FlowRep: Descriptive Curve Networks for Free-Form Design Shapes

GIORGIO GORI, ALLA SHEFFER, NICHOLAS VINING, and ENRIQUE ROSALES,
University of British Columbia
NATHAN CARR, Adobe
TAO JU, Washington University, St. Louis

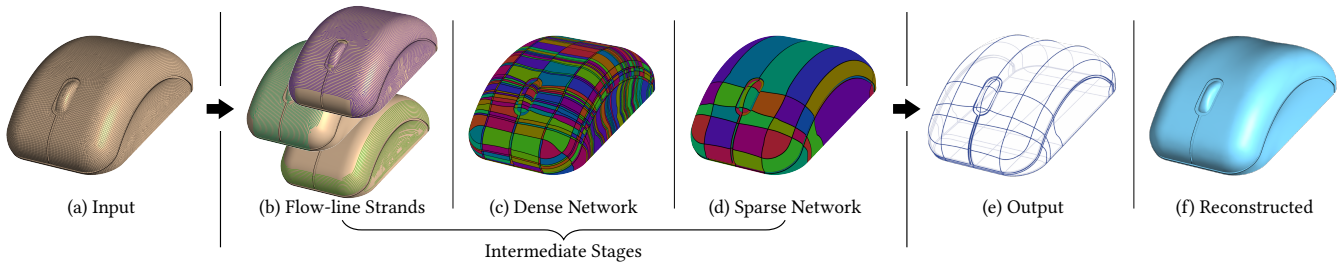


Fig. 1. *FlowRep* describes complex free-form 3D geometries (a) by a compact network of descriptive and projectable curves (e) that can be used to both depict and reconstruct (f) the input (L_2 distance between (a) and (f) is 0.1% of bounding box diagonal). Given an input quad mesh (a) it extracts strands of dominant flowlines (b), uses those to compute a dense descriptive network (c) and then systematically simplifies it to obtain the desired compact net (d).

We present *FlowRep*, an algorithm for extracting descriptive compact 3D curve networks from meshes of free-form man-made shapes. We infer the desired compact curve network from complex 3D geometries by using a series of insights derived from perception, computer graphics, and design literature. These sources suggest that visually descriptive networks are *cycle-descriptive*, i.e their cycles unambiguously describe the geometry of the surface patches they surround. They also indicate that such networks are designed to be *projectable*, or easy to envision when observed from a static general viewpoint; in other words, 2D projections of the network should be strongly indicative of its 3D geometry. Research suggests that both properties are best achieved by using networks dominated by *flowlines*, surface curves aligned with principal curvature directions across anisotropic regions and strategically extended across sharp-features and isotropic areas. Our algorithm leverages these observation in the construction of a compact descriptive curve network. Starting with a curvature aligned quad dominant mesh we first extract sequences of mesh edges that form long, well-shaped and reliable flowlines by leveraging directional similarity between nearby meaningful flowline directions. We then use a compact subset of the extracted flowlines and the model's sharp-feature, or trim, curves to form a sparse, projectable network which describes the underlying surface. We validate our method by demonstrating a range of networks computed from diverse inputs, using them for surface reconstruction, and showing extensive comparisons with prior work and artist generated networks.

CCS Concepts: •**Computing methodologies** → **Shape analysis; Non-photorealistic rendering;**

Additional Key Words and Phrases: Shape representation, line rendering, sketch based modeling, shape abstraction

0730-0301/2017/7-ART59 $15.00
DOI: http://dx.doi.org/10.1145/3072959.3073639

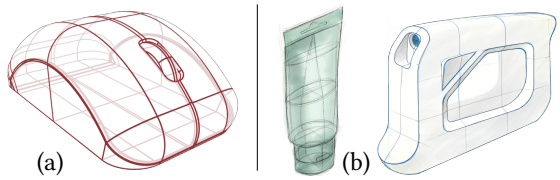


Fig. 2. Artist generated 3D (a) and 2D (b) descriptive curve networks succinctly convey complex free-form shapes.

ACM Reference format:
Giorgio Gori, Alla Sheffer, Nicholas Vining, Enrique Rosales, Nathan Carr, and Tao Ju. 2017. FlowRep: Descriptive Curve Networks for Free-Form Design Shapes. *ACM Trans. Graph.* 36, 4, Article 59 (July 2017), 14 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073639

1 INTRODUCTION

Artists employ sparse descriptive networks of 3D curves lying on the surface of imagined objects as a starting point for modeling such envisioned shapes, with curve generation followed by surfacing, and quickly communicate 3D shapes on paper by sketching 2D projections of such 3D curve networks depicted from informative viewpoints (Figure 2). In addition to providing an effective visual communication tool, sparse descriptive curve-network representations of 3D models provide designers with intuitive handles for shape editing [Gal et al. 2009]; facilitate compact shape representation and abstraction [Mehra et al. 2009]; support shape-preserving mesh simplification [Gehre et al. 2016]; and enable other high-level operations. We propose *FlowRep*, a method for computing visually descriptive compact curve networks of free-form man-made, or designed, shapes from existing models (Figure 1). FlowRep networks effectively convey the shape of input models to human observers and can be used for the range of applications above. They accurately encode input geometry enabling both perceptual and geometric reconstruction; the mouse in Figure 1f was accurately reconstructed from our curve network alone using the method of [Pan et al. 2015].

While previous methods have addressed the related problems of surface partitioning [Bnire et al. 2013; Campen et al. 2012; Cohen-Steiner et al. 2004; Eppstein et al. 2008; Myles et al. 2014] and extraction of sets or networks of different feature curves [DeCarlo et al. 2003; Gehre et al. 2016; Mehra et al. 2009], the partition boundaries or sets of curves they produce do not provide a detailed description of complex man-made free-form shapes (Section 2); the compact sets of curves they generate either do not allow a human viewer to visualize the detailed 3D shape that they represent, or are not sufficient to reconstruct the shape using existing surfacing methods. In contrast, and as demonstrated by our comparisons, our method addresses the fundamental problem of generating a curve network that unambiguously defines the target shape for human observers and allows for effective reconstruction of the shape from the curves alone using perception-driven surfacing techniques [Bessmeltsev et al. 2012; Pan et al. 2015].

The first challenge that we face in extracting the desired curve networks from input models is determining the properties these networks must possess to adequately describe a given shape. Design literature points to two sets of criteria that these curve configurations should satisfy (Section 3). The desired curve networks should be *projectable* - i.e. the 3D shape of the curves should be predictable from their 2D projection when viewed from non-accidental viewpoints - and the network cycles should clearly *describe* the surface regions they bound. As observed by previous literature [Pan et al. 2015; Xu et al. 2014], artist drawn networks are typically dominated by a combination of trimming curves, which indicate sharp features, and *flowline* curves, or surface curves aligned with principal curvature directions in anisotropic regions and smoothly extending into and traversing isotropic areas. These flowlines are key to both network projectivity and cycle descriptiveness (Section 3). By construction, curvature tensor-aligned curves are orthogonal; this is a key property for recovering their 3D shape from a 2D projection of the curve network [Xu et al. 2014]. Moreover, human observers tend to mentally surface 3D network cycles by interpreting most cycle curves as aligned with principal curvature directions on an imaginary surface. They consequently envision surfaces on which the curvature directions are interpolating the directions of these curves and the curvature magnitudes are a blend of the curvatures along these curves [Bessmeltsev et al. 2012; Pan et al. 2015]. When extending flowlines across isotropