Q-Zip: Singularity Editing Primitive for Quad Meshes

Leman Feng, Yiying Tong, Mathieu Desbrun

Q-zip: Singularity Editing Primitive for Quad Meshes

LEMAN FENG, WeRide
YIYING TONG, Michigan State University
MATHIEU DESBRUN, Inria / Ecole Polytechnique / Caltech

Singularity editing of a quadrangle mesh consists in shifting singularities around for either improving the quality of the mesh elements or canceling extraneous singularities, so as to increase mesh regularity. However, the particular structure of a quad mesh renders the exploration of allowable connectivity changes non-local and hard to automate. In this paper, we introduce a simple, principled, and general *quad-mesh editing primitive* with which pairs of arbitrarily distant singularities can be efficiently displaced around a mesh through a deterministic and reversible chain of local topological operations with a minimal footprint. Dubbed Q-zip as it acts as a zipper opening up and collapsing down quad strips, our practical mesh operator for singularity editing can be easily implemented via parallel transport of a reference compass between any two irregular vertices. Batches of Q-zips performed in parallel can then be used for efficient singularity editing.

CCS Concepts: • **Mathematics of computing → Mesh generation**.

Additional Key Words and Phrases: Quadragulations, irregular vertex editing, connectivity optimization.

ACM Reference Format:
Leman Feng, Yiying Tong, and Mathieu Desbrun. 2021. Q-zip: Singularity Editing Primitive for Quad Meshes. *ACM Trans. Graph.* 40, 6, Article 258 (December 2021), 13 pages. https://doi.org/10.1145/3478513.3480523

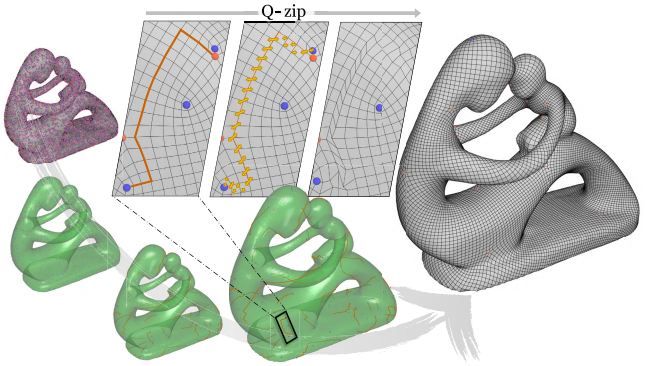


Fig. 1. **Automated singularity editing.** We present Q-zip, an efficient algorithm to displace singularity pairs to improve the regularity of a quad mesh and/or its elements' shape quality. The Q-zip primitive (top) can automatically and efficiently edit a pair of nearby or faraway singularities, see tawny "surgery path" linking singularities of type v3 (blue ball) and v5 (orange ball). Starting from an irregular quad mesh (left: fertility model containing 35.3% of singularities), a few batches of Q-zip operations (green models, with Q-zips displayed in orange) followed by local smoothing cancel out most singularities (right: 7 v3 and 31 v5 are left in the final mesh).

1 INTRODUCTION

Quadrilateral meshes (henceforth abbreviated as quad meshes) are a popular choice of surface representation in shape modeling, texture synthesis, and simulation due to the tensor-product nature of their cells. The presence in a quad mesh of a large number of singularities (i.e., irregular vertices that do not have four adjacent quad cells) is often deemed a nuisance: aesthetically, these topological defects break the tensor-product symmetry and render the mesh unstructured; numerically, these irregularities induce distortion in low-curvature regions, thereby decreasing the accuracy of finite element computations and simulations. Mesh generation methods that globally optimize the number and placement of singularities (see [Dong et al. 2006; Bommes et al. 2009] for early attempts, and [Ray et al. 2009; Myles and Zorin 2013; Ebke et al. 2014, 2016; Huang et al. 2018; Fang et al. 2018; Lyon et al. 2020] for more recent works) avoid this problem to a large extent, but at the cost of high computational complexity. Conversely, efficient quad meshing approaches, e.g., advancing front methods which propagate a layer of elements from boundaries into the domain [Owen et al. 1999] or quadtree-based methods [Rushdi et al. 2017] can produce millions of quads per second in practice [Ansys 2020], but offer little to no control over the number and placement of singularities. As a consequence, post-processing algorithms that can improve an unstructured input quad mesh by displacing or removing its singularities — and hopefully, improving the shapes of its elements in the process — have been highly sought after. However, mesh improvements are often limited in practice to very localized or user-guided edits: existing singularity editing approaches cannot handle distant singularities in a general manner without generating a significant change in vertex count (through chord collapses for instance). Exploring the large space of quad mesh connectivity improvements automatically is thus currently intractable for even moderately large meshes.

In this paper, we present a singularity editing primitive and its algorithmic implementation, with which pairs of arbitrarily-distant singularities can be efficiently moved around a mesh by applying a chain of local topological operations with a minimal footprint.

1.1 Related work

The computational attractiveness of quad meshes for numerical methods in geometry processing and simulation has led to the exploration of flexible and systematic ways to quadrangulate arbitrary domains. Creating a quad mesh without any specific requirements on size, regularity, or anisotropy is, in fact, trivial: one can simply convert any polygonal mesh approximating the domain into a quad mesh through one step of Catmull-Clark subdivision [Catmull and Clark 1978]. However, controlling the number of singularities and the shape of quad elements is paramount in ensuring high numerical accuracy in computations. Consequently, a large array of meshing approaches have been devised over the years [Bommes

Authors' addresses: L. Feng, WeRide, Guangzhou, China; Y. Tong, Michigan State University, East Lansing, MI, USA; M. Desbrun, Inria Saclay, LIX/DIX, Institut Polytechnique de Paris, Palaiseau, France; on leave from Caltech.

0730-0301/2021/12-ART258
https://doi.org/10.1145/3478513.3480523

et al. 2012] to help design quad meshes with various requirements. Far less attention has been given to the *post-editing* (including motion and cancellation) of the resulting singularities: the non-local constraints that a pure quad mesh structure imposes render the design of algorithmic approaches dedicated to this task difficult or computationally intensive. Yet, singularity editing is highly desirable since state-of-the-art automatic quadrangulation techniques cannot rival the clean, high-level structures of carefully-designed meshes that are manually created by specialists in gaming, animation, or simulation. Before explaining our contributions towards this general goal, we first briefly review relevant techniques.

Decimating quads. A first family of approaches reduces the number of singularities in a mesh through *mesh simplification* — see, e.g., [Daniels et al. 2008, 2009; Tarini et al. 2010, 2011; Xu et al. 2020]. These methods iteratively decrease the number of singularities by decimating an input quad mesh through local operations (e.g., quad collapse or doublet removal) and/or semi-global operations (e.g., (half-)polychord removal, separatrix split between pairs of 5-singularities, or separatrix collapse between pairs of 3-singularities) that preserve boundary features. While such a strategy leads to a drastic reduction in singularities, it is also computationally intensive and changes the vertex count of the input mesh significantly.

Singularity editing. A sequence of well-chosen local edge flips, vertex splits, and quad collapses can be an efficient way to remove extraneous singularities without affecting the vertex count much [Docampo-Sanchez and Haimes 2019]. However, such a local approach will stop improving the mesh once all the singularities are sufficiently isolated from each other, rendering it useless on large meshes. A template-based approach, such as Minimum Singularity Templates [Verma and Suresh 2017], can efficiently remove singularities that are further apart by finding *convex patches* of quads containing three or more singularities, and comparing these patches with a set of "templates" to determine whether the patch can be remeshed with fewer singularities. Unfortunately, the number of possible patches grows exponentially with their size, so finding patches above a moderate size becomes intractable. Noticing that separatrices between two singularities in a quad mesh are often long and helical, Bommes *et al.* [2011] introduced a way to simplify the base complex while keeping the singularities fixed. To achieve this editing task, they introduce the notion of *grid-preserving operation*: if a mesh surgery follows the rules induced by a finite automaton, the result is guaranteed *not to have additional singularities*. This concept was shown to be a generalization of the polychord collapse introduced in [Daniels et al. 2008]. The work of [Peng et al. 2011], building upon [Li et al. 2010], offered an even more general framework for singularity editing: they demonstrated that one could *move certain pairs of singularities around a mesh* without introducing additional singularities and *with minimal connectivity changes*. Their proposed mesh surgery is conceptually simple: one just has to split an initial singularity to form a transient singularity pair, and then "drag" this transient pair along a "surgery" path going through only *regular* vertices and leading to the second singularity, with which it will merge and recombine, resulting in an overall motion of the initial pair. Such a process offers efficient singularity editing, allowing to move irregular vertices around to improve the mesh quality, cancel singularities, or both. However, Peng *et al.* [2011] do not provide an automated approach or algorithm which can implement this idea for any configuration of singularities. The user must thus manually trigger selected edits on a case-by-case basis to reduce the number of singularities, which can be a long and tedious task if a singularity pair is not inside a convex region without any other singularities — a case happening quite often on common meshes (see Figs. 2 and 12), yet not covered by the extensive analysis of [Peng et al. 2011]. Moreover, the consequences on the final mesh connectivity of a change in the surgery path were not discussed.

