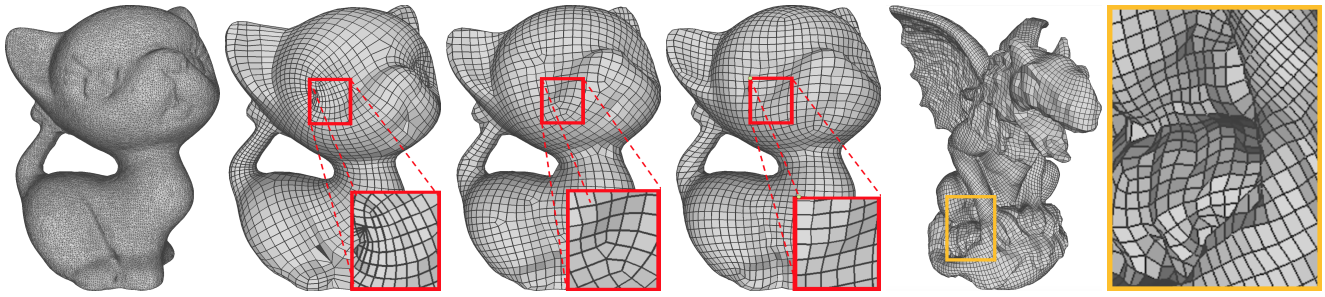
DOI: 10.1111/cgf.13498
Eurographics Symposium on Geometry Processing 2018
T. Ju and A. Vaxman
(Guest Editors)

Volume 37 (2018), Number 5

QuadriFlow: A Scalable and Robust Method for Quadrangulation

Jingwei Huang[†1] Yichao Zhou[†2] Matthias Niessner[3] Jonathan Richard Shewchuk[2] Leonidas J. Guibas[1]

[1]Stanford University [2]University of California, Berkeley [3]Technical University of Munich



(a) Input triangulation (b) Integer-Grid Maps (c) Instant Meshes (d) QuadriFlow (e) QuadriFlow (f) QuadriFlow zoom

Figure 1: *Quadrilateral meshes generated by Integer-Grid Maps (IGM) [BCE*13], Instant Meshes [JTPSH15], and our algorithm QuadriFlow. IGM (b) sometimes produces badly distorted quadrilaterals. Instant Meshes (c) produces more vertices of valence 3 or 5. QuadriFlow (d, e, f) produces fewer vertices of irregular valence than Instant Meshes by removing all the singularities from the position field, while producing less distortion than IGM.*

Abstract

QuadriFlow is a scalable algorithm for generating quadrilateral surface meshes based on the Instant Field-Aligned Meshes *of Jakob et al. (*ACM Trans. Graph. *34(6):189, 2015). We modify the original algorithm such that it efficiently produces meshes with many fewer singularities. Singularities in quadrilateral meshes cause problems for many applications, including parametrization and rendering with Catmull–Clark subdivision surfaces. Singularities can rarely be entirely eliminated, but it is possible to keep their number small. Local optimization algorithms usually produce meshes with many singularities, whereas the best algorithms tend to require non-local optimization, and therefore are slow. We propose an efficient method to minimize singularities by combining the Instant Meshes objective with a system of linear and quadratic constraints. These constraints are enforced by solving a global minimum-cost network flow problem and local boolean satisfiability problems. We have verified the robustness and efficiency of our method on a subset of ShapeNet comprising 17,791 3D objects in the wild. Our evaluation shows that the quality of the quadrangulations generated by our method is as good as, if not better than, those from other methods, achieving about four times fewer singularities than Instant Meshes. Other algorithms that produce similarly few singularities are much slower; we take less than ten seconds to process each model. Our source code is publicly available.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Quadrangulation

1. Introduction

Many applications in computer graphics and shape modeling use models of surfaces that are composed of triangles or polygons. Although triangular meshes are the most commonly-used surface models, quadrilateral meshes are also important because they are particularly useful for Catmull–Clark subdivision surfaces, texturing, mesh editing, visualization, and physics-based simulation.

State-of-the-art algorithms for quadrilateral surface meshing typically compute, as a first step, an *orientation field* that assigns local coordinate axes to some points on the input surface [Knu95, RLL*06, KNP07, BZK09]. The *Instant Field-Aligned*

† indicates equal contribution

Meshes algorithm of Jakob et al. [JTPSH15] subsequently computes a *position field* that assigns local coordinates to those points. The orientation field determines the directions of the edges of a quadrilateral mesh, and the position field determines where the mesh vertices are placed. Ideally, both fields should vary smoothly over the surface, while obeying constraints that help to align the mesh with the sharp edges and the curvature of the object.

Both types of field can have irregularities called *singularities*. If the fields are defined continuously over the surface, a singularity is a region where one of the fields is not locally smooth. The pitfall of these singularities is that the quad mesh subsequently produced is likely to have an *irregular vertex*—a vertex whose valence is not 4—near the singularity. Unfortunately, irregular vertices can cause problems for applications; for instance, they cause unsightly visual artifacts in Catmull–Clark subdivision, or additional work for an artist to edit a model.

Algorithms that rely purely on local mesh computations are fast, but they produce meshes with many singularities. It is possible to modify the fields to move singularities and sometimes even to eliminate them; but eliminating a singularity usually involves merging pairs of singularities located all around the geometry, which is "nonlocal." A global view of quad meshing taken by Bommes et al. [BZK09], who cast the problem of seamless global parametrization as a mixed-integer constrained optimization problem (MIP). The method produces quadrilateral surface meshes of very high quality, but it is slow and it does not scale well to large meshes. By contrast, the much more efficient Instant Meshes algorithm of Jakob et al. [JTPSH15] uses local smoothing operators to compute an orientation field and a position field quickly. Their method is scalable and produces high quality quad-dominant meshes without much distortion. However, it may produce singularities in the position field in addition to singularities in orientation field.

We present *QuadriFlow*, a scalable, robust algorithm for automatic quad meshing that builds upon Instant Meshes but uses a global method to remove all the singularities from the position field. We do not change the orientation field, which typically has many fewer singularities. Our method solves a minimum cost network flow problem as a subproblem, for which efficient algorithms are available. The speed and reliability of our algorithm can enable designers to work on a modeling task interactively and extract a quad mesh in less than a second for tens of thousands of faces, and enable physical simulations to perform per-timestep remeshing updates. Our current implementation is a *remeshing* algorithm, meaning that its input is a triangular mesh of the input surface (like many other quad meshing algorithms), though it could be modified to take a point cloud input as Instant Meshing does.

We view singularity-free position field computation as a globally constrained optimization problem. Unlike Bommes et al., we do not solve the problem by mixed-integer programming. Instead, we split it into three stages.

- Compute the orientation and position fields just as Jakob et al. do, without enforcing additional constraints.
- Enforce our constraints by modifying only the integer variables of the position field, changing the integers as little as possible.
- Re-optimize the continuous variables of the position field with the integer variables held fixed.

The third stage is not difficult, requiring the solution of a linear system. Our main contribution is a fast and effective method for the second, largely combinatorial stage. Because of the regularity constraints, the second stage is a mixed-integer programming problem, for which it is NP-hard to find an optimal solution, but we can obtain good approximate solutions in practice. We reduce the problem to an integer linear program (ILP). We approximate the ILP as an easier *minimum cost network flow* (MCF) problem, which can be solved in polynomial time [Kle67]. We further improve the efficiency by a multi-resolution algorithm. To enforce consistent triangle orientations (no inverted triangles), we impose a set of quadratic inequality constraints. We are able to satisfy most of these constraints through simple greedy edge contractions; we find we can satisfy most of the remaining difficult ones by locally solving a small boolean satisfiability problem (SAT).

By replacing the MIP solver with an MCF solver that globally reduces the number of singularities, plus edge contractions and a SAT solver that locally impose triangle orientation constraints, we obtain a scalable quad remesher that produces many fewer singularities than Instant Meshes, often by a factor of four, while being much faster than the method of Bommes et al. QuadriFlow remeshes a one-million triangle mesh in 5 seconds, which is comparable to the interactive method of Ebke et al. [ESCK16]. Our meshes have less distortion compared to other global methods, and rarely suffer from nonmanifold structure or holes, thanks to the consistent orientation constraints. In a test on 17,000+ surfaces created from ShapeNet [CFG*15], QuadriFlow robustly removed all the singularities from every discrete position field.

2. Related Work

Robust Quadrangulations. Robust quadrangulations based on local optimization have a rich literature. Bommes et al. [BLP*13] survey many existing methods for quad-mesh generation and processing. Many methods transform a pre-existing triangle mesh into an all-quadrilateral mesh: Q-Morph [OSCS99] does so with an advancing front algorithm. Blossom-Quad [RLS*12] uses a perfect matching algorithm to pair triangles into quads with a global optimal solution. Velho and Zorin [VZ01] greedily identify the most eligible neighboring triangles to pair. SQuad [GLLR11] improves the representation of the connectivity of meshes, which can be applied to quadrangulation. Since these methods are not guided by an orientation field, they have difficulty achieving global regularity or smoothness, and their meshes are often very irregular or have many singularities. Spectral and Morse complex-based algorithms [DBG*06, ZHLB10, LHJ*14] require no integer optimization but singularity control is hard for them.

Orientation Fields. An orientation field is a powerful tool to guide the edge directions in a quad mesh. Specifically, we use 4-way rotationally symmetric orientation fields [RVLL08, LJX*10], which are explained in Section 3.1. The target directions are derived from the principal curvatures [CSM03, CP05], but modified to vary smoothly. Smooth orientation fields are generated by optimizing a nonlinear energy function based on periodic functions [HZ00, RVAL09] or a mixed-integer representation [RVLL08, BZK09]. However, these approaches may get stuck in poor local minima

that have many singularities. Knöppel et al. [KCPS13] propose a global optimization method to obtain better minima. Many approaches [RVLL08,RVAL09,CDS10,DVPSH14,JFH*15] integrate user interactions to help remove singularities.

Field-Aligned Quadrangulations. A direct approach to generating a quad mesh is to trace the curves in an orientation field [ACSD*03], but it is difficult to control the sizes of the quads obtained that way. Lai et al. [LKH08] directly optimize a triangle mesh to align its edges to an orientation field, then extract a quad mesh by pairing triangles. These methods are local, so they generate many unnecessary singularities.

Another line of work that deals with global parameterization of maps is based on global optimization that explicitly bounds the map distortion and produces injective maps. The parameterization can be extracted as quad meshes using `libQEx` [EBCK13]. Levi and Zorin [LZ14] achieve minimum worst-case distortion and prioritize higher distortion reduction. Chien et al. [CLW16] solve a locally injective map efficiently with sequential convex programming. Myles et al. [MPZ14] use cross-field line tracing to initialize the quad patch partition, then compute a bijective global parametrization.

Global methods aim at jointly optimizing the parametrization with integer constraints that are usually not polynomial-time tractable. The objective is typically represented as an MIP problem [BZK09]. The numerical solvers are designed to enforce low distortion and reduce the number of singularities [BCE*13, MZ13, LZ14, MPZ14]. Another global integrable approach [DVPSH15] minimizes a nonlinear energy, but the optimization process is still challenging. The output of the global methods is called the Integer-Grid Map (IGM) [BCE*13], where the final quad mesh can be extracted by `libQEx` [EBCK13]. These methods have full control of edge alignment and singularity placement, and usually generate very high quality quad meshes. However, their implementations are complex, and usually not scalable. Quantized Global Optimization [CBK15] uses motorcycle graphs to quickly construct a valid quantization based on a seamless parametrization.

3. Methods

In this section, we discuss the computation of orientation and position fields and our innovations for removing singularities from the latter. We first review the Instant Meshes algorithm of Jakob et al. [JTPSH15], which we use to initialize the orientation and position fields, in Section 3.1. We impose constraints on the position field's integer offsets to reduce the number of singularities in Section 3.2. We propose algorithms to enforce those constraints in Sections 3.3 and 3.4. We re-optimize the position field in Section 3.5 and extract a quad mesh in Section 3.6. Table 1 summarizes our notation. Figure 2 distinguishes our contributions from the parts we borrow from Instant Meshes.

3.1. Instant Field-Aligned Meshes

Here, we recapitulate the main ideas of Instant Meshes [JTPSH15]. Let $\mathcal{M} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ be an input surface triangulation where $\mathcal{V} = \{1, 2, \ldots, |\mathcal{V}|\}$ is a set of vertex indices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed edges, and $\mathcal{F}$ is a set of triangular faces.

Table 1: *Notation.*

$\mathcal{M}$	triangle mesh	$\mathcal{V}$	vertices in $\mathcal{M}$
$\mathcal{E}$	directed edges in $\mathcal{M}$	$\mathfrak{E}$	redirected edges in $\mathcal{M}$
$\mathcal{F}$	triangles in $\mathcal{M}$	$\mathbf{n}_u$	normal vector at u
$\mathbf{o}_u$	orientation field at u	$\mathbf{p}_u$	position field at u
ρ	desired edge length	$\mathbf{O}_u$	local frame basis at u
$\mathbf{R}_2(k)$	2D rotation through $90 \cdot k^\circ$		
$\mathbf{R}_3(\mathbf{n}, k)$	3D rotation around axis $\mathbf{n}$ through $90 \cdot k^\circ$		
k_{uv}, k_{vu}	integer rotations to align $\mathbf{o}_u, \mathbf{o}_v$		
$\mathbf{t}_{uv}, \mathbf{t}_{vu}$	integer offsets to pull $\mathbf{p}_u, \mathbf{p}_v$ together		
$\mathbf{d}_{uv}, \mathbf{d}_{vu}$	integer offsets at redirected edge $(u, v) \in \mathfrak{E}$		
$\mathcal{R}^w_{uv}$	rotation matrix to rotate $\mathbf{d}_{uv}$ to the frame of O_w		
G	network used in the minimum cost flow problem		
V	vertices in G	E	edges in G
c	capacity of an edge	w	cost of an edge
s	source of G	t	sink of G

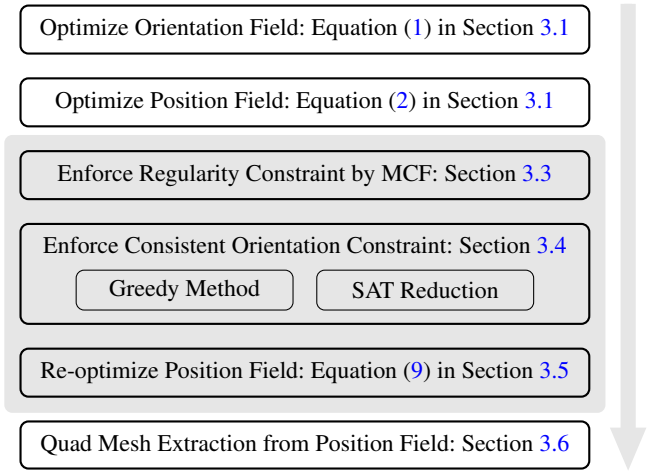


Figure 2: *The pipeline of QuadriFlow. The white nodes are from Instant Meshes [JTPSH15]; the shaded nodes are our contribution.*

Orientation Field. The first step is to compute a four-way rotationally symmetric (*4-RoSy*) *orientation field* [RVLL08]. 4-RoSy fields are also known as *cross fields*; the orientation field maps each vertex of $\mathcal{M}$ to a cross that is locally tangent to the surface. Each cross is invariant to rotations by 90° in its tangent plane. The orientation field guides the alignment of the edges in the quad mesh. For each vertex $v \in \mathcal{V}$, Jakob et al. represent the cross at v with a *representative direction* vector $\mathbf{o}_v \in \mathbb{R}^3$ that lies in vertex v's tangent plane; that is, $\mathbf{o}_v$ is orthogonal to v's normal vector $\mathbf{n}_v$. To obtain representational invariance, let $\mathbf{R}_3(\mathbf{n}, k) \in \mathbb{R}^{3\times 3}$ be a 3D rotation matrix that rotates a vector through $90 \cdot k^\circ$ counterclockwise around an axis $\mathbf{n}$ (which is the local normal vector). To smooth the orientation field $\mathbf{o}$, Jakob et al. define the *extrinsic smoothness energy* of $\mathbf{o}$ to be

$$E_o(\mathbf{o}, k) = \sum_{(u,v)\in\mathcal{E}} \measuredangle\Big(\mathbf{R}_3(\mathbf{n}_u, k_{uv})\mathbf{o}_u, \mathbf{R}_3(\mathbf{n}_v, k_{vu})\mathbf{o}_v\Big)^2,$$

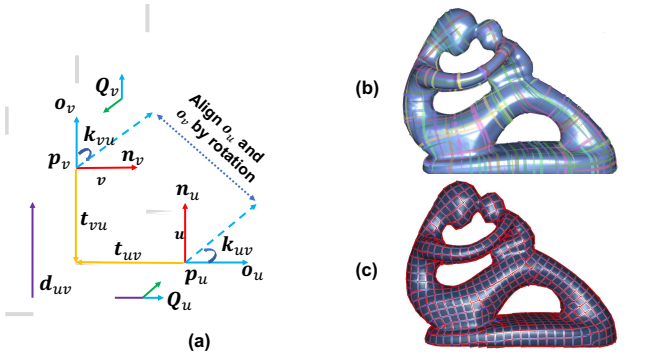


Figure 3: *Geometry (a), orientation field (b), and position field (c), per Jakob et al. [JTPSH15]. The adjacent vertices u and v with positions* $\mathbf{p}_u$ *and* $\mathbf{p}_v$ *lie on orthogonal tangent planes with normal vectors* $\mathbf{n}_u$ *and* $\mathbf{n}_v$*.* $\mathbf{o}_u$ *and* $\mathbf{o}_v$ *can be aligned by rotating them through* $90 \cdot k^\circ_{uv}$ *and* $90 \cdot k^\circ_{vu}$*. Jakob et al. try to make a point on one lattice nearly coincide with a point on the other lattice; these two points are at an integer offset of* $\mathbf{t}_{uv}$ *lattice points from* $\mathbf{p}_u$ *and* $\mathbf{t}_{vu}$ *lattice points from* $\mathbf{p}_v$ *in local frames. The integer offset* $\mathbf{d}_{uv}$ *is the sum of these integer translations in u's frame.*

where $\measuredangle(\cdot,\cdot)$ denotes the angle between two vectors, $k_{uv}, k_{vu} \in \{0,1,2,3\}$, and $\mathbf{n}_u$ and $\mathbf{n}_v$ are the normal vectors of vertices u and v. The integers k_{uv} and k_{vu} are chosen to make the angle as small as possible. Figure 3(a) illustrates how $\mathbf{o}_u$ and $\mathbf{o}_v$ are realigned by k_{uv} and k_{vu} to point in similar directions (in this example, the same direction). Jakob et al. show how to use a mixed-integer Gauss–Seidel algorithm to find

$$\mathbf{o}^*, k^* = \operatorname*{argmin}_{\mathbf{o},k} E_o(\mathbf{o}, k), \tag{1}$$

thereby obtaining a smooth orientation field like the one illustrated in Figure 3(b). With this extrinsic energy, the orientation field $\mathbf{o}^*$ also aligns well with shape features of the input geometry.

Position Field. Given an orientation field $\mathbf{o}^*$, Jakob et al. compute a consistent *position field* that determines the placement of the vertices of the quadrilateral mesh. Let ρ be a user-specified distance specifying the desired length of the edges in the output mesh. For each vertex $v \in \mathcal{V}$, the position field maps v to a square lattice that lies in v's tangent space, has all its edge lengths equal to ρ, and is aligned with the cross field as indicated by $\mathbf{o}^*_v$. The lattice is invariant under "horizontal" or "vertical" translations of distance ρ—that is, in the directions $\mathbf{o}^*_v$ or $\mathbf{n}_v \times \mathbf{o}^*_v$. (Jakob et al. call this *positional symmetry* or *PoSy*.) Hence the only degrees of freedom for the lattice can be specified as "fractional" horizontal and vertical translations in the range $[0,\rho)$. For a vertex $v \in \mathcal{V}$, let $\mathbf{p}_v \in \mathbb{R}^3$ be a representative lattice point near vertex v in v's tangent plane. Let

$$\mathbf{O}_v = [\mathbf{o}^*_v, \mathbf{n}_v \times \mathbf{o}^*_v]$$

be a basis for v's tangent plane whose basis vectors are aligned with the orientation field. The *tangent lattice* for v is

$$\begin{aligned} \mathcal{T}(\mathbf{p}_v, \mathbf{n}_v, \mathbf{o}^*_v) &= \{\mathcal{T}(\mathbf{p}_v, \mathbf{n}_v, \mathbf{o}^*_v, \mathbf{t}) : \mathbf{t} \in \mathbb{Z}^2\} \text{ where} \\ \mathcal{T}(\mathbf{p}_v, \mathbf{n}_v, \mathbf{o}^*_v, \mathbf{t}) &= \mathbf{p}_v + \rho \mathbf{O}_v \mathbf{t}. \end{aligned}$$

It is desirable for vertices joined by an edge in $\mathcal{E}$ to have tangent lattices that coincide or nearly coincide—or, more realistically, to each have one nearby vertex in its lattice such that the two vertices nearly coincide. Figure 3(a) illustrates the two tangential lattices of vertices u and v represented by $\mathbf{p}_u$ and $\mathbf{p}_v$, whose lattice points happen to coincide on the intersection of the two planes. To smooth the position field $\mathbf{p}$, Jakob et al. define the extrinsic smoothness energy of $\mathbf{p}$ to be

$$E_p(\mathbf{p}, \mathbf{t}) = \sum_{(u,v)\in\mathcal{E}} \left\| \mathcal{T}(\mathbf{p}_u, \mathbf{n}_u, \mathbf{o}^*_u, \mathbf{t}_{uv}) - \mathcal{T}(\mathbf{p}_v, \mathbf{n}_v, \mathbf{o}^*_v, \mathbf{t}_{vu}) \right\|_2^2,$$

where $\mathbf{t}_{uv}, \mathbf{t}_{vu} \in \mathbb{Z}^2$ are selected to remove the translation ambiguity and make the distance as small as possible. As with the orientation field, Jakob et al. use a mixed-integer Gauss–Seidel algorithm to find the minimizer

$$\mathbf{p}^*, \mathbf{t}^* = \operatorname*{argmin}_{\mathbf{p},\mathbf{t}} E_p(\mathbf{p}, \mathbf{t}). \tag{2}$$

This smoothing procedure produces a position field $\mathbf{p}^*$ smooth enough to obtain a quad mesh, most of whose edges have length close to (but not exactly) ρ, as shown in Figure 3(c).

3.2. Integer Offsets and Constraints

Integer Offsets. Jakob et al. note that a singularity appears in the position field when the sum of integer offsets over a triangle in the triangle mesh $\mathcal{M}$ is nonzero. To mathematically express this observation, we first define the integer offset along an edge of $\mathcal{M}$. We find it useful to redirect the mesh edges $\mathcal{E}$ in a canonical way, with each edge directed from the vertex with lesser index to the vertex with greater index.

Definition 3.1. *Let*

$$\mathfrak{E} := \{(u,v) : u < v \text{ and } ((u,v) \in \mathcal{E} \text{ or } (v,u) \in \mathcal{E})\}$$

be the set of redirected edges. *For each redirected edge* $e = (u,v) \in \mathfrak{E}$*, define the 2D integer offset*

$$\mathbf{d}^*_e = \mathbf{t}^*_{uv} - \mathbf{R_2}(k^*_{uv} - k^*_{vu})\mathbf{t}^*_{vu},$$

the integer offset from u to v in u's frame, where $\mathbf{R_2}(k)$ *is a 2D rotation matrix through* $90 \cdot k^\circ$*. For notational convenience, we use* $\mathbf{d}_e$*,* $\mathbf{d}_{uv}$*, and* $\mathbf{d}_{vu}$ *interchangeably in the following paragraphs.*

After we optimize (2), $\mathbf{p}^*_u$ and $\mathbf{p}^*_v$ should be close to each other after applying the integer offsets $\mathbf{t}^*_{uv}$ and $\mathbf{t}^*_{vu}$ in the frames of $\mathbf{O}_u$ and $\mathbf{O}_v$, respectively. The formula for the offset is based on the observation that $\mathbf{t}_{vu}$ in frame $\mathbf{O}_v$ can be measured in frame $\mathbf{O}_u$ as $\mathbf{R_2}(k_{uv} - k_{vu})\mathbf{t}_{vu}$. See the purple vector in Figure 3(a) for an interpretation of $\mathbf{d}_{uv}$.

To detect singularities in the position field, we sum up the integer offsets of the three edges of a triangle under a single frame. For any $(u,v) \in \mathcal{E}$ and any w adjacent to both u and v (including $w = u$), the integer offset for an edge (u,v) in the frame of vertex w can be achieved by applying a 2D rotation $\mathcal{R}^w_{uv}$ to $\mathbf{d}_{uv}$. Specifically, if $u < v$, we can directly change the frame from u to w by a rotation along edge (w,u): $\mathcal{R}^w_{uv} = \mathbf{R}_2(k_{wu} - k_{uw})$. For $u > v$, we first compute the translation from u to v in v's frame as $-\mathbf{d}_{uv}$, and apply the rotation along edge (w,v): $\mathcal{R}^w_{uv} = -\mathbf{R}_2(k_{wv} - k_{vw}) = \mathbf{R}_2(k_{wv} - k_{vw} + 2)$.

Regularity Constraints. If the sum of the integer offsets over a triangle element is nonzero, this triangle encloses a singularity in the position field [JTPSH15]. Hence we can remove the singularities from the position field if we can enforce

$$\mathcal{R}^u_{uv}\mathbf{d}_{uv} + \mathcal{R}^u_{vw}\mathbf{d}_{vw} + \mathcal{R}^u_{wu}\mathbf{d}_{wu} = 0 \quad \forall \Delta_{uvw} \in \mathcal{F}. \tag{3}$$

Consistent Orientation Constraints. If the position field has triangles with negative orientation, they lead to inverted faces in the quad mesh. To ensure that there are no inverted faces in the output, each triangle should have nonnegative orientation, i.e.,

$$\det\left[\mathcal{R}^u_{uv}\mathbf{d}_{uv},\ \mathcal{R}^u_{uw}\mathbf{d}_{uw}\right] \geq 0. \tag{4}$$

This constraint is similar to the consistent orientation constraint proposed by Bommes et al. [BCE*13]. Jakob et al. do not enforce this constraint, so their meshes can have inverted faces.

Formulation. Combining Equations (2), (3), and (4), we formulate the position field problem as follows.

$$\begin{aligned} \underset{\mathbf{p},\mathbf{t}}{\text{minimize}} \quad & E_p(\mathbf{p},\mathbf{t}) \\ \text{subject to} \quad & \mathcal{R}^u_{uv}\mathbf{d}_{uv} + \mathcal{R}^u_{vw}\mathbf{d}_{vw} + \mathcal{R}^u_{wu}\mathbf{d}_{wu} = 0 \quad \forall \Delta_{uvw} \in \mathcal{F}, \\ & \det\left[\mathcal{R}^u_{uv}\mathbf{d}_{uv},\ \mathcal{R}^u_{uw}\mathbf{d}_{uw}\right] \geq 0 \quad \forall \Delta_{uvw} \in \mathcal{F}. \end{aligned}$$

This is a mixed-integer programming problem. Solving it directly is NP-hard and thus not generally scalable. Our method finds an approximate solution by first computing $\mathbf{p}^*$ and $\mathbf{t}^*$ without constraints, using Gauss–Seidel iterations [JTPSH15], then computing $\mathbf{d}^*$ from $\mathbf{t}^*$ according to the definition, then adjusting $\mathbf{d}$ to enforce Equations (3) and (4). To enforce Equation (3), we try to solve an integer programming problem, namely to

$$\begin{aligned} \underset{\mathbf{d}}{\text{minimize}} \quad & \|\mathbf{d} - \mathbf{d}^*\|_1 \\ \text{subject to} \quad & \mathcal{R}^u_{uv}\mathbf{d}_{uv} + \mathcal{R}^u_{vw}\mathbf{d}_{vw} + \mathcal{R}^u_{wu}\mathbf{d}_{wu} = 0 \quad \forall \Delta_{uvw} \in \mathcal{F}. \end{aligned} \tag{5}$$

We temporarily ignore the quadratic consistent orientation constraints (4). We describe an algorithm to solve this relaxed problem in Section 3.3. In Section 3.4, we describe heuristics to locally repair the inverted triangles and satisfy Equation (4) without breaking the regularity constraints (3). Finally, we re-optimize the position field in Section 3.5.

3.3. Removing Singularities from the Position Field

Equation (5) is an integer programming problem (ILP), but we are able to approximate it as a *minimum cost network flow* (MCF) problem, which is our key contribution. Efficient algorithms such as the network simplex method [Orl97] can be applied to find its optimal solution in polynomial time.

Minimum Cost Flow. We first show that we can reduce the following class of ILP problems to MCF problems. Given $\mathbf{A} \in \mathbb{R}^{n\times m}$, $\mathbf{b} \in \mathbb{R}^n$, $\varsigma_i \in \mathbb{Z}$, and $\omega \in \mathbb{R}^m_+$, an ILP problem can be written as

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{Z}^m}{\text{minimize}} \quad & \omega^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & 0 \leq x_i \leq \varsigma_i \quad \forall i \in \{1,2,\ldots,m\}. \end{aligned} \tag{6}$$

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & x_1 + 2x_2 + 3x_3 + x_4 + 4x_5 \\ \text{subject to} \quad & x_2 - x_3 = -5 \\ & x_5 - x_2 - x_1 = -2 \\ & x_3 + x_4 - x_5 = 4 \\ & x_1 - x_4 = 3 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2 \leq 2 \\ & 0 \leq x_3 \leq 2 \\ & 0 \leq x_4 \leq 3 \\ & 0 \leq x_5 \leq 1 \\ & x_i \in \mathbb{Z} \quad \forall i \end{aligned}$$

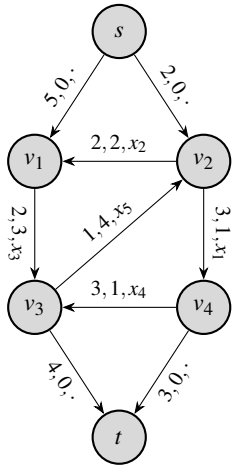


Figure 4: *Left: an integer linear program in which all the variables are balanced. Right: the corresponding minimum cost network flow problem. The three symbols of the label on each edge represent the capacity, the cost, and the corresponding ILP variable, respectively. Under the full flow condition, these problems are equivalent.*

Suppose that each column of the matrix A contains one $+1$, one -1, and $n-2$ zeros. In other words, each variable x_i must appear exactly twice in the equality constraints, once with coefficient $+1$ and once with coefficient -1. For convenience, we say that a variable is *balanced* if it satisfies this requirement. We claim that this ILP can be reduced to an MCF problem if all the variables are balanced.

We begin by constructing a network graph $G = (V,E,c,w,s,t)$, in which $c : E \to \mathbb{R}$ is the capacity of each edge, $w : E \to \mathbb{R}$ is the cost of each edge, and s and t are the source and the sink nodes of the network, respectively. For $i \in \{1,2,\ldots,n\}$, we add a node v_i to V corresponding to the ith equality constraint $\mathbf{A}_i\mathbf{x} = b_i$; we also add s and t to V. Then, for each variable x_k and for $A_{ik} = -1$ and $A_{jk} = +1$, we create an edge e_{ij} from node v_i to v_j, which will carry a flow $f_{ij} = x_k$. The capacity of e_{ij} is $c_{ij} = \varsigma_k$ and the cost of e_{ij} is $w_{ij} = \omega_k$. Finally, for each constraint $\mathbf{A}_i\mathbf{x} = b_i$, if $b_i > 0$ we create a zero-cost edge from node v_i to sink t with capacity $c_{it} = b_i$, and if $b_i < 0$ we create a zero-cost edge from source s to node v_i with capacity $c_{si} = -b_i$. Figure 4 shows an example of an ILP and its equivalent MCF problem.

Solving the ILP in Equation (6) is equivalent to finding the minimum cost flow $f : E \to \mathbb{Z}$ of G under the *full flow condition*, in which flows of every outgoing edge from s and every incoming edge to t are required to reach their full capacity, i.e., $f_{si} = c_{si}$ and $f_{it} = c_{it}$ for every node i. After solving the MCF problem, we solve the ILP by setting each x_k equal to the corresponding f_{ij}. These two problems are equivalent because of the *flow conservation* condition: for each node, the sum of the flows entering the node is equal to the sum of the flows leaving it; that is, $\sum_u f_{uw} = \sum_v f_{wv}$ for all $w \in V$. We have the following observations.

- The objective functions of the ILP and MCF are the same.
- Each equality constraint with $b_i = 0$ in the ILP is equivalent to the flow conservation condition at the corresponding node in the MCF network.

- For each equality constraint with $b_i > 0$ in the ILP, it is equivalent to the flow conservation condition at the corresponding node if the edge capacity from the node to t is fully occupied, i.e., $f_{it} = c_{it}$. For $b_i < 0$, the equivalence holds if the capacity of the edge from s is fully occupied, i.e., $f_{si} = c_{si}$.

Consider the example in Figure 4. The flow conservation condition at v_1 requires that $f_{s1} + f_{21} = f_{13}$. This is equivalent to the constraint $x_2 - x_3 = -5$ in the ILP when edge (s, v_1) is at full capacity: $f_{s1} = c_{s1} = 5$. Therefore, a flow that satisfies the full flow condition corresponds to a feasible solution of Equation (6), while the identical objective functions imply that a solution optimal for one is optimal for the other.

Let $\mathbf{t}^*$ be a solution obtained from the algorithm of Jakob et al. that does not respect all the constraints from Section 3.2. We compute $\mathbf{d}^*$ from $\mathbf{t}^*$. Our first goal is to satisfy the regularity constraints (3) with a minimum change to $\mathbf{d}^*$ per objective (5). When we enforce the constraints, it will cause a change $\delta\mathbf{d} = \mathbf{d} - \mathbf{d}^*$ to the integer offsets. To satisfy the requirement (6) that the variables are constrained to be nonnegative, we split $\delta\mathbf{d}_e$ into two nonnegative variables $\delta\mathbf{d}_e = \delta\mathbf{d}_e^+ - \delta\mathbf{d}_e^-$. Our aim is to choose $\delta\mathbf{d}_e^+$ and $\delta\mathbf{d}_e^-$ for all $e \in \mathfrak{E}$ to

$$\text{minimize} \quad \sum_{e\in\mathfrak{E}} \delta\mathbf{d}_e^+ + \delta\mathbf{d}_e^- \tag{7}$$

$$\begin{aligned}
\text{subject to} \quad & \mathcal{R}^u_{uv}\mathbf{d}_{uv} + \mathcal{R}^u_{vw}\mathbf{d}_{vw} + \mathcal{R}^u_{uw}\mathbf{d}_{wu} = \mathbf{0} \quad \forall \Delta_{uvw} \in \mathcal{F}, \quad (8)\\
& \mathbf{d}_e = \mathbf{d}_e^* + \delta\mathbf{d}_e^+ - \delta\mathbf{d}_e^- \quad \forall e \in \mathfrak{E},\\
& \mathbf{0} \le \delta\mathbf{d}_e^+ \le \mathbf{H}_e^+ \quad \forall e \in \mathfrak{E},\\
& \mathbf{0} \le \delta\mathbf{d}_e^- \le \mathbf{H}_e^- \quad \forall e \in \mathfrak{E},\\
& \delta\mathbf{d}_e^+, \delta\mathbf{d}_e^- \in \mathbb{Z}^2 \quad \forall e \in \mathfrak{E},
\end{aligned}$$

where $\mathbf{H}^+_{uv}$ and $\mathbf{H}^-_{uv}$ are the maximum allowed modification for $\mathbf{d}$ in the positive direction and the negative direction, respectively. Initially, we set H^+_{uv} and H^-_{uv} so that the $\mathbf{d}_{uv} \in [-2, 2]^2$. Then we repeatedly increase this limit until the corresponding MCF problem is feasible. The reason we want to keep $|\mathbf{d}_{uv}|_\infty$ as small as possible is that if there exists a long edge in the integer offsets, we need to subdivide it (later on in this section), which creates more vertices. This ILP has the form of Equation (6) and satisfies most of the prerequisites to be cast as an MCF problem, but it does not satisfy the balance condition.

Balancing Variables. To use the MCF formulation to solve Equation (7), we need to balance the variables in Equation (8), making each variable appear twice with opposite signs. For a manifold triangle mesh, each edge adjoins exactly two triangles except for the edge at the boundary. To make each variable appear exactly twice in Equation (8), we simply fix $\delta\mathbf{d}_e$ to be a constant for each $e \in \mathcal{E}$ at the boundary.

Some variables may not be balanced initially, but we can balance them by a simple 2D rotation. Suppose $\mathbf{d}_e$ appears in two regularity constraints in Equation (8) with coefficients $\mathbf{R}_2(k_1)$ and $\mathbf{R}_2(k_2)$. We can rotate the second equation by multiplying it by $\mathbf{R}_2(k_1 - k_2 + 2)$ so that the second coefficient becomes $-\mathbf{R}_2(k_1)$ and balances the two signs of $\mathbf{d}_e$. Figure 5 shows an example containing two triangles, in which the frame of each vertex is marked.

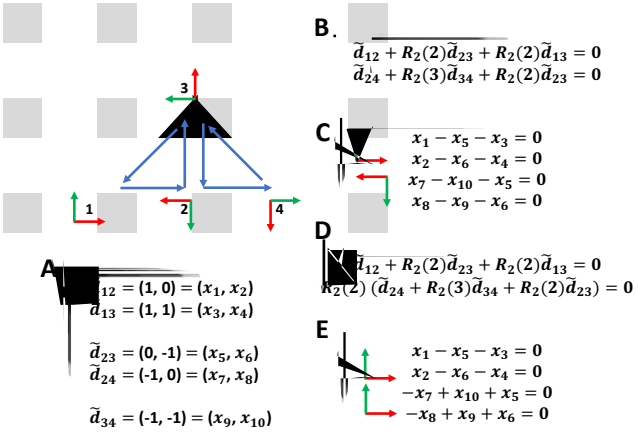


Figure 5: *This example shows why variables may not be balanced. Panel A shows the ten integer variables corresponding to the five edges of two adjacent triangle elements. Their regularity constraints are shown on Panel B in the vector form and Panel C in the scalar form, in which variables x_5 and x_6 are not balanced. By rotating the second equation of the right triangle through 180° in Panel D, x_5 and x_6 become balanced as shown in Panel E.*

Panel A shows the variables $\mathbf{d}$. We show the regularity constraints, i.e., Equation (8), in Panel B as the vector form and in Panel C as the scalar form. The variable $\mathbf{d}_{23}$ (or variables x_5 and x_6) has two negative signs and thus is not balanced. By rotating the second equation 180° as Panel D illustrates, we are able to balance $\mathbf{d}_{23}$.

To balance as many variables as possible, we arbitrarily pick a triangle as the reference (the root of the search tree), and do a breadth first search (BFS) to visit other triangles. For each pair of equations corresponding to the two adjacent triangles along the search tree, we rotate the second equation to balance their shared variables. For the edges that are not on the BFS search tree, the balanced condition is not guaranteed. We fix those unbalanced variables as constants so that all mutable variables are balanced.

Feasibility Condition. Fixing unbalanced variables to $\mathbf{d}^*$ as constants may lead to an infeasible ILP problem. One necessary condition for feasibility is $\sum_{i=1}^n b_i = 0$ for $\mathbf{b}$ in Equation (6). To prove this, we add all the equality constraints together $\sum_{i=1}^n \mathbf{A}_i\mathbf{x} = \sum_{i=1}^n b_i$, and because each column of $\mathbf{A}$ contains one $+1$, one -1 and $n-2$ zeros, the left hand side is equal to zero, so is $\sum_{i=1}^n b_i$. From the view of MCF, this requires the outbound capacity of the source s equal to the inbound capacity of the sink t.

To guarantee $B := \sum_{i=1}^n b_i = 0$, we apply the following greedy strategy to determine $\delta\mathbf{d}_{uv}$ for unbalanced edges and boundary edges. Initially, we set $\delta\mathbf{d}_{uv} = 0$ for all unbalanced and boundary edges. Next, we randomly select one of these edges and change it in the direction to decrease the magnitude of B. This process is repeated until $B = 0$. This strategy is based on the fact that incrementing/decrementing a variable for a boundary edge changes B by one, whereas incrementing/decrementing a variable for an unbalanced edge changes B by two. Note that $B = 0$ can always be achieved

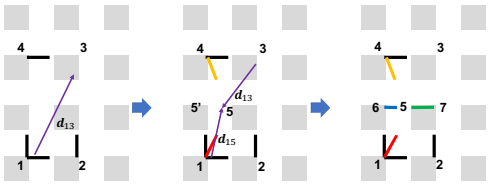


Figure 6: *Subdivision example. At the left, there are initially three long edges $\boldsymbol{d}_{13} = (1,2)$, $\boldsymbol{d}_{14} = (0,2)$ and $\boldsymbol{d}_{23} = (0,2)$. We first subdivide $\boldsymbol{d}_{13}$ as $\boldsymbol{d}_{15} = (0,1)$ and $\boldsymbol{d}_{35} = (-1,-1)$, as shown in the middle. Note that the actual location of v_5 is at $v_{5'}$, but we draw it at v_5 for clarity. Finally, we subdivide $\boldsymbol{d}_{14}$ and then $\boldsymbol{d}_{23}$. As a result, all edges have $\|\boldsymbol{d}\|_\infty \leq 1$.*

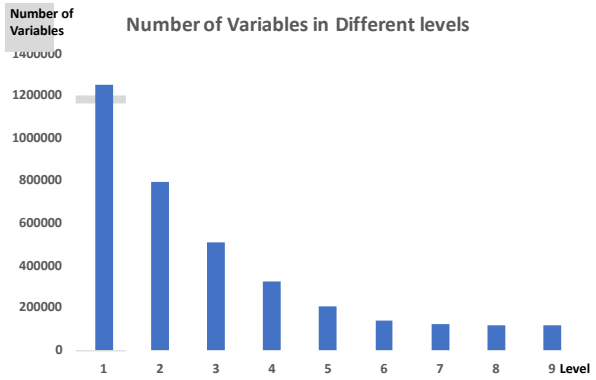


Figure 7: *Number of variables at each level of our multi-resolution structure.*

because we can at least let $\mathbf{d}_{uv} = 0$ for all unbalanced edges and boundary edges (u,v) to satisfy it, i.e., setting $\delta\mathbf{d}_{uv} = -\mathbf{d}^*_{uv}$.

Now, we show the sufficient condition for the feasibility of the ILP considering its equivalent MCF problem. Let $C^- := \sum_u c_{su}$ be the outbound capacity of the source and $C^+ := \sum_v c_{vt}$ be the inbound capacity of the sink. The following theorem states that with proper assumption, the full flow condition in the MCF formulation is always achievable.

