathbf{v}_j) - \tfrac{1}{4}(\lambda_i\mathbf{n}_i + \lambda_j\mathbf{n}_j),$$

where the Lagrange multiplier λ_i is

$$\lambda_i = \frac{2\left\langle(\mathbf{n}_i + \langle\mathbf{n}_i,\mathbf{n}_j\rangle\mathbf{n}_j,\ \mathbf{v}_j - \mathbf{v}_i\right\rangle}{1 - \langle\mathbf{n}_i,\mathbf{n}_j\rangle^2 + \varepsilon},$$

and λ_j is defined analogously with i and j swapped. The parameter ε (set to 10^{-4} in our implementation) ensures that $\mathbf{q}_{ij}$ approximates the arithmetic mean of $\mathbf{v}_i$ and $\mathbf{v}_j$ when $\mathbf{n}_i \approx \mathbf{n}_j$.