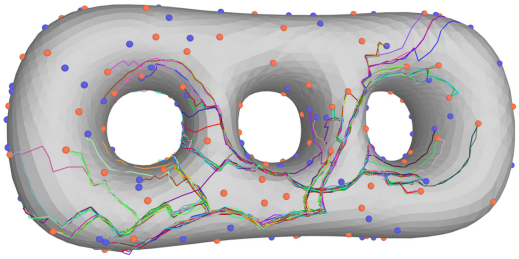


Fig. 2. **Paths between singularity pairs.** Even on a simple 3-hole mesh with 4866 vertices, we easily find many pairs of singularities such that even the *shortest* paths connecting them do not stay within a convex region free of other singularities. Orange balls mark v5 vertices, while blue balls are v3 vertices; 82 such pairwise shortest paths are shown in various colors, many having more than 4 left (or right) turns, i.e., the path would self-intersect if laid out in the plane.

1.2 Motivation

Based on prior work, we identified a few concepts and exigencies which motivated and helped shape our contributions.

Need for singularity editing. Quad meshing has made impressive advancements in the past two decades. Yet, several authors have argued that it only makes the need for singularity editing more pressing. For instance, an automatically-generated quad mesh is easily perfectible in practice as it is often riddled with helical separatrices as pointed out in [Bommes et al. 2011]. Peng *et al.* [2011] also demonstrated that even meshes obtained from optimization techniques such as Mixed-Integer Quadrangulation [Bommes et al. 2009, 2013] or wave-based anisotropic quadrangulation [Zhang et al. 2010; Fang et al. 2018] can be improved further via singularity editing — although most edits require user guidance as no automated approach to achieve these improvements exists. Both papers also noted that quad decimation techniques, which automatically simplify the base complex on an input mesh at the cost of a drastic change in vertex count, do not offer the kind of efficient connectivity improvement that is sought after: one has to work instead with "*grid-preserving operators*", i.e., chains of local mesh operations that do not introduce additional irregular vertices [Bommes et al. 2011]. An automated singularity editing framework is thus needed if one wishes to compete in regularity with hand-designed meshes.

Importance of singularity pairs. A number of techniques related to singularity editing have noticed the key nature of *separatrices*, which can be viewed as parameter lines connecting two irregular vertices. In fact, Bommes *et al.* [2011] showed that rectifying a

few separatrices that are long and helical can already dramatically improve the base complex of a quad mesh. Peng *et al.* [2011] also showed that edits need to involve at least two irregular vertices as one cannot requadrangulate a convex region containing a single irregular vertex without introducing additional irregular vertices: a singularity pair is thus the *smallest possible editing operation that does not increase the number of irregular vertices in the mesh*. Moreover, the two singularities can be very close (even adjacent, i.e., sharing en edge, at its shortest) or quite distant: singularity editing acting on a pair of irregular vertices thus covers the whole gamut of local, semi-global, and global editing actions that previous works focused on. Our contribution will exploit and generalize this fundamental editing operation on singularity pairs, which suffered a case of *arrested development* for the last decade after the publications of [Peng et al. 2011] and [Bommes et al. 2011].

1.3 Contributions

Building upon prior work, we contribute in this paper a novel and general grid-preserving operator between any two singularities that we call Q-zip. This canonical quad editing primitive displaces two singularities in concert by one edge relative to the remainder of the mesh through a sequence of localized edge splits and quad collapses performed along an arbitrary simple edge path joining two singularities. Q-zip is a zipper-like, general variant of the pairwise operation proposed in [Peng et al. 2011] that does not restrict the valence of vertices along the path between the two singularities. It is also automatically grid-preserving (i.e., it does not introduce additional singularities), without the need for an automaton as proposed in [Bommes et al. 2011], through the use of parallel transport. We also prove that the effect of a Q-zip operation on the connectivity of the mesh is *unique* for all edge paths *in the same homotopy class*, reinforcing our claim of its canonical nature. This versatile connectivity operation can then be used to explore connectivity optimization through, e.g., repeated Q-zips in order to reduce an energy functional as we will demonstrate.

2 SINGULARITY PAIR MOVEMENT

We now delve into our approach to move a given pair of singularities of a pure quad mesh through the Q-zip algorithm by performing mesh surgery along a simple path between singularities.

2.1 Definitions and assumptions

While we will mostly use conventional nomenclature, we start this section with a brief recap of the basic definitions of terms and concepts that we will use throughout the remainder of the paper, as well as our assumptions on the input quad mesh.

Mesh terminology. We assume that an input mesh $\mathcal{M}$ is a regular CW-complex formed by the collection of a vertex set $\mathcal{V}$, an edge set $\mathcal{E}$, and a face set $\mathcal{F}$ such that the boundary of each face is a set of four edges in $\mathcal{E}$ (forming a quadrangle), and the boundary of each edge is formed by two vertices in $\mathcal{V}$. (We will also discuss the case of hybrid meshes, with both triangles and quadrangles, in Sec. 3.2.) The mesh is further supposed to be isomorphic to an orientable 2D surface (i.e., it tessellates a 2-manifold), possibly with boundary. The *valence* of a vertex denotes the number of its incident edges, so a vertex is *regular* if its valence is 4, and called a *singularity* otherwise. Note that we will use v3, v4, and v5 as shortcuts to refer to valence-3, valence-4 and valence-5 vertices respectively.

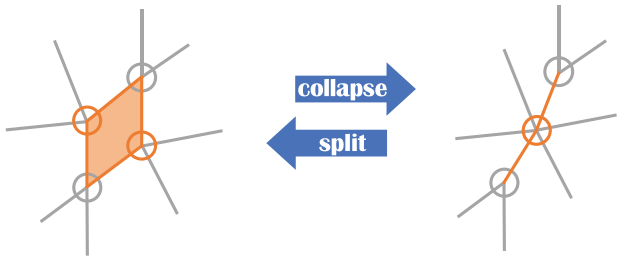


Fig. 3. **Canonical operators.** The *collapse* operation deletes a quad, a vertex, and two edges, while the *split* operation forms a new quad, a new vertex, and two new edges. Q-zip editing only applies a series of these operations along a surgery path to shift a pair of irregular vertices.

Simplifying assumptions. In the remainder of this section, we simplify our explanations by assuming, without loss of generality, that all interior vertices in $\mathcal{V}$ have valences 3, 4 or 5. This constraint is easy to enforce in a preprocessing step: a valence-2 vertex can be directly removed (reducing the valence of each of the two adjacent vertices by one); and any vertex of valence 6+ can be split into vertices of lower valences. Note that our connectivity editing tool will not introduce any valence-2 or valence-6+ vertices, so this step only needs to be done once.

2.2 Foundations of Q-zip

We begin our exposition with a few principles upon which our method builds, both for simplicity and efficiency.

Two atomic topological operations. In order to avoid requadrangulating entire regions from scratch, connectivity editing should proceed through local topological operations with a minimal footprint that preserve the Euler characteristic $\chi = |\mathcal{V}| - |\mathcal{E}| + |\mathcal{F}|$. Two atomic operations, called *quad collapse* and *vertex split*, involve the least amount of changes in the mesh connectivity while keeping only quad faces in the mesh: removing (resp., adding) a quad simultaneously involves the merging of two edges (resp., the splitting of an edge in two) and the merging of two vertices (resp., the splitting of a vertex into two), see Fig. 3. We will express our algorithm using only these two canonical operations as they are known to generate all others (more involved) quad-mesh topological changes, such as edge rotations; moreover, since they are inverse of each other, it will allow us to easily undo a surgery sequence if needed.

Strip insertions and deletions. Applying a quad collapse or a vertex split on a quad mesh does not generate non-quad faces, but it affects the valence of the neighboring vertices: as Fig. 3 indicates, a collapse increments the valence of each of the two merging vertices by one while decreasing by one the valence of the two other vertices of the collapsed quad — the same happens for a vertex split with opposite valence changes. Thus, applying these two operations arbitrarily is bound to introduce a large number of new irregular vertices. However, a serial application of collapses along a *strip of contiguous quads* cancels valence changes along the strip, leaving only two changed valences at each end of the stitched-up strip; this is, in fact, a simple case of the traveling transient 3-5 pair proposed by Peng

et al. [2011]. Similarly, a serial application of splits along a simple edge path adds a quad strip to an existing mesh with, again, only a pair of changed valences at each extremity.

Chained operations between singularities. Noting that a strip deletion followed by a strip insertion starting on the last stitched edge also cancels out two valence change pairs, we can further chain a number of strip insertions and deletions together, leaving no changed valence behind except at the beginning and the end of the chain, see Fig. 4. It is thus particularly strategic to have these chained topological surgeries start and end at irregular vertices: it will induce a move for this pair of irregular vertices as advocated and analyzed in [Peng et al. 2011]. Repeated movements of 3-5, 3-3, and 5-5 vertex pairs can then be the basis of editing techniques seeking a reduction in the number of irregular vertices, or an improvement in the mesh elements' quality.

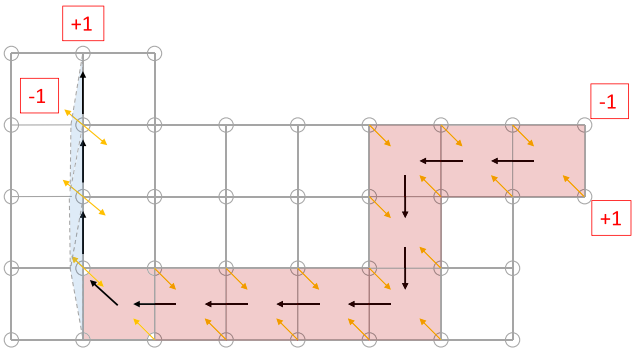


Fig. 4. **Chained strip operations.** Three strip collapses (in red) and one strip split (the soon-to-be-formed quads are in blue) chained up serially on a quad mesh only result in a pair of valence changes at both ends.

Automated singularity editing. While moving pairs of irregular vertices in [Peng et al. 2011] was mostly user-driven, handling large quad meshes can only be achieved through a fully automated approach which can *systematically* edit arbitrary meshes through this pairwise-editing primitive: although local mesh editing could enumerate all possible valences and boundary presence in a small neighborhood and form associated templates, large-scale mesh surgical operations must proceed in a methodical fashion to handle all situations simply and reliably in a grid-preserving fashion. Thus, finding *automatically* a series of chained operations between a given singularity pair that induces the desired pairwise motion is key.

Generality. Finally, we seek an approach general enough to guarantee that any pair of irregular vertices, however close or remote, can be potentially moved: they should not be artificially obstructed by unnecessary topological constraints if we want our connectivity tool to be usable at any scale. We prove that our operators provide equivalent connectivity editing through arbitrary homotopic paths between pairs of irregular vertices, even when these paths run through nonconvex regions on meshes such as in Fig. 12. This very general algorithm *guarantees* that performing a pairwise movement of irregular vertices through chained operations along two different paths starting and ending at the same vertices results in isomorphic meshes if the paths are homotopic to each other within

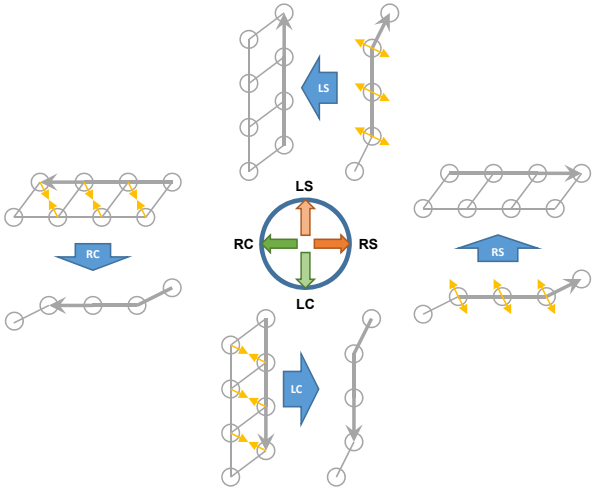


Fig. 5. **Compass of strip operations.** Four strip operations (RS for right-split, LS for left-split, RC for right collapse and LC for left collapse) are indicated by a "compass", to keep track of the operations to perform.

a regular region, i.e., if one can deform one into the other without sweeping over irregular vertices. Moreover, while one may want to operate along a shortest path between two singularities to minimize the amount of editing operations, the grid preserving movement of irregular vertices through Q-zip can operate on arbitrary simple paths that may intersect other singularities or even encroach on boundaries, proving its robustness.

Compass of actions. One key ingredient to achieving our goals based on the concept of topological "surgery" of quad strips is to consider the four distinct types of strip operations: as sketched in Fig. 5, they are what we call *right collapses* (or **RC**, where all quads are collapsed towards the right along the strip, see left), *left collapses* (or **LC**, where all quads are collapsed towards the left along the strip, see bottom), *right splits* (or **RS**, where quads are created on the right of the path through splits, see right), and *left splits* (or **LS**, where quads are created on the left of the path through splits, see top). These four possible strip operations can be chained together as explained earlier; moreover, the compass summarizing these operations, drawn in the middle of the figure, helps keep track of the canonical directions that each operation affects. Therefore, if we were to carry, or more precisely "parallel transport", this compass along a series of chained strip operations on a regular quad mesh as in Fig. 4, the hand of the compass aligned with the next edge on which to operate actually indicates *which* operation (out of LS, RS, LC, or RC) has to be performed. If we follow the instructions provided by this compass (*which are akin to the states of an automaton* [Bommes et al. 2011]), we guarantee that no irregular vertices can be created along the "surgery" path. This simple, but powerful concept of a parallel-transported compass will be the core of our approach, and will ensure its generality.

2.3 Overview of the Q-zip primitive

Based on these foundations, we define a Q-zip operation on two arbitrary singularities as the overall grid-preserving mesh surgery that is achieved in four simple and distinct steps:

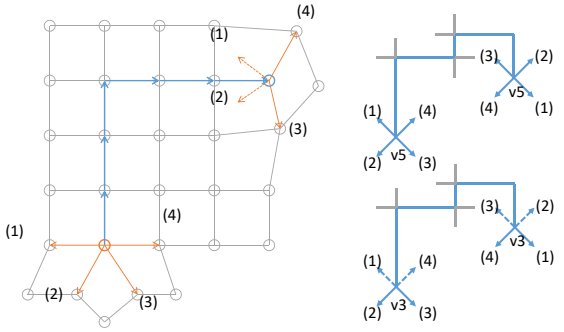


Fig. 6. **Singularity pair movements.** For any of the four possible movements (arrows) of a singularity, there is a unique corresponding movement on the other singularity in the pair. Left, a mesh example for a 3-5 pair; right, a schematic description of a similar behavior for 3-3 and 5-5 pairs.

① *Surgical sketching:* Once two singularities have been selected, we compute a simple *edge path* joining them: it defines a **surgical path** along which the Q-zip operation will be performed. See Sec. 2.4 for details.

② *Surgical planning:* From one of the two singularities (picked randomly), and based on where we want this singularity to be displaced towards, we propagate a **compass** along the vertices of the surgical path through *parallel transport*. This compass, keeping track of the (u, v) directions relative to the singularities, will allow us to plan the surgery: instructions regarding whether a split or collapse needs to be performed locally will be directly read off from the compass at each vertex along the surgery path as we will detail in Sec. 2.5.

③ *Surgery preparation:* We then perform minor adjustments of the surgery path and its compasses *restricted to the one-ring of each singularity* (i.e., for the first and final few operations) in order to ensure an exception-free overall surgery as reviewed in Sec. 2.6.

④ *Surgery execution:* The actual surgery then serially follows the instructions given by the compass as one travels along the surgery path. Resulting changes to the mesh thus fall in two categories: a) the **surgery footprint**, which contains all the mesh elements that have *disappeared* (quads through collapses, edges though split), and b) the **surgery scar**, which contains the *new* mesh elements (quads that appeared out of split operations, and edges through which quads collapsed); see Sec. 2.7.

We now cover how to perform each step in detail.

2.4 Surgery sketching

From two chosen irregular vertices of an input mesh, we first plan a Q-zip surgery to displace them by selecting an edge path connecting them, then explore the four possible ways to move them around.

Surgery edge path. The precise selection of the path (i.e., a series of connected edges joining the two singularities) is entirely up to the user, but the edge path should be *simple* (not having repeating vertices) to allow for trivial reversibility of the Q-zip operation. A valuable choice is to compute a *shortest* path in between the two singularities, through, e.g., Dijkstra's algorithm where each edge has a unit weight as it will minimize the number of splits and collapses of the surgery; we can even incorporate high-level strategies in this path search, by for instance increasing the weights of the edges we really do not want altered, or to tweak the BFS search to explore non-boundary vertices first to avoid affecting boundaries as much as possible. We will from now on refer to the resulting simple edge path as the *surgery path*. Finally, we arbitrarily assign an orientation to this path, thus now classifying the two singularities into an *initial singularity*, and a *terminal singularity*.

Four possible movements. Because Q-zip leads to a shift of the initial singularity by one edge, the vertex that is currently the initial singularity will turn into a regular vertex; thus one can think of the surgery as having four possible effects: the initial singularity will end up at one of the four neighbors of the newly-affected vertex where the singularity used to stand. This is precisely one of the contributions of Peng *et al.* [2011], who showed that a pair of singularities could only move in one of four directions (Fig. 6). In the case of our Q-zip operator, we can simply pick a random operation (LS, RS, LC, RC) to kickstart the whole surgery: each choice, which we denote as the *initial operation*, will correspond to one of the four possible directions. The other operations will lead to different surgeries, displacing the singularity pair in three different directions.

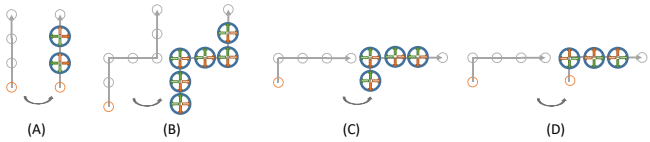


Fig. 7. **Parallel-transport of compass.** Here, we depict a few examples of parallel transport of the compass along the surgery path. Cases (A) and (D) start with LS and RS respectively, so there are no compasses labeled on the starting vertex (red). On the contrary, cases (B) and (C) start with RC, so the starting vertex is labeled with a command corresponding to the hand of the compass aligned with the first edge.

2.5 Surgery planning

For a given surgery path and one of the four choices for the initial operation, we assign a compass (see center of Fig. 5) at each vertex. In our visualization, we always align one of the four directions of the compass with the next edge along the path; our code proceeds similarly, in the sense that we tag each vertex with the operation (RS, LS, RC, or LC) indicated by the compass hand pointing towards the next edge. Assigning the proper operation per vertex thus requires a very simple rule, best understood as *parallel-transporting a frame* as it keeps track of the four directions corresponding to an implicit choice of (u, v) coordinates at the beginning singularity. First, we initialize the first vertex of the surgery path with the *initial operation* tag, i.e., one of the four possible motions of the original singularity; this amounts to orient the compass at the beginning of the surgery. Then we propagate the compass through the surgery path, assigning the operation on the next vertex of the surgery path as follows:

- if the next vertex is *regular* (v4), we implicitly translate the compass from the current vertex to this vertex, and store at this vertex the operation that is indicated by the compass hand pointing towards the next edge. In other words, if the path goes straight at this next vertex, we keep the same tag as the previous vertex;

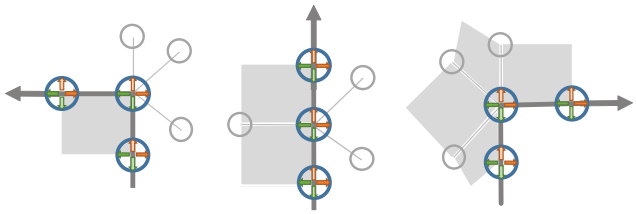


Fig. 8. **Compass transport for singularities along surgery path.** If an irregular vertex is encountered along the surgery path and the previous edge is either LS or RC, we define the notion of parallel-transport of the compass simply based on the number of quads on the left side of the surgery path at the singularity: a single one (left) amounts to a left turn, two (middle) amount to going straight, while three or more (right) imply a right turn. We adopt a mirrored convention for the RS and LC cases.

if the path turns left (resp., right), we pick the next counterclockwise (resp., clockwise) operation on the compass, see Fig. 7.

- if the next vertex is irregular, parallel-transporting our compass needs to be properly defined, and there are in fact multiple options one could choose from. We opted for a simple deterministic rule to handle this case as it makes the code simpler while offering a robust solution with no adverse impact other than systematically picking just one of the many ways with which one could proceed. When we reach a singularity and we try to parallel-transport the compass to the next vertex along the surgery path, we count the number of quads on the left of the surgery path that are adjacent to the singularity if the previous edge is LS or RC. If only one quad is present, we consider this a left turn; if two quads are present, we consider that this corresponds to going in a straight direction; if three or more quads are present, we consider that a right turn is happening. If the previous edge is RS or LC, we count from the right-hand side instead. We then assign the appropriate tag based on this change of direction, i.e., we keep the same tag if we go straight, or we pick the next counterclockwise (resp., clockwise) operation on the compass if the path turns left (resp., right), see Fig. 8.

This process is stopped when we reach the terminal singularity. With our choice of parallel-transport for singularities along the path (which is actually a simple extension of the regular case as they match if there are three or less quads to the left of the surgery path at each path vertex), our implementation allows irregular vertices along the surgical path: it simply amounts to assuming that one of its incident edges is a boundary edge, or equivalently, that we consider the actual path as winding around the singularity without having to actually alter the path (and risk more special cases to have to deal with if we were to do so).

Realizable motions. We note here that the four possible types of initial operations at the starting singularity lead to a specific terminal operation stored on the final singularity, thus *imposing* on this irregular vertex a corresponding movement, out of its four possible ones. This exactly reflects the contribution of [Peng et al. 2011] that elucidated the possible pairwise motions of singularities. Note that we will provide a proof of invariance of the editing procedure up to homotopy in Sec. 3.1; therefore, this four-motion property holds for any simple path in a simply-connected region. Moreover, our use of a parallel-transported compass also implicitly enforces the grid-preserving property pointed in [Bommes et al. 2011] described via an automaton, as we keep track of the four canonical directions of a regular grid along the way. However, for an arbitrary input mesh, a given pair of singularities may not be moved in four different directions — not because of topological constraints on Q-zip, but because of the properties of the input mesh along the surgery path. For instance, if one of the singularities lies on a boundary, this may severely restrict its movements; similarly, if the surgery path goes through a one-quad-wide bottleneck of the mesh, some of these operations assigned by our compass may not be realizable; or the surgery involves collapsing a quad we want to preserve at all costs for instance. Thus, at this stage, we quickly make sure that each tag assigned along the surgery path corresponds to a *realizable operation*; if there is not enough room to perform the designated operation, we simply cancel the corresponding surgery, thus removing one possible motion for the pair of singularities.

2.6 Surgical preparation

Before performing surgery on a pair of singularities, we may need to adjust the surgery path and compass assignments slightly to guarantee an exception-free surgery: we modify the extremities of the surgery path and their actions to account for the fact that the final split or collapse may need extra care to end properly. This initial and terminal alteration of the surgery path, *strictly restricted to the one-ring of the two singularities*, is the last needed preparation, so that a systematic application of chained mesh operations can be performed *without* having to deal with special cases later on. At the start of the path, only three cases can present themselves:

- If we start from a valence-3 singularity with a (left or right) split, or if we start from a valence-5 singularity with a (left or right) collapse, the simple path in the direct vicinity of the singularity (i.e., in the one-ring neighborhood) does *not* need to be adjusted. However, for a v3 starting with a (left or right) split, the first operation tag is changed to `no-op`.
- If we start from a v3 with a left (resp., right) collapse, we start the surgical path from the adjacent vertex on the left (resp., right) of the singularity corresponding to the tail-end vertex of the collapse, and follow the border of the one-ring neighborhood to reconnect to the original edge path, see Fig. 9(a). The first vertex of the adjusted new path keeps the original starting tag (i.e., left (resp., right) collapse), and the second vertex is assigned a right (resp., left) collapse tag at the turn so that they are all consistent w.r.t. parallel transport, independent of their respective valences.
- Similarly, if we start from a v5 with a left (resp., right) split, we start the surgical path from the adjacent vertex on the left (resp., right) of the singularity corresponding to the end vertex of the split, see Fig. 9(b). This prepended edge will serve as a guide for the first split: this new edge and the former first edge of the surgery path defines the (left or right) split operation. Consequently, the new initial vertex of the surgery path is assigned a *no-op* tag as it is only there to help handle the next vertex.

Conversely, the end of the surgery path is adjusted as follows:

- No adjustment is needed if we end at a valence-3 singularity.

- When the surgery path ends at a v5 with a left (resp., right) collapse tag, we end the surgical path at the closest neighbor on the left (resp., right) side where the collapse ends, see Fig. 9(c). The action tag of this vertex right before the last v5 is thus changed from a left (resp., right) collapse to a right (resp., left) split, reflecting that the compass stays the same but we change the "next edge" direction. Accordingly, the new final vertex is tagged as a left (resp., right) collapse.
- When the edge path ends at a v5 with a left (resp., right) split, we end the surgical path at the farthest neighbor on the right (resp., left) side where the split ends, see Fig. 9(d). This added final edge just helps us define the endpoint of the last split action. Consequently, this new terminal vertex of the surgery path is simply assigned a *no-op* tag.

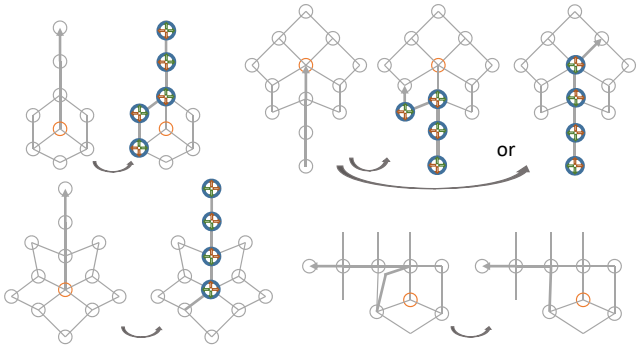


Fig. 9. **End-point adjustments.** For an initial v3 (top left), (left or right) collapses start at the vertex to the corresponding side of the v3. For an initial v5 (bottom left), splits start with the backward neighbor on the corresponding side. For a final v5 (top right), collapses end at the closest neighbor on the corresponding side, while splits end on the farthest neighbor on the corresponding side. Finally, if the surgery path folds onto itself in the one-ring of the final singularity (bottom right), we simply cancel the edges being crossed twice in opposite directions.

Clean-up of adjustments. Finally, if any surgical path edge ends up overlapping another due to the adjustments we performed, we cancel them out in pairs as shown in Fig. 9 (bottom right), and adjust the action tags accordingly so that the local change of trajectory is accounted for while keeping the local direction of the compass unchanged. These minor adjustments allow Q-zip to work seamlessly and without templates for *any* pair of singularities, whether they are just adjacent or far away from each other.

2.7 Q-zip execution

Once all the vertices of a surgery path have been labeled with a local topological operation, the Q-zip surgery is trivial to execute.

Serial execution. From the first vertex of the surgery path, we begin performing the labeled operations serially as we travel along the path, and stop at the last vertex (which is either the second singularity, or one of its direct neighbors based on the adjustment that was performed). The execution of Q-zip is rather trivial and requires no special care: from a compass tag at a vertex and knowing the edge right before and right after the vertex along the surgery path, we can blindly apply the topological changes indicated in

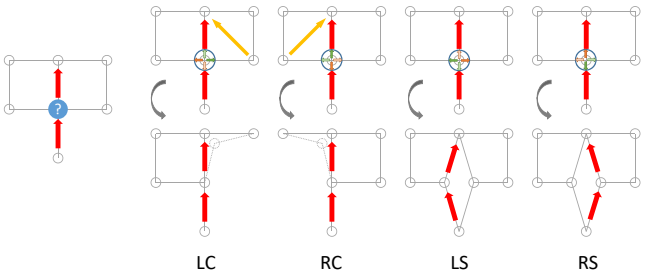


Fig. 10. **Surgical instructions.** Four atomic operations instructed by the hand of the compass. From left to right: left collapse (light green), right collapse (dark green), left split (light orange), and right split (dark orange).

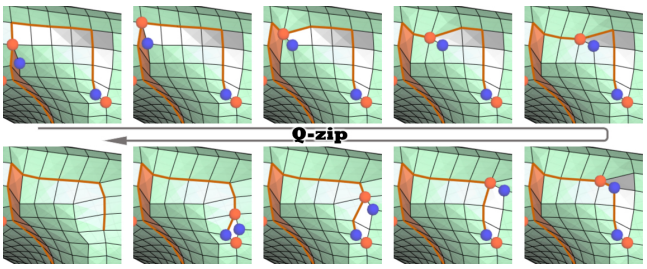


Fig. 11. **Q-zip in action.** As Q-zip is executed, a transient 3-5 pair travels along the surgery path as quads are successively collapsed or created via edge splits, until the pair encounters the final v3, creating a regular vertex.

Fig. 10, independent of the valences of the vertices involved as the path and compasses have been specifically designed to account for all cases seamlessly — see Fig. 11 for a concrete occurrence on a mesh. However, when a singularity is passed as we move along the surgical path, *two exceptions* can happen after the topological operation at the irregular vertex has been executed: the merging of two v3's into a valence-2 vertex, or the merging of two v5's into a valence-6 vertex. In both cases, a simple post-processing suffices: the valence-2 vertex is removed with the introduction of two v3's, and the valence-6 vertex is split into regular vertices with the introduction of two v5's.

Footprint and scar. Notice that the chained topological operations that Q-zip executes only add or remove quad strips. The effect of the Q-zip surgery is thus quite obvious: the surgery footprint over the mesh is the union of all the elements that were removed as a result of the surgery (i.e., quads that got collapsed, and edges which got split); inversely, the leftover "scar" is the union of all new elements (i.e., quads that got created through splits and edges which got formed through collapses). Both these regions share the surgery path edges, by definition. If smoothing needs to be performed following a Q-zip operation, we know precisely which parts to target.

2.8 Discussion

As promised at the beginning, our Q-zip primitive and its associated algorithmic implementation is very general: our few parallel-transport rules and extremity adjustments make the overall algorithm apply for v3-v5, v3-v3, and v5-v5 pair movements, whether irregular vertices are present along the surgery path or not. This significantly increases the applicability of Q-zip , and renders its

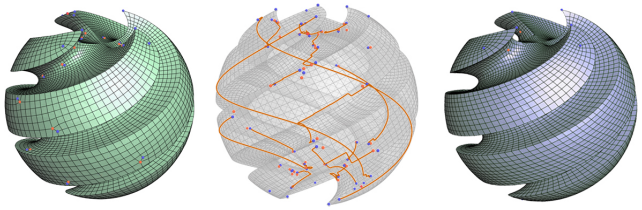


Fig. 12. **Spiraling features.** Reducing the number of singularities on this mesh with spiraling sharp features (left) forces long surgery paths (middle) to act on pairs of remote singularities; Q-zip can handle this case without difficulty (right): while the initial mesh has 55 v3, 43 v5, and 2 v6+, the resulting mesh after Q-zip surgeries has only 16 v3 and 8 v5.

application totally automated; it can even be reverted without having to store the intermediary steps as starting a surgery from the opposite singularity and with a compass rotated by π just undoes the former Q-zip operation. Note that we omitted a discussion on self-intersecting paths, since they are rarely useful in practice; but they can be decomposed into the union of loops and simple paths if they are needed, provided that the loops are not around singularities. Finally, we reiterate that the way we incorporated a notion of parallel transport for the action compass is, in essence, a substitute to the automaton concept introduced in [Bommes et al. 2011] to guarantee a grid-preserving operator, providing also a more geometric understanding of why chained topological changes of quad strips creates no new singularities.

3 ANALYSIS AND EXTENSION OF Q-ZIP

We now provide an analysis of our Q-zip operator, proving that the path used between two singularities actually does not affect the result of the surgery as long as the surgery path remains within the same homotopy class. We also extend our novel operator to the case of hybrid meshes, i.e., for meshes containing mostly quads but also a few triangles as often found in practical applications.

3.1 Topological analysis of Q-zip

Our Q-zip procedure inherits a strong topological property from its principled design: for a given singularity pair and a choice of one of the (maximum) four possible displacements of one of these singularities, the resulting meshes after a Q-zip operation along *homotopic* simple surgical paths are *isomorphic*. In other words, the actual surgery path is mostly irrelevant (up to its winding around other singularities) to the produced mesh: this property makes Q-zip canonical. We provide next a description and proof of this property for completeness.

Given a starting singularity v_s and a terminal singularity v_t on a quad mesh $\mathcal{M}$, consider two simple surgery paths p_1 and p_2 connecting them. Based on our treatment of singularities along the path, we consider the path to be on the left or right of the singularity based on our deterministic rule. Two paths are called *homotopic* if we can continuously deform one to the other without crossing any other singularity (which are considered as punctures) on the surface; the same holds on a quad mesh, where now the deformation of the path is done one quad at a time, so the two surgery paths are homotopic if they bound a region formed by unions of regular quads. With these definitions, we proceed to prove the following statement.

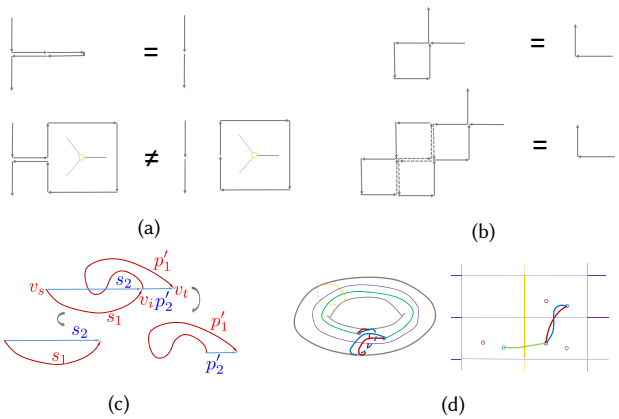


Fig. 13. **Illustrations for the proof.** (a) Equivalence under homotopy. (b) Resulting mesh isomorphism for a single quad loop implies isomorphism for a union of loops. (c) For a simply connected region, intersecting homotopic paths can be reduced to case (b). (d) Homotopic paths (blue and red) are lifted to paths with same endpoints in the universal covering. Note that the green path is not in the same homotopy path as it is lifted to a different copy of the endpoint in the universal cover.

Theorem. *If p_1 and p_2 are homotopic in $\mathcal{M}$, then the meshes resulting from running Q-zip along p_1 and p_2 are isomorphic.*

Proof of Theorem. First, we note that any two consecutive edges in the surgery path that are on the same edge with opposite directions, i.e., a local path of the form $v \to w \to v$ that goes from vertex v to another vertex w and then immediately back to v, would result in a collapse and a split that cancel out each other. Thus, we can repeatedly remove such "folded" cases, see Fig. 13(a). Note that this does not imply that homologous paths lead to equivalent mesh edits in general since the same edge may carry a compass pointing at a different direction after looping around a singularity.

Next, it is trivial to verify that for any compass, following a path containing a loop around a single quad will locally result in two splits and two collapses. Thus the resulting mesh is isomorphic to the mesh before these four operations. By repeatedly applying the above isomorphism, we can thus see that a loop around a union of quads can be removed from a surgery path (Fig. 13(b)) without affecting the mesh connectivity. Now, for a simply connected surface, any two non-intersecting paths form the boundary of a topological disk, which is a union of quads—so the equivalence for Q-zip running on either path follows in such a case. If the two paths intersect as in Fig. 13(c) instead, call the intersection between p_1 and p_2 closest to the starting vertex v_s on p_1 is v_i. The surgery segment s_1 between v_s and v_i on p_1 and the segment s_2 between v_s and v_i on p_2 have no intersection other than the start and end vertices, thus the two segments result in isomorphic edited meshes. The segments that are left can be denoted as p_1' and p_2'. If they result in isomorphic edited meshes as well, the concatenation of chain operations results in equivalent editing implied by the composition of the two isomorphisms. Thus, the equivalence between p_1' and p_2' can be established through recursion, which always terminates as segment lengths always decrease. In our illustrative example, as p_1' and p_2' are not intersecting, it terminates after the intersection v_i is processed.

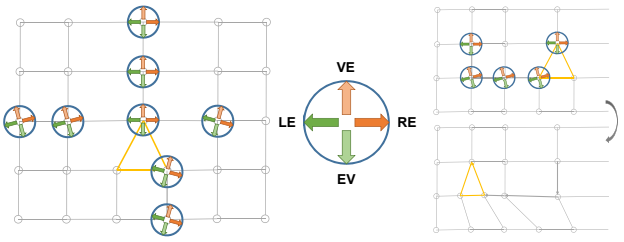


Fig. 14. **Hybrid Q-Zip.** Given an isolated triangle in a sea of quads, the connectivity can be locally updated so that this triangle moves by one edge in four possible directions, similar to the general case discussed in Q-zip and still resembling a zipper-like surgery. The four operations are vertex split to edge (VE), edge collapse to vertex (EV), left edge flip (LE) and right edge flip (RE). The parallel-transport of this hybrid compass along the simple path contains a small rotation as an indication of where the base edge of the triangle is (i.e., the EV direction), after the tip is shifted to the current node on the simple path. If EV points towards an existing vertex, that vertex will be split into an edge when the triangle reaches the compass location.

Finally, to show the equivalence for arbitrary topology, we can simply invoke the universal cover of the quad mesh. Recall that a *covering space* of a quad mesh $\mathcal{M}$ consists of another quad mesh $\overline{\mathcal{M}}$ and a continuous surjective map $\pi : \overline{\mathcal{M}} \to \mathcal{M}$ satisfying the condition that, for each point $x \in \mathcal{M}$, there exists an open neighborhood U of x such that $\pi^{-1}(U)$ is the union of disjoint sets, each of which is mapped homeomorphically onto U by π. A *universal cover* is then a covering space that is *simply connected*, i.e., any loop in this space is homotopic to a point — for more background, see [Hatcher 2002; Yin et al. 2007; Kälberer et al. 2007]. Now, when two paths are homotopic, they lift into two paths with the same starting and ending vertices in the universal cover, which is a simply connected space, thus completing the proof (see Fig. 13(d)). Of course, when the paths are not homotopic to each other, these paths lifted on the universal covering will not share the same starting and end vertices, leading to potentially different post-surgery connectivity. □

3.2 Q-zip for triangle-quad meshes

In engineering applications, the use of quad-dominant meshes is commonplace — especially because many fast quad meshing techniques end up having sparse triangles among mostly quad elements, to adjust the local edge flow densities without having to change the directions of quad edges as shown in Fig. 14. We can easily extend our Q-zip editing tool to handle such meshes, where the triangles are composed of one v5 (tip), and two v4.

Dealing with triangles along Q-zip path. When a Q-zip surgery path comes in contact with a triangle, its three edges can be treated as boundary edges: this ensures that a quad collapse operation will not be attempted on the triangle. Of course, an alternate solution is to simply wind the path around the triangle when possible, removing the issue altogether. No other aspect of Q-zip needs to be altered.

Q-zip for a triangle. Moving a triangle within a quad mesh has already been proposed, see for instance [Tarini et al. 2010], in their Section 6.1: triangles can be made to "crawl" over the hybrid mesh toward each other until they can be merged into quads. In fact, this well-known property is, in disguise, a Q-zip operation itself. Recall that our chained sequence of local split and collapse operators can be seen as a zipper that changes the number of edges when moving along two directions. Now, moving the tip of the triangle also amounts to creating a sequence of vertex splits or edge collapses as sketched in Fig. 14: when moving along the direction pointed by the triangle, it unzips (going up) or zips up (going down) the edges along the path; when moving sideways, it just reconnects the edge paths along the direction of the triangle. More precisely, while Q-zip is in essence the transportation of a transient adjacent 3-5 pair, we can interpret this hybrid Q-zip as the transportation of *half* of a transient 3-5 pair, aligned to one of the 4 edge directions in the regular region. So moving the triangle by one edge in one of the four possible directions is, itself, a Q-zip operation! However, just like when one moves an adjacent 3-5 pair, there is no need to use the full-blown Q-zip algorithm we described above: everything is local, thus only requiring four templates to implement the four movements allowed for a triangle. For longer migrations, however, the same compass idea applies, as explained in Fig. 14. The only additional detail is the connectivity changes that happen when triangles collide. When two oppositely oriented triangles collide, the connectivity becomes regular; when two orthogonally-oriented triangles collide, they form a transient adjacent 3-5 pair. Two same-direction triangles can also collide, in which case a pair of v3 and v5 end up on two opposite sides of a quad, see Fig. 15.

4 RESULTS

As we explained until now, Q-zip is a topological editing primitive that extends local mesh operators such as quad collapse, edge split, or edge rotation to arbitrarily-long topological operators. It can thus be useful in a variety of scenarios involving mesh optimization, varying from global structure optimization of quadrilateral meshes to the quality improvement of mesh elements according to a prescribed quality measure for quads. A typical utilization of Q-zip would explore connectivity changes and decide which pairs of singularities to move in order to improve a predefined "score" for the mesh, such as the number of singularities in the simplest case or more specific and/or complex quality measures in more practical cases. Q-zip operations would also be interleaved with local mesh smoothing in order to improve the mesh quality, in particular around surgical scars, until no further Q-zip leading to improvements can be found. Note that given the fact that our Q-zip operations involve arbitrary long distances, this exploration is much less likely to fall into local minima than if only local topological changes were considered.

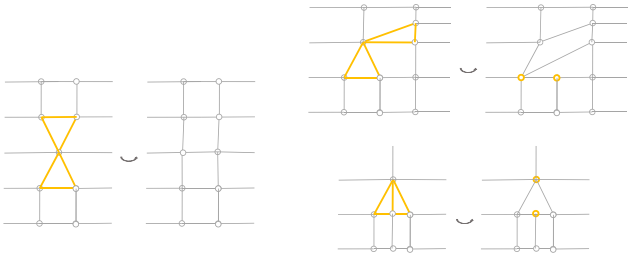


Fig. 15. **Merging triangle pairs into quads.** The orange vertices show the locations of possible singularities after the merging of two triangles.

4.1 Automatic Q-zip batches for connectivity optimization

In order to demonstrate the robustness, usefulness, and efficiency of Q-zip, we implemented an automatic tool which, from an arbitrary quad mesh input, performs repeated Q-zips in order to reduce the number of singularities while improving the isotropy and uniformity of the mesh elements. Countless variants are simple to design depending on which specific application one wishes to target.

Scoring strategy. We implemented a basic scoring approach which evaluates how efficient a Q-zip operation can be, in order to select a batch of them to apply on the mesh at a time. We account for a number of existing strategies to guide the singularity pair movements, as we now detail for completeness:

- *Electric push and pull.* Given the nature of singularities, we consider them as emitting an electric field such they a) attract complementary singularities, and b) repulse similar singularities. This simple strategy favors the merging of v3's with v5's, while keeping v3's (resp., v5's) away from each other to offer better mesh quality (bunched-up singularities would induce high distortion). Additionally, we consider boundaries and sharp features as also weakly repulsing *all* singularities. An electric score is then established by evaluating the distance of a singularity to the three closest singularities and the closest boundary/feature, and assigning a score based on the sum of the inverse of these signed distances (measured in the number of edges). A Q-zip operation that most decreases the electric potential will thus be favored.
- *Isotropy control.* Because we know the local collapses or splits that a Q-zip operation performs, we can anticipate whether the effect of the surgery will improve, or worsen, the isotropy of the elements around the surgery path. Indeed, we can measure the stretch of a current quad along 4 directions (two axis-aligned directions, and two diagonal directions), and deduce whether performing an LC, RC, LS, or LR will improve any of these factors: this is what we call the *isotropy score*. This simple approach helps to decide on which topological surgeries are most relevant based on the current mesh geometry (see Fig. 18 for a resulting effect).
- *Uniform sizing control.* The exact same general idea applies to the control of sizing via a *sizing score*: we seek a uniform sizing, and thus penalize any collapsing surgery in a region already too sparse, or splits in a region already too dense. This is also implemented by simply summing the local sizing score of the surgery paths to provide a general score for a given Q-Zip operation.
- *Separatrix control.* Finally, we also add a score which evaluates whether a pairwise singularity move can help align singularities with nearby separatrices: as noted in multiple previous works, "near misses" created by short distances between a singularity and a separatrix can create distortion issues easily remedied by collapsing this short distance. Our *separatrix score* is simply the inverse square of the distance to the closest separatrix, to favor alignment and thus mesh quality.

Batch of best Q-zips. The combinatorial complexity involved in considering all pairs of irregular vertices calls for heuristics in determining which pairwise singularity movements can efficiently decrease the total sum of our scores so as to reduce the number of irregular vertices while promoting good quad shapes throughout the mesh. In our implementation, we use a multiple-choice strategy, similar in spirit to what was used for mesh decimation in [Wu and Kobbelt 2002]. We first compute the current total mesh score, then randomly pick a batch of s singularities (where s is set to 50% of the number of singularities in our implementation) currently present on the quad mesh. For each of these selected singularities, we explore all the possible Q-zip operations between this singularity and one of its three closest singularities of type v3 and type v5 respectively (which makes for an exploration of 6 surgery paths for a total of 24 possible surgeries); the operation corresponding to the largest decrease in mesh score is stored in a priority list of possible Q-zips. From the best scores we found, we execute the top half of the list, serially, while invalidating any surgery that comes in contact with a region already operated on. (Note that an obvious extension could be to perform these operations in parallel if the surgical paths are kept safely away from each other.)

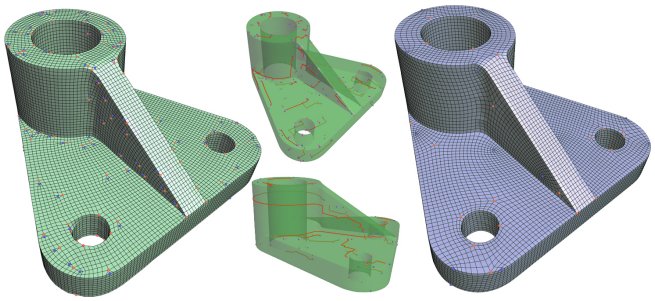


Fig. 16. **Long and winding surgery paths.** Feature alignments are preserved by avoiding paths crossing feature curves. As consequence, this mesh of a mechanical part (left) undergoes long, winding surgery paths (middle, early batch on top, later batch at bottom) before resulting in a more regular structure (right). The mesh went from 154 v3 and 166 v5, to 26 v3 and 42 v5.

Smoothing. Local smoothing is also done after each batch of Q-zips. Mesh smoothing has been extensively studied, and various heuristics could be used. In our implementation, we use a simple Laplacian smoothing performed in the polar geodesic parameterization of the one-ring of each vertex. To favor uniform meshes even around singularities, we use a valence-weighted Laplacian: each edge is considered as a spring, whose stiffness depends on the valence of its extremities — we use 1, 0.5, or 2 when the sum of valence = 8, < 8, or > 8 respectively. A fixed number of iterations is triggered after each Q-zip in the vicinity of the surgery scar.

Boundary defects. When a vertex is on a boundary, we declare it as a topological defect based on its effective valence vs. its discrete

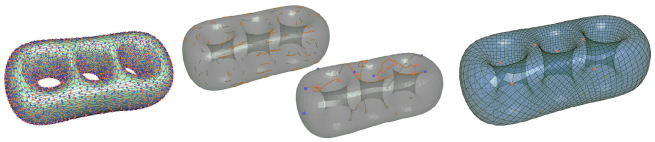


Fig. 17. **Robustness test.** A triangle mesh turned into a quad mesh through Catmull-Clark scheme contains many singularities (left); Applying batches of q-zip exhibits short surgical paths (middle left) at first given the initial density of singularities, while later iterations contain far longer surgery paths (middle right); the final output quad mesh contains the expected and optimal 16 v5 singularities (right).

geodesic curvature. Based on the closest multiple of $\pi/2$ that the sum of tip angles makes at the vertex, we know the ideal valence: for a subtended angle θ, the ideal valence is $\max(2, \left\lceil \frac{\theta}{\pi/2} + \frac{1}{2} \right\rceil)$. The boundary defect is then set to be the effective valence minus this ideal value; for instance, a value of -1 means that we consider this boundary vertex to be akin to a valence-3 vertex when we pair singularities together, in order to favor valences that lead to the least amount of quad distortion on boundaries.

Feature protection. Sharp features, frequently present in meshes of mechanical parts, must be preserved during editing. We can easily accommodate this constraint by extending slightly the "realizable operation" check explained in Sec. 2.5: we make sure during surgery planning that none of the operations along the surgery path will alter a sharp feature. Consequently, we can also prevent a surgery path (by modifying the breadth-first search when computing a path between two singularities) from cutting through a sharp feature as it would systematically fail this check.

Protected regions. A useful extension of this protection of features that we can exploit in our repeated applications of Q-zip is the notion of protected regions: we let the user select entire regions of the mesh where *no mesh changes* are allowed. We then simply disallow our surgery paths to go through these regions. Similar to the notion of binary masks in the seam carving algorithm for image resizing, this will guarantee that no edge splits or quad collapses occur there, thus protecting the flow lines inside these regions. Fig. 21 shows an example, where the edge flows around the eyes, ears, nose, and mouth are kept intact while the rest of the mesh is made more regular. Note that the use of protected regions shows the practical implications of the flexibility of Q-zip and of the proof of invariance up to homotopy of the possible singularity edits: the presence of multiple blocked regions can easily prevent the existence of surgery paths without irregular vertices.

4.2 Evaluation of quad mesh improvements

In order to evaluate this simple mesh improvement approach, we selected different types of inputs. We selected "Instant Meshes" from [Jakob et al. 2015] as one of the few methods (with advancing-front methods mentioned early on) which can create large-scale meshes fast—with the drawback that many spurious singularities may be present. We also chose meshes from [Fang et al. 2018] as they are the result of a more involved optimization process, thus rendering the presence of spurious singularities less likely. In both cases, we found that our approach can easily improve their results as a post-processing with a complexity roughly equal to their vertex count, even without knowing which frame field they used to create the meshes in the first place. For instance, meshes without sharp features like Fig. 1 and Fig. 20 can be cleaned up with a straightforward application of Q-zip batches; the David mesh is rapidly cleaned up after a few batches of Q-zip, but we stopped after 30 batches to avoid simplifying the base structure too much. For meshes involving sharp features like the mechanical part used in Fig. 19, our surgery sketch avoids crossing sharp edges (by treating them as boundaries) to prevent their destruction, restricting the possible set of Q-zip operations and thus reducing the computational time for connectivity exploration. Long surgeries are however frequent in later stages of batch rounds, applying connectivity changes that previous work on singularity pair editing could not handle (see Fig. 16 with a mesh from [Jakob et al. 2015] and Fig. 12 for a mesh from [Fang et al. 2018]): since the mesh has few singularities left, we must have recourse to far-remote pairs of singularities to explore further improvements. We also tried a "robustness test" by picking a triangle mesh of a three-torus, simplifying it to make its triangles

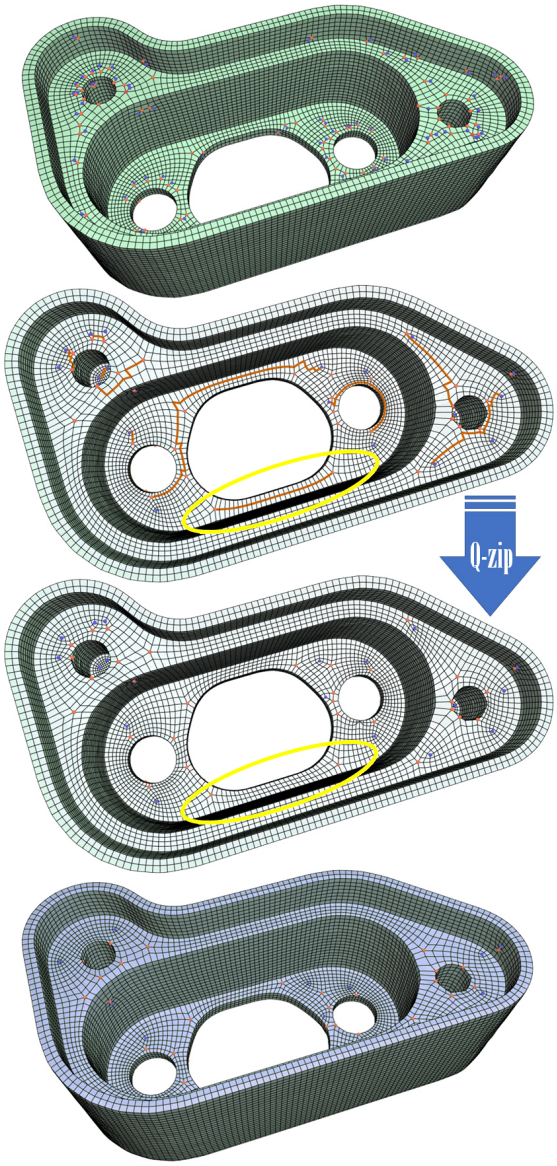
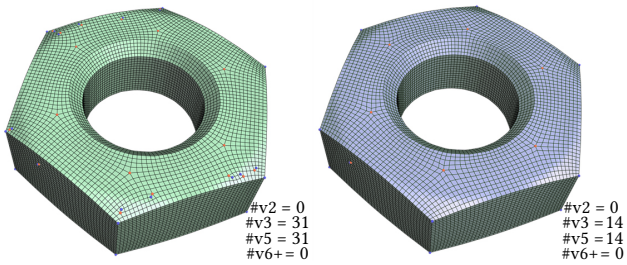


Fig. 18. **Topological surgery guided by mesh density.** By anticipating the effect of topological alterations (splits increasing the density of quads while collapses decreasing it), Q-zip can select narrow passages (see yellow mark) formed by sharp features; here, the split-based surgery automatically chosen by our scoring function turns the narrow region at the bottom into 3-rows of quads while canceling some singularities during a batch of Q-zips, making it more in line with the densities over other narrow regions.



Nut mesh
from [Jakob et al. 2015]

Resulting quad mesh
after 2 batches of Q-zip

Fig. 19. **Nut mesh.** For meshes with sharp features and a few singularities (left, with 11,598 quads), two applications of Q-zip batches are enough to clean up the mesh without major alteration (right, now with 11,563 quads).

quite stretched, then subdivided it into quads to generate a very bad initial quad mesh, see Fig 17. After 12 batches of Q-zip (and longer and longer surgeries), the result is a quad mesh with the lowest expected amount of v3 and v5 vertices. The same robustness test was performed on the fertility model in Fig. 1. Finally, we note that the simple uniform sizing and isotropy control scores we use in our approach can find surprisingly good surgery paths: in Fig. 18, we show one of the later batches of Q-zip, and one of the long operations go right through a two-quad wide region and turn it into a three-wide region, improving the isotropy and uniformity as desired. Such surgeries would be singularly difficult for a human to design, proving the usefulness of Q-zip and of our simple scoring strategy. We note that our mesh improvement tool, for which more results can be found in Fig. 22, does not use *any frame field or sizing field*, as we wanted to illustrate the efficacy of Q-zip as an editing tool rather than producing meshes which optimize a precise quality measure. Different scoring strategies than the ones described in this section are easily formulated and can be equally applied, depending on the application being sought after.