Theorem 3.1. *Given a network $G = (V,E,c,w,s,t)$, if $C^+ = C^-$, all the internal vertices $V\backslash\{s,t\}$ are strongly connected (a vertex v_a is strongly connected to v_b if there exist two paths, one from v_a to v_b and another from v_b to v_a), and $c_{uv} \geq C^+$ for all $(u,v) \in E$ in which $u,v \notin \{s,t\}$, then the maximum flow f satisfies the full flow condition.*

Proof. We prove it by contradiction. Given a network $G = (V,E,c,w,s,t)$ where $V\backslash\{s,t\}$ are strongly connected, we assume f does not satisfy the full flow condition. Because the edge capacity is larger than C^+, the flow on any internal edge (u,v) is smaller than the capacity. This means that the connectivity of the residual network of f remains unchanged. Therefore, $V\backslash\{s,t\}$ is still strongly connected in the residual network. Also because $C^+ = C^-$ and the full flow condition is not satisfied, s and t must be connected to $V\backslash\{s,t\}$ in the residual network. So there exists an augmenting path from s to t, which contradicts with the assumption. □

Fortunately, we guarantee the strong connectivity in the MCF network: Because we use BFS to balance variables, any adjacent triangles on the search tree share an edge e with balanced $\mathbf{d}_e$. Therefore as long as the triangle mesh $\mathcal{M}$ is connected, the corresponding nodes of the adjacent triangles in the flow networks are connected. Because BFS reaches all triangles in the mesh, the corresponding nodes in the flow networks are strongly connected. In addition, according to the way we construct the network, we have $C^+ = \sum_{b_i>0} b_i$ and $C^- = -\sum_{b_i<0} b_i$. Thus, $B = 0$ implies $C^+ = C^-$. By using the above theorem, we can guarantee the achievement of a full flow by computing the maximum flow once we have $c_{uv} \geq C^+$ for all edges $(u,v) \in E$, or large enough $\mathbf{H}_{uv}$ in the ILP (Equation 8).

Summary. To remove position singularities, we first BFS triangles on the mesh to rotate the corresponding equations to balance variables. We fix boundary and unbalanced variables, and randomly modify them to achieve $\sum_{i=1}^n b_i = 0$. Then, we build an equivalent MCF network. We set all edge capacities c_{uv} (or $H^-_{uv} = H^+_{uv}$ in the ILP) so that $\mathbf{d}_{uv} \in [-2,2]^2$ and run an MCF solver. If it returns infeasible, we retain the flow value f for each edge and repeatedly increase the capacity until the problem is feasible.

Multi-Resolution MCF. Under many scenarios, the required density of the quad mesh is significantly lower than the density of the input triangle mesh. This means that most of $\mathbf{d}$ will finally be zero. Therefore, we are able to accelerate the MCF algorithm with a multi-resolution structure.

For each resolution, we build a coarser network by removing approximately half of zero edges ($\mathbf{d}^*_e = \mathbf{0}$) to reduce the number of variables. Consider a general case where the regularity constraints are satisfied for two equations corresponding to Δ_{abc} and Δ_{aef} sharing the edge a. The equation for Δ_{abc} can be written

$$R_a\mathbf{d}_a + R_b\mathbf{d}_b + R_c\mathbf{d}_c = 0.$$

When $\mathbf{d}_a = \mathbf{0}$, we can replace $\mathbf{d}_b$ by $\mathbf{d}_b = -R_b^{-1}R_c\mathbf{d}_c$. Therefore, we can simplify the set of regularity constraints by a collapse operation: Remove the variable $\mathbf{d}_a$ and the two constraints for Δ_{abc} and Δ_{aef}, and replace $\mathbf{d}_b$ and $\mathbf{d}_e$ with $-R_b^{-1}R_c\mathbf{d}_c$ and $-R_e^{-1}R_f\mathbf{d}_f$.

To build a hierarchy structure, we scan all the edges with $\mathbf{d}_e = \mathbf{0}$ at each resolution and collapse them if two corresponding equality constraints of the zero edge are already satisfied. Once we remove the variable for one edge, we do not allow the collapse operation on adjacent edges at the same level. Figure 7 shows the number of variables for each level with our collapse operation. We first compute the flow at the lowest resolution, and then propagate the change of variables to the higher resolution until the full flow condition is satisfied. We keep the edge capacity be two in the multi-resolution solver and only increase it if the full flow condition cannot be satisfied in the highest resolution. In practice, we usually can solve most singularities at the lowest resolution, where the number of variables is less than one-tenth of the number of variables in the original (highest) resolution.

In the beginning, we use the network simplex implementation

from the LEMON library [DJK11] to solve the MCF at the lowest resolution. This resolves most of singularities with a minimum change of $\mathbf{d}$. For further trials with large networks, we approximate the MCF problem as a maximum flow problem for efficiency. We use the Boykov–Kolmogorov algorithm [BK04] if the number of remaining unsatisfied regularity constraints is greater than 10, and use the Edmonds–Karp algorithm [EK72] otherwise. The maximum flow approximation sacrifices some optimality of Equation (7), but it greatly improves the efficiency and works well in practice, as Section 4.4 will demonstrate.

Subdivision on Integer Offsets. After the Multi-Resolution MCF, $\|\mathbf{d}_{uv}\|_\infty$ might be greater than 1 in order to satisfy the full flow condition. This will complicate the mesh extraction stage later. Therefore, we subdivide the integer offsets of those long edges by adding a midpoint and two new edges. Because we require the lengths of the edges to be integers, the two subdivided edges of $\mathbf{d}_{uv}$ are computed as $\mathbf{d}_{uv}$ div 2 and $(-\mathbf{d}_{uv}+\mathbf{d}_{uv})$ div 2. Figure 6 shows an example: the long edge $\mathbf{d}_{13}=(1,2)$ is divided into $\mathbf{d}_{15}=(0,1)$ and $\mathbf{d}_{35}=(-1,-1)$, then $\mathbf{d}_{14}$ and $\mathbf{d}_{23}$ are subdivided.

3.4. Eliminating Inverted Normals

To enforce the consistent orientation constraint in Equation (4), we employ a two-stage method. First, a greedy algorithm scans the triangle elements in the mesh and iteratively shrinks the inverted triangles. After that, we locally model Equations (3) and (4) as a Boolean satisfiability problem and try to resolve the remaining inversions with a SAT solver. Using this strategy, we are able to generate inversion-free quad mesh for many testing data, but this is not always guaranteed due to the NP-completeness of the Boolean satisfiability problem.

Greedy Method. One way to shrink an inverted triangle is to move one of its vertices to another. To move a vertex from u to v, we set $\mathbf{d}_{uv}=0$ and modify u's adjacent edges $\mathbf{d}_{uw}$ for all $(u,w)\in\mathfrak{E}$ accordingly to maintain the regularity constraints in Equation (3). This operation locally changes the adjacent edges, and thus the area of the adjacent triangles. We scan all the edges of inverted triangles, and shrink an edge only if it does not produce long edges ($\|d^*\|>1$) and reduces the total inverted area. Our algorithm terminates until no further movement is feasible. This greedy algorithm can efficiently remove most of the inverted triangles. We observe that the remaining inversions are normally located near the orientation singularities.

Reduction to SAT. To solve the remaining tough inversions, we model it as a Boolean satisfiability (SAT) problem. An SAT problem aims at finding an assignment to satisfy a given Boolean equation. Although the SAT problem has been proven to be NP-complete, researchers have built efficient SAT solvers based on sophisticated heuristics that are able to solve practical problems with tens of thousands of variables. To turn the constraints in Equations (3) and (4) to a Boolean equation, we represent each integer vector variable $\mathbf{d}_{uv}$ with nine Boolean variables $D^{\mathbf{x}}_{uv}$, where $\mathbf{x}\in\mathbf{S}:=\{-1,0,+1\}^2$ represents the nine possible values of the integer vector as these values are guaranteed to be in $\{-1,0,+1\}$ after the subdivision stage. The relationship between the integer variable and Boolean variable is

$$\mathbf{d}_{uv}=\mathbf{x}\Longleftrightarrow D^{\mathbf{x}}_{uv}=\mathbf{true}$$

for all $(u,v)\in\mathfrak{E}$ and $\mathbf{x}\in\mathbf{S}$. Then we turn the constraints in Equations (3) and (4) into Boolean expressions in conjunctive normal form (CNF). CNF is a list of *clauses* that need to be satisfied simultaneously, and each clause contains a list of variables connected by OR operators. For each $\Delta_{uvw}\in\mathcal{F}$, we add the following clauses to our SAT solver:

$$\begin{aligned}\neg D^{\mathbf{x}}_{uv}\vee\neg D^{\mathbf{y}}_{vw}\vee\neg D^{\mathbf{z}}_{wu} \quad & \forall\mathbf{x},\mathbf{y},\mathbf{z}\in\mathbf{S}:\mathbf{x}+\mathbf{y}+\mathbf{z}\neq\mathbf{0},\\ \neg D^{\mathbf{x}}_{uv}\vee\neg D^{\mathbf{y}}_{uw} \quad & \forall\mathbf{x},\mathbf{y}\in\mathbf{S}:\det[\mathbf{x},\mathbf{y}]\leq 0.\end{aligned}$$

The first Boolean equation enforces the regularity constraint and the second Boolean equation enforces the consistent orientation constraint. The above representation is more for notation It is a little bit redundant because it needs 9 Boolean variable per integer offset. We can reduce that to 6 Boolean variables by splitting the dimension of $\mathbf{x}$ for $D^{\mathbf{x}}_{uv}$ so that $D^{\mathbf{x}}_{uv}=:D^{x_1}_{uv}\wedge D^{x_2}_{uv}$, where $\mathbf{x}=(x_1,x_2)$.

In practice, we find that most triangle inversion problems can be solved locally. That is, it is sufficient to only change the geometry of the nearby regions of the inverted triangles. So we iteratively increase the diameter of the mutable $\mathbf{d}$ until the resulting SAT problem is feasible or the SAT solver times out. In our implementation, we use the open source SAT solver [LGPC16].

3.5. Updating the Continuous Positions

To make the real-valued variables of the position field consistent with our regularized, inversion-free integer offsets $\mathbf{d}^\star$, we re-optimize $\mathbf{p}$ by minimizing the sum of squared differences between the actual and the desired distances,

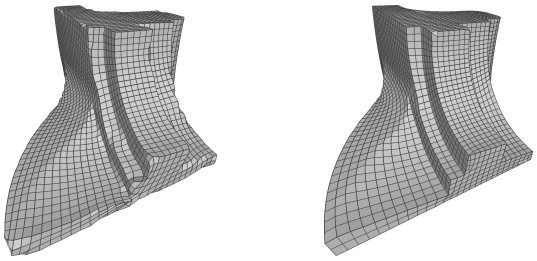
$$E_p(\mathbf{p})=\sum_{(u,v)\in\mathcal{E}}||\mathbf{p}_v-\mathbf{p}_u-\rho(\mathbf{O}_u\mathbf{d}^\star_{uv})||_2^2, \tag{9}$$

where $\rho(\mathbf{O}_u\mathbf{d}^\star_{uv})$ is the desired 3D translation from vertex u to v. As with the method in Section 3.1, for each vertex $u\in\mathcal{V}$, we restrict $\mathbf{p}_u$ to lie on u's tangent plane. This is a linear least-squares problem, easily and efficiently solvable.

Our re-optimization can be made to preserve sharp edges in the triangle mesh. We call an edge in $\mathcal{E}$ "sharp" if the angle between the two adjoining triangles' normals exceeds a user-specified threshold. If vertices on sharp edges are permitted to move in the tangent plane, sharp features may be lost, as Figure 8(a) shows. Thus, for a vertex $v\in\mathcal{V}$ on a sharp edge, we further constrain $\mathbf{p}_v$ to move only along the edge's affine hull. This constraint is easily incorporated into the linear least-squares problem. Figure 8(b) shows that this constraint yields better results.

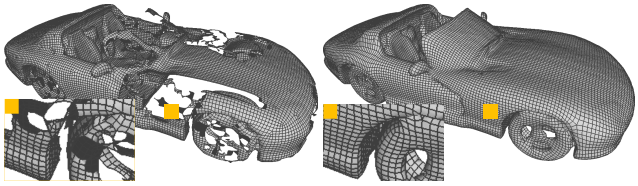
3.6. Quad Mesh Extraction

Our quad mesh extraction algorithm is simpler than that of Jakob et al. [JTPSH15]. Because of the subdivision routine described in Section 3.3, our position field (specified by $\mathbf{p}^\star$ and $\mathbf{d}^\star$) does not have large integer translations (long edges); specifically, $|\mathbf{d}^\star|_\infty\leq 1$.



(a) Only tangential constraints (b) With sharp edge constraints

Figure 8: *(a) The tangent plane restriction does not suffice to preserve sharp edges. (b) Line constraints preserve sharp features better.*



(a) Instant Meshes (b) QuadriFlow

Figure 9: *(a) Example where a nonmanifold input triangulation leads to an Instant Mesh with holes. (b) Because QuadriMesh removes inverted triangles from the position field representation, it produces a manifold mesh with the correct topology.*

Moreover, as our position field has no singularities, each non-degenerate face is a right-angled isosceles triangle with one hypotenuse and two legs, and two triangles sharing the hypotenuse form a quad. Hence our mesh extraction algorithm is straightforward: first, collapse all zero-length edges (e for which $\mathbf{d}_e^\star = \mathbf{0}$). Then for each hypotenuse edge ($|\mathbf{d}_e^\star|_1 = 2$), extract the quad for its two neighboring triangles. Because we enforce consistent orientation constraints, the quad mesh thus extracted is nearly always manifold.

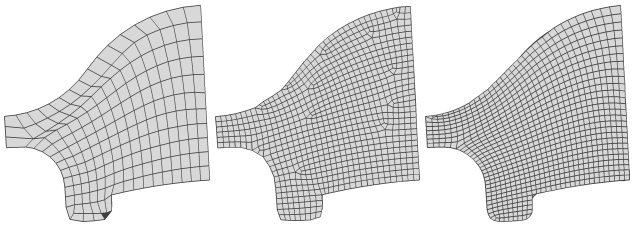
By comparison, the mesh extraction method for Instant Meshes [JTPSH15] must cope with corner cases such as long edges, inverted triangles, and position field singularities. Figure 9 gives an example for which their implementation has difficulty with a complex geometry, whereas ours produces the correct topology.

4. Evaluation

Here we evaluate the quality, efficiency, and robustness of our method. We compare our mesh quality with prior methods; we thank the creators of those methods for sharing their meshes with us. We also compare implementations of several methods on 110 challenging car geometries from ShapeNet [CFG*15, HSG18].

4.1. Mesh Quality

We compare our meshes with meshes generated by several other state-of-the-art methods. Table 2 lists the names of the models and properties related to the quality of the quad mesh: angle distortion, area distortion, and the number of singularities. The best numbers are in boldface.



(a) IGM (b) Instant Meshes (c) QuadriFlow

Figure 10: *Quadrangulation of the model Fandisk using IGM, Instant Meshes, and QuadriFlow (our method). Our mesh has less angle distortion than IGM, and fewer vertices of irregular valence than Instant Meshes.*

Angle Distortion is measured as $\sqrt{\frac{1}{N}\sum_i(\theta_i - 90°)^2}$, where the sum is over all the angles in the mesh and N is their number. Our meshes have less angle distortion than IGM [BCE*13], and are comparable with Instant Meshes. As Figure 10 shows, the bottom of the Fandisk mesh produced by IGM has large distortion, while ours does not. Instant Meshes usually introduces many unnecessary singularities, especially for irregular shapes.

Area Distortion is the standard deviation of the areas of the quadrilateral faces. As reported by Jakob et al., additional position singularities may alleviate distortion and improve the isotropy of the quad mesh. Our algorithm achieves comparable isotropy without additional position singularities, as Figure 1 shows.

Table 2 suggests that Mixed-Integer Quadrangulation (MIQ) [BZK09], Integer-Grid Map (IGM) [BCE*13], Instant Meshes [JTPSH15], and our QuadriFlow are the four best methods to discuss in detail. QuadriFlow meshes have slightly larger distortions than Instant Meshes, which is a reasonable price to pay for the dramatic reduction in the number of singularities—in practice, we are able to remove all the singularities from the position field. For the Buddha and Kitten100K models, QuadriFlow outperforms all other methods for singularities, but MIQ and IGM produce fewer singularities for other models.

The cross fields and resolutions of Instant Meshes and QuadriFlow meshes are exactly the same. The comparison meshes generated by other state-of-the-art methods used different cross fields and mesh resolutions. We subdivide their meshes to our resolution, followed by a Laplacian smoothing step. QuadriFlow meshes exhibit angle distortion similar to these methods, but it produces less area distortion, probably due to different cross fields or our MCF problem being easier to solve than the mixed integer programming problems. For all the models in Table 2, QuadriFlow is able to generate an inversion-free integer offset $\mathbf{d}^\star$ after enforcing the consistent orientation constraints.

Figure 11 plots the number of singularities in meshes of the Knot1 model (illustrated in Figure 12) with respect to the target number of vertices. The blue bars count the number of orientation singularities for QuadriFlow, which produces no position singular-

Table 2: *Comparison of different methods. The best scores appear in boldface. QuadriFlow has slightly larger angle and area distortions than Instant Meshes, but it is usually better than the global methods MIQ and IGM. In terms of the number of singularities, QuadriFlow is competitive with these global methods, and it is significantly better than Instant Meshes.*

Method	Angle Distortion	Area Distortion	# of Sings	Method	Angle Distortion	Area Distortion	# of Sings
David [ACSD*03]				Pig [ACSD*03]			
Alliez et al.	23.6	0.74	10310	Alliez et al.	17.7	0.61	436
Instant Meshes	**10.9**	**0.22**	2708	Instant Meshes	**8.4**	0.19	148
QuadriFlow	14.4	0.26	**212**	QuadriFlow	10.0	**0.18**	**38**
Fandisk [MK06]				RockerArm [MK06]			
Marinov & Kobbelt	18.0	0.63	59	Marinov & Kobbelt	14.9	0.43	117
Instant Meshes	**7.14**	**0.18**	117	Instant Meshes	**6.9**	**0.15**	132
QuadriFlow	7.65	0.38	**38**	QuadriFlow	10.9	0.20	**52**
Bunny [TPC*10]				Gargoyle [TPC*10]			
Tarini et al.	15.8	0.24	3438	Tarini et al.	17.4	0.23	4283
Instant Meshes	**7.2**	**0.15**	351	Instant Meshes	**9.85**	**0.20**	659
QuadriFlow	10.4	0.19	**56**	QuadriFlow	16.5	0.27	**218**
Omotondo [TPC*10]				Rampant [TPC*10]			
Tarini et al.	17.4	0.24	3903	Tarini et al.	17.4	0.31	3745
Instant Meshes	**7.8**	**0.17**	367	Instant Meshes	**8.3**	**0.18**	455
QuadriFlow	14.3	0.23	**80**	QuadriFlow	12.2	0.23	**158**
Fandisk [BZK09]				Fertility [BZK09]			
MIQ	8.21	0.39	**30**	MIQ	8.59	0.26	**48**
Instant Meshes	**7.14**	**0.20**	68	Instant Meshes	**7.09**	**0.15**	256
QuadriFlow	7.65	0.22	38	QuadriFlow	7.78	0.16	70
RockerArm [BZK09]				Buddha [BCE*13]			
MIQ	**5.5**	0.30	**36**	IGM	12.0	0.28	108
Instant Meshes	7.6	0.19	132	Instant Meshes	**9.3**	**0.20**	301
QuadriFlow	10.9	**0.17**	52	QuadriFlow	11.6	0.22	**92**
Fandisk [BCE*13]				Feline [BCE*13]			
IGM	11.3	0.40	**30**	IGM	17.7	0.44	**110**
Instant Meshes	**7.14**	**0.20**	117	Instant Meshes	**8.11**	**0.18**	592
QuadriFlow	7.65	0.22	38	QuadriFlow	18.0	0.42	158
Hand [BCE*13]				Kitten100K [BCE*13]			
IGM	8.5	0.46	**40**	IGM	7.84	0.48	63
Instant Meshes	**6.46**	0.23	43	Instant Meshes	**6.87**	**0.16**	127
QuadriFlow	7.4	**0.22**	42	QuadriFlow	8.43	0.19	**32**
Bunny [MPZ14]				Gargoyle [MPZ14]			
Myles et al.	13.5	0.25	**30**	Myles et al.	10.7	0.33	328
Instant Meshes	**7.2**	**0.15**	351	Instant Meshes	**9.85**	**0.20**	659
QuadriFlow	10.4	0.19	56	QuadriFlow	16.5	0.27	**218**
Kitten100K [MPZ14]				Pig [MPZ14]			
Myles et al.	13.1	0.46	75	Myles et al.	14.0	0.25	55
Instant Meshes	**6.87**	**0.16**	127	Instant Meshes	**8.4**	0.19	148
QuadriFlow	8.43	0.19	**32**	QuadriFlow	10.0	**0.18**	**38**

ities. The orange bars count the number of orientation singularities for Instant Meshes, and the green bars count the sum of orientation and position singularities for Instant Meshes. The number of position singularities in Instant Meshes increases linearly with the mesh density, which is one of its weaknesses.

Because we use the extrinsic formulations from Instant Meshes, our mesh edges align with shape features better than IGM's, as Figure 12 shows.

We compare Instant Meshes, Integer-Grid Map, and QuadriFlow in a subclass of ShapeNet with 110 challenging car models that the implementation of MIQ [BZK09] in `libigl` cannot handle. As before, the resolutions and the cross fields of the Instant Meshes and QuadriFlow meshes remain identical to each other, whereas we subdivide and smooth the Integer-Grid Map meshes to the same resolution. We do not have room to list all the models, so Table 3 shows only the percentage of meshes for which QuadriFlow outperforms Instant Meshes or IGM according to the specified measure. Note that there are models for which these two methods cannot produce reasonable meshes (see Section 4.2); though QuadriFlow can mesh all the models well, we omit from the comparison models for which one of the other methods produce conspicuous visual artifacts. Table 3 indicates that QuadriFlow meshes exhibit less distortion than IGM, but more than Instant Meshes. For most models, we attain the minimum number of singularities. Figure 15 illustrates

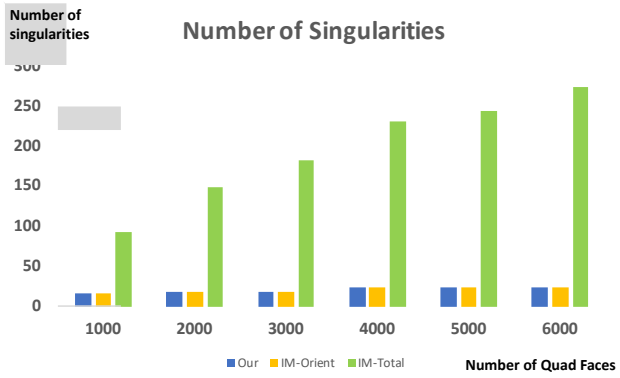
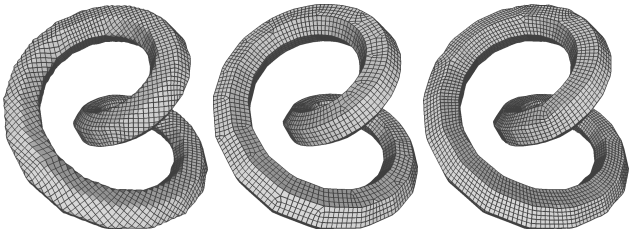


Figure 11: *Number of singularities in meshes of Knot1 as a function of the target number of vertices. Blue bars count the orientation singularities in QuadriFlow meshes; there are no position singularities. Orange bars count the orientation singularities in Instant Meshes, and green bars count their total orientation and position singularities.*



(a) IGM (b) Instant Meshes (c) QuadriFlow

Figure 12: *Surface quadrangulations of the model Knot1 using Integer-Grid Map (IGM), Instant Meshes, and QuadriFlow. As we borrow the extrinsic energy formulation from Instant Meshes, our mesh edges are aligned with shape features, unlike IGM's.*

several of our meshes, with the models kindly supplied by Jakob et al. or ShapeNet.

4.2. Robustness

To test the robustness of our algorithm, we ran QuadriFlow on 17,791 watertight triangle manifolds generated by Huang, Su, and Guibas [HSG18] from the ShapeNet repository [CFG*15], as well as the models provided by Jakob et al. [JTPSH15]. For every model, QuadriFlow always generates a manifold quadrilateral mesh and removes all the position singularities, and the chamfer distances to the original meshes are always less than 5% of the average edge length in the quad mesh. This validates the robustness of our algorithm. We are able to preserve the watertightness of every model provided by Jakob et al., but not for about 20% of the ShapeNet models, because the SAT algorithm cannot eliminate every inverted triangle. By contrast, MIQ [BZK09] as implemented in `libigl` [JP*17] fails on most of these models.

Recall from Table 3 that we tested Instant Meshes and IGM on 110 watertight car manifolds from ShapeNet. IGM can produce high-quality meshes for 62 of them. Instant Meshes is able to generate quad meshes for all of them, but 52 of those contain large holes. QuadriFlow succeeds on all of them. We provide the models and meshes in the supplementary material.

Table 3: *Comparisons tabulating the percentage of 110 ShapeNet car models for which QuadriFlow outperforms Integer-Grid Maps (IGM) or Instant Meshes based on the angle distortion, the area distortion, or the number of singularities. We exclude models on which IGM or Instant Meshes fails to produce a usable mesh. Instant Meshes have the least distortion, whereas QuadriFlow meshes have the fewest singularities.*

Method	Angle	Area	# of sings
QuadriFlow vs. IGM	59%	100%	97%
QuadriFlow vs. Instant Meshes	9%	5%	100%

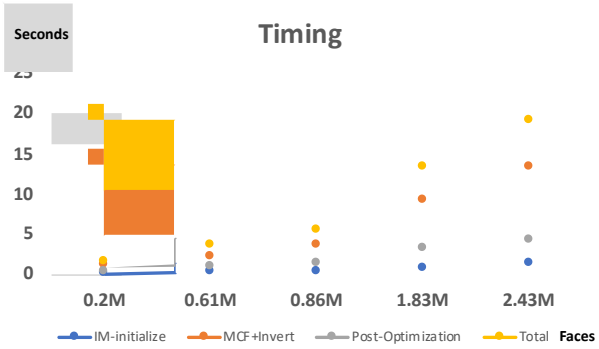
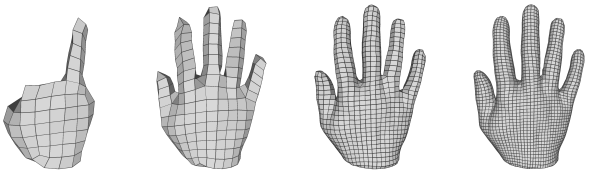


Figure 13: *QuadriFlow running times on the Hand model as a function of the number of faces of the input triangulation. (We subdivide the Hand model to the desired number of faces in advance.) We plot the initial Instant Meshes time, the time to enforce constraints, the time for post-processing, and the total time.*

4.3. Efficiency

In Figure 13, we chart running times of several stages of QuadriFlow as a function of the number of input triangular faces. The input is the Hand model from IGM [BCE*13], subdivided to obtain a suitable number of faces. We implemented Instant Meshes using CUDA with a GTX 1070 GPU, and ran our implementation on a 2.4 GHz CPU with a single thread. Our implementation has speed comparable to the fastest existing method [ESCK16], which runs on a decimated mesh and maps back to the original resolution. They report 5.7 seconds to mesh a model with 0.84 million faces, while directly processing it with a state-of-the-art global method [ECBK14] takes 161 seconds. We take only 5 seconds to process 0.86 million faces, and 20 seconds for 2.43 million faces.

In our experiments, the cost of MIQ [BZK09] as implemented in `libigl` varies a lot for different models. It takes over two minutes to process 14,000 faces, and more than two hours for 100,000 faces for the Gargoyle model [TPC*10]. On our 110-car dataset, IGM takes 50 to 600 seconds to process each model, while our method meshes each model in at most 10 seconds.



131 vertices 344 vertices 1486 vertices 3365 vertices

Figure 14: *A limitation of our method for coarse mesh generation. As we approximate an MIP problem as a minimum cost flow problem, geometric details can be lost when the target mesh density is low.*

4.4. Methodology

Recall that our algorithm introduces an ILP approximation of an MIP problem, then formulates it as an MCF problem, which is also approximate because of the need to fix some integer offsets to balance the variables. Here we evaluate the influence of these approximations on the mesh quality.

ILP Approximation. Instead of jointly optimizing the continuous energy and integer constraints with MIP, we approximate the MIP problem with an ILP problem. Because we do not directly optimize the energy, the ILP solution might not obtain the best energy. This can cause a loss of geometric details when the mesh is coarse. Figure 14 shows the QuadriFlow meshes for the Hand model with different choices of mesh density. At the coarsest resolution, QuadriFlow loses four fingers.

MCF Approximation. To solve the ILP problem, we further approximate it as an MCF problem by fixing some integer offsets to balance the variables. To test how such an approximation affects the mesh, we experimented on the Hand model with a target resolution of 3,365 vertices. We randomly picked ten different starting triangles for the BFS algorithm. We find that the percentage of fixed variables ranges from 0.6% to 0.7%, which is small. The gap between the worst and the best angle distortion or area distortion is less than 2% of the median score. Thus we judge the influence of the MCF approximation to be small and acceptable.

Comparison of Integer Solvers. To justify the effectiveness of our network flow formulation and our multi-resolution framework, we performed experiments with different integer optimization algorithms on two test examples. Their running times and distortion metrics appear in Table 4. We tested five different algorithms. `MF` is the Boykov–Kolmogorov algorithm that solves the maximum flow problem. `MF_MR` is a multi-resolution version of `MF`. `MCF` is the network simplex algorithm from the LEMON library, which solves the minimum cost flow problem. `MCF_MR` is a multi-resolution version of `MCF`, in which we first solve the lowest resolution with the network simplex algorithm, and then solve the highest resolution with `MF_MR`. Lastly, `ILP` uses Gurobi Optimization [Gur16] to solve Equation (5) as an integer linear program.

From the table, we see that multi-resolution can greatly shorten the running times. Furthermore, the network flow algorithms are far more efficient and stable than the integer linear programming algorithms provided by Gurobi, as the former are more specialized whereas ILP is NP-hard. To our surprise, maximum flow algorithms perform nearly as well as minimum cost flow algorithms as measured by the distortion metrics. Perhaps this is because many maximum flow algorithms operate by repeatedly finding the shortest augmenting path, which tends to keep the L_1 norm of Expression (5) small.

Table 4: *Comparison of multiple methods for integer optimization. We show the running times, average angle distortions, and average area distortions on two test examples. The number 900 or 1,500 represents the specified edge density. MF, MCF, MR, and ILP stand for maximum flow, minimum cost flow, multi-resolution, and integer linear programming, respectively.*

Mesh & Algorithm	Time	Angle error	Area error
Hand_900_MF	0.85	**11.195277**	0.272820
Hand_900_MF_MR	**0.09**	12.695140	**0.237884**
Hand_900_MCF	4.12	12.555485	0.294125
Hand_900_MCF_MR	0.11	13.011465	0.263241
Hand_900_ILP	280.00	12.555485	0.294125
Hand_1500_MF	1.76	9.387929	**0.193454**
Hand_1500_MF_MR	**1.05**	10.391423	0.205469
Hand_1500_MCF	13.41	**8.786778**	0.210081
Hand_1500_MCF_MR	1.09	8.982389	0.220997
Hand_1500_ILP	164.00	**8.786778**	0.210081

5. Conclusion

QuadriFlow solves the global problem of removing position singularities much more quickly than other global methods. It is not as fast as Instant Meshes, a purely local method, but it produces many fewer singularities than Instant Meshes. QuadriFlow is also quite robust in practice, in the sense that it rarely produces inverted quads or fails to produce a usable mesh.

As a target for future improvement, observe that our minimum cost flow problem ignores the geometric properties of the input mesh. A careful formulation of the cost that takes account of geometric features might further improve the quality. Our method is fully automatic, but it could be augmented with user interaction, supported by applying additional constraints to the MCF problem. We believe that other classic problems can be reformulated as MCF problems to make quadrangulation a more powerful tool.

Acknowledgements

This research was funded in part by the NSF grants CCF-1423560, CCF-1514305, and CRI-1729205, a TUM-IAS Hans Fischer Fellowship, and gifts from Amazon AWS and Autodesk. We thank Olga Diamanti, who provided insight and expertise that greatly assisted this research.

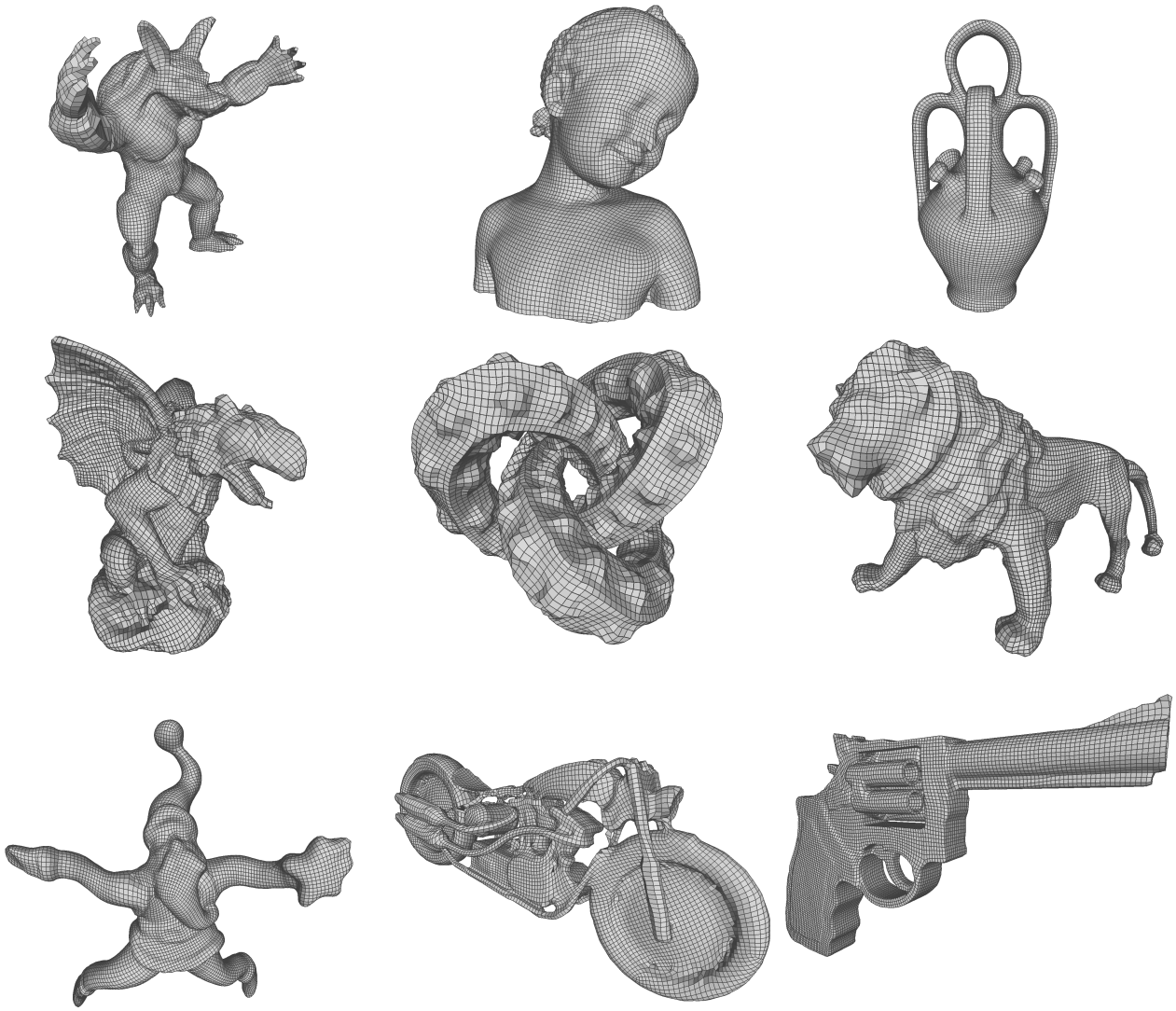


Figure 15: *More meshes generated by QuadriFlow. We thank Jakob et al. [JTPSH15] and ShapeNet [CFG*15, HSG18] for providing the models.*

References

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic polygonal remeshing. In *ACM Transactions on Graphics (TOG)* (2003), vol. 22, ACM, pp. 485–493. 3, 10

[BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 98. 1, 3, 5, 9, 10, 11

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137. 8

[BLP*13] BOMMES D., LÉVY B., PIETRONI N., PUPPO E., SILVA C., TARINI M., ZORIN D.: Quad-mesh generation and processing: A survey. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 51–76. 2

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. In *ACM Transactions On Graphics (TOG)* (2009), vol. 28, ACM, p. 77. 1, 2, 3, 9, 10, 11

[CBK15] CAMPEN M., BOMMES D., KOBBELT L.: Quantized global parametrization. *ACM Transactions on Graphics 34*, 6 (Oct. 2015), 192:1–192:12. 3

[CDS10] CRANE K., DESBRUN M., SCHRÖDER P.: Trivial connections on discrete surfaces. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1525–1533. 3

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012* (2015). 2, 9, 11, 13

[CLW16] CHIEN E., LEVI Z., WEBER O.: Bounded distortion parametrization in the space of metrics. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 215. 3

[CP05] CAZALS F., POUGET M.: Estimating differential quantities using

polynomial fitting of osculating jets. *Computer Aided Geometric Design 22*, 2 (2005), 121–146. 2

[CSM03] COHEN-STEINER D., MORVAN J.-M.: Restricted Delaunay triangulations and normal cycle. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry* (2003), ACM, pp. 312–321. 2

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J. C.: Spectral surface quadrangulation. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 1057–1066. 2

[DJK11] DEZSŐ B., JÜTTNER A., KOVÁCS P.: LEMON—an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science 264*, 5 (2011), 23–45. 8

[DVPSH14] DIAMANTI O., VAXMAN A., PANOZZO D., SORKINE-HORNUNG O.: Designing *n*-PolyVector fields with complex polynomials. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 1–11. 3

[DVPSH15] DIAMANTI O., VAXMAN A., PANOZZO D., SORKINE-HORNUNG O.: Integrable PolyVector fields. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 38. 3

[EBCK13] EBKE H.-C., BOMMES D., CAMPEN M., KOBBELT L.: QEx: Robust quad mesh extraction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 168. 3

[ECBK14] EBKE H.-C., CAMPEN M., BOMMES D., KOBBELT L.: Level-of-detail quad meshing. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 184. 11

[EK72] EDMONDS J., KARP R. M.: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19*, 2 (1972), 248–264. 8

[ESCK16] EBKE H.-C., SCHMIDT P., CAMPEN M., KOBBELT L.: Interactively controlled quad remeshing of high resolution 3D models. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 218. 2, 11

[GLLR11] GURUNG T., LANEY D., LINDSTROM P., ROSSIGNAC J.: SQuad: Compact representation for triangle meshes. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 355–364. 2

[Gur16] GUROBI OPTIMIZATION, INC.: Gurobi optimizer reference manual, 2016. URL: http://www.gurobi.com. 12

[HSG18] HUANG J., SU H., GUIBAS L.: Robust watertight manifold surface generation method for ShapeNet models. *arXiv preprint arXiv:1802.01698* (2018). 9, 11, 13

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 517–526. 2

[JFH*15] JIANG T., FANG X., HUANG J., BAO H., TONG Y., DESBRUN M.: Frame field generation through metric customization. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 40. 3

[JP*17] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2017. http://libigl.github.io/libigl/. 11

[JTPSH15] JAKOB W., TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Instant field-aligned meshes. *ACM Transactions on Graphics 34*, 6 (Oct. 2015), 189:1–189:15. 1, 2, 3, 4, 5, 8, 9, 11, 13

[KCPS13] KNÖPPEL F., CRANE K., PINKALL U., SCHRÖDER P.: Globally optimal direction fields. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 59. 3

[Kle67] KLEIN M.: A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science 14*, 3 (1967), 205–220. 2

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover-surface parameterization using branched coverings. In *Computer Graphics Forum* (2007), vol. 26, Wiley Online Library, pp. 375–384. 1

[Knu95] KNUPP P.: Mesh generation using vector fields. *Journal of Computational Physics 119*, 1 (1995), 142–148. 1

[LGPC16] LIANG J. H., GANESH V., POUPART P., CZARNECKI K.: Learning rate based branching heuristic for SAT solvers. In *International Conference on Theory and Applications of Satisfiability Testing* (2016), Springer, pp. 123–140. 8

[LHJ*14] LING R., HUANG J., JÜTTLER B., SUN F., BAO H., WANG W.: Spectral quadrangulation with feature curve alignment and element size control. *ACM Transactions on Graphics (TOG) 34*, 1 (2014), 11. 2

[LJX*10] LAI Y.-K., JIN M., XIE X., HE Y., PALACIOS J., ZHANG E., HU S.-M., GU X.: Metric-driven RoSy field design and remeshing. *IEEE Transactions on Visualization and Computer Graphics 16*, 1 (2010), 95–108. 2

[LKH08] LAI Y.-K., KOBBELT L., HU S.-M.: An incremental approach to feature aligned quad dominant remeshing. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling* (2008), ACM, pp. 137–145. 3

[LZ14] LEVI Z., ZORIN D.: Strict minimizers for geometric optimization. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 185. 3

[MK06] MARINOV M., KOBBELT L.: A robust two-step procedure for quad-dominant remeshing. In *Computer Graphics Forum* (2006), vol. 25, Wiley Online Library, pp. 537–546. 10

[MPZ14] MYLES A., PIETRONI N., ZORIN D.: Robust field-aligned global parametrization. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 135. 3, 10

[MZ13] MYLES A., ZORIN D.: Controlled-distortion constrained global parametrization. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 105. 3

[Orl97] ORLIN J. B.: A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming 78*, 2 (1997), 109–129. 5

[OSCS99] OWEN S. J., STATEN M. L., CANANN S. A., SAIGAL S.: Q-Morph: An indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering 44*, 9 (1999), 1317–1340. 2

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Transactions on Graphics (TOG) 25*, 4 (2006), 1460–1485. 1

[RLS*12] REMACLE J.-F., LAMBRECHTS J., SENY B., MARCHANDISE E., JOHNEN A., GEUZAINET C.: Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering 89*, 9 (2012), 1102–1119. 2

[RVAL09] RAY N., VALLET B., ALONSO L., LEVY B.: Geometry-aware direction field processing. *ACM Transactions on Graphics (TOG) 29*, 1 (2009), 1. 2, 3

[RVLL08] RAY N., VALLET B., LI W. C., LÉVY B.: *n*-symmetry direction field design. *ACM Transactions on Graphics (TOG) 27*, 2 (2008), 10. 2, 3

[TPC*10] TARINI M., PIETRONI N., CIGNONI P., PANOZZO D., PUPPO E.: Practical quad mesh simplification. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 407–418. 10, 11

[VZ01] VELHO L., ZORIN D.: 4–8 subdivision. *Computer Aided Geometric Design 18*, 5 (2001), 397–427. 2

[ZHLB10] ZHANG M., HUANG J., LIU X., BAO H.: A wave-based anisotropic quadrangulation method. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 118. 2