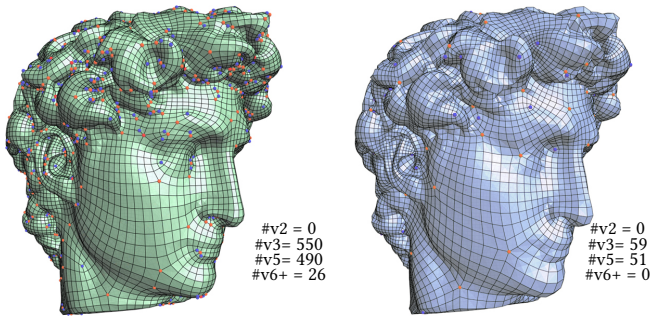
4.3 Timings

Q-zip is easy to implement, and requires so few computations in practice that even long paths are no impediment to the exploration of connectivity changes in terms of computational time. The local BFS searches and trivial edge-to-edge topological operations based on the compass actions can be easily executed on even very large meshes. In fact, in all our examples, the scoring takes over 99% of the overall computational time, the actual surgery sketching, planning, preparation, and execution being therefore negligible. As a means to better illustrate the simplicity of Q-zip, we did some statistics on the rounds of Q-zip batches done on the David mesh of Fig. 20: we found that the average surgery path length was seven edges, and the average timing for a full Q-zip surgery (discounting score evaluation) was 0.5ms on a laptop with a Xeon CPU E5-1620 3.6GHz.

5 CONCLUSION

We introduced in this paper a simple singularity editing primitive and its principled implementation, with which pairs of arbitrarily-distant singularities can be efficiently moved around a mesh by applying a chain of local operations with a minimal footprint. Our approach significantly generalizes the existing state-of-the-art approaches to connectivity editing of quad meshes targeting a better placement of, or a reduction in, singularities [Peng et al. 2011; Bommes et al. 2011; Verma and Suresh 2017; Docampo-Sanchez and Haimes 2019]. The resulting algorithm is offering a general, practical, automated, scalable and reversible approach to performing singularity editing, for which we discussed possible applications when dealing with quad mesh processing. Theoretical contributions extending the work of [Peng et al. 2011] were formulated, proving that our Q-zip automated procedure results in isomorphic meshes for all homological surgical planning paths between a given singularity pair. Parallel-transport of a compass of topological actions was also introduced to guarantee grid-preserving editing without an automaton [Bommes et al. 2011].

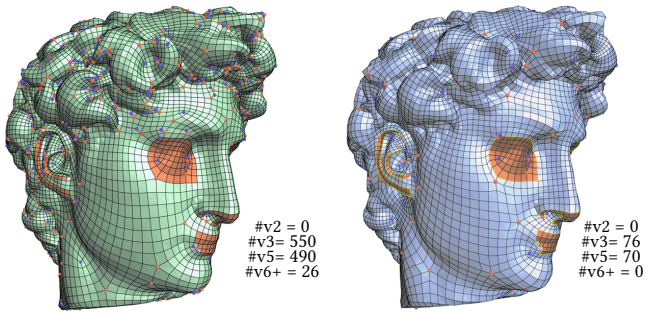
Given the generality and simplicity of Q-zip, we believe it can apply to a slew of geometry processing methods applied to quad or triangle-quad meshes. While the efficiency of Q-zip makes it



David mesh
from [Fang et al. 2018]

Resulting quad mesh
after 20 batches of Q-zip

Fig. 20. **David mesh.** For inputs with a large amount of singularities, we can produce several batches of Q-zip operations to make the mesh more regular with only a small decrease in mesh elements (here, around 10%). However, in one batch process having no input frame field, parts of the mesh may become overly simplified, see around the eyes for instance where the initial placement of singularities is lost. User input, or a more involved scoring strategy, can easily protect regions of the input mesh if needed.



David + protected areas
from [Fang et al. 2018]

Resulting quad mesh
after 15 batches of Q-zip

Fig. 21. **User-input Protection.** Contrary to Fig. 20, the user can first select regions on the mesh to protect, see tawny regions (left); batches of Q-zip will then automatically preserve the local edge flow of these protected regions by simply forbidding surgery paths from entering them.

Fig. 22. **Zoo.** Examples of quad meshes from [Fang et al. 2018], of various sizes, on which a few automatic Q-zip batches were performed based on the heuristic for connectivity optimization described in Sec. 4.1. Irregular vertices are marked as colored balls, and yellow lines indicate sharp features.

particularly attractive for mesh optimization where the effect of a singularity pair motion needs to be evaluated, performed, or undone efficiently (using, for instance, the heuristic approach presented in Sec. 4.1), this editing tool may also allow for a simple and intuitive framework for user-driven mesh optimization, helping artists design low-count quad meshes more efficiently. Finally, we note that Q-zip is technically generalizable to hexahedra. Whether it can become as useful for the beautifying of hexahedral meshes remains to be tested, and is left as future work.

ACKNOWLEDGMENTS

We thank Ameera Abdelaziz for early assistance. We also benefited from technical discussions with Jihong Wang and Jan Frykestig. This work was partially funded by NSF grant III-1900473. M. Desbrun gratefully acknowledges generous support from Inria, Pixar Animation Studios and Ansys Inc.

REFERENCES

Ansys. 2020. Meshing User's Guide. http://www.ansys.com ANSYS, 275 Technology Drive, Canonsburg, PA 15317.

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013. Integer-grid Maps for Reliable Quad Meshing. *ACM Trans. Graph.* 32, 4, Article 98 (2013).

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30, 2 (2011), 375–384.

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2012. State of the Art in Quad Meshing. In *Eurographics State-of-The-Art Reports*.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph.* 28, 3, Article 77 (July 2009), 10 pages.

Edwin E. Catmull and James. H. Clark. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design* 10, 6 (1978), 350–355.

Joel Daniels, Claudio T. Silva, and Elaine Cohen. 2009. Localized Quadrilateral Coarsening. *Computer Graphics Forum* 28, 5 (2009), 1437–1444.

Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. 2008. Quadrilateral Mesh Simplification. *ACM Trans. Graph.* 27, 5, Article 148 (2008).

Julia Docampo-Sanchez and Robert Haimes. 2019. *Towards Fully Regular Quad Mesh Generation.* arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2019-1988

Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C. Hart. 2006. Spectral Surface Quadrangulation. *ACM Trans. Graph.* 25, 3 (July 2006), 1057–1066.

Hans-Christian Ebke, Marcel Campen, David Bommes, and Leif Kobbelt. 2014. Level-of-Detail Quad Meshing. *ACM Trans. Graph.* 33, 6, Article 184 (2014).

Hans-Christian Ebke, Patrick Schmidt, Marcel Campen, and Leif Kobbelt. 2016. Interactively Controlled Quad Remeshing of High Resolution 3D Models. *ACM Trans. Graph.* 35, 6, Article 218 (2016).

Xianzhong Fang, Hujun Bao, Yiying Tong, Mathieu Desbrun, and Jin Huang. 2018. Quadrangulation through Morse-Parameterization Hybridization. *ACM Trans. Graph.* 37, 4, Article 92 (2018).

Allen Hatcher. 2002. *Algebraic Topology.* Cambridge University Press.

Jingwei Huang, Yichao Zhou, Matthias Niessner, Jonathan Richard Shewchuk, and Leonidas J. Guibas. 2018. QuadriFlow: A Scalable and Robust Method for Quadrangulation. *Computer Graphics Forum* (2018).

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

Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover - Surface Parameterization using Branched Coverings. *Computer Graphics Forum* 26, 3 (2007), 375–384.

Yuanyuan Li, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2010. Editing Operations for Irregular Vertices in Triangle Meshes. *ACM Trans. Graph.* 29, 6, Article 153.

M. Lyon, D. Bommes, and L. Kobbelt. 2020. Cost Minimizing Local Anisotropic Quad Mesh Refinement. *Computer Graphics Forum* 39, 5 (2020), 163–172.

Ashish Myles and Denis Zorin. 2013. Controlled-Distortion Constrained Global Parametrization. *ACM Trans. Graph.* 32, 4, Article 105 (2013).

Steve J. Owen, Matthew L. Staten, Scott A. Canann, and Sunil Saigal. 1999. Q-Morph: an indirect approach to advancing front quad meshing. *Int. J. Num. Meth. Eng.* 44, 9 (1999), 1317–1340.

Chi-Han Peng, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2011. Connectivity Editing for Quadrilateral Meshes. *ACM Trans. Graph.* 30, 6, Article 141 (2011).

Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. 2009. Geometry-Aware Direction Field Processing. *ACM Trans. Graph.* 29, 1, Article 1 (2009).

Ahmad A. Rushdi, Scott A. Mitchell, Ahmed H. Mahmoud, Chandrajit C. Bajaj, and Mohamed S. Ebeida. 2017. All-quad meshing without cleanup. *Computer-Aided Design* 85 (2017), 83–98.

Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. 2010. Practical quad mesh simplification. *Computer Graphics Forum* 29, 2 (2010), 407–418.

Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. 2011. Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Transactions on Graphics (SIGGRAPH Asia)* 30, 6 (2011).

Chaman Singh Verma and Krishnan Suresh. 2017. A Robust Combinatorial Approach to Reduce Singularities in Quadrilateral Meshes. *Comput. Aided Des.* 85, C (April 2017), 99–110.

Jianhua Wu and Leif Kobbelt. 2002. Fast Mesh Decimation by Multiple-Choice Techniques. In *Workshop on Vision, Modeling, and Visualization*. 241–248.

Kaoji Xu, Muhammad Naeem Akram, and Guoning Chen. 2020. Semi-Global Quad Mesh Structure Simplification via Separatrix Operations. In *SIGGRAPH Asia 2020 Technical Communications*. Article 2.

Xiaotian Yin, Miao Jin, and Xianfeng Gu. 2007. Computing shortest cycles using universal covering space. *Visual Comput.* 23, 12 (2007), 999–1004.

Muyang Zhang, Jin Huang, Xinguo Liu, and Hujun Bao. 2010. A Wave-Based Anisotropic Quadrangulation Method. In *ACM SIGGRAPH Proceedings*. Article 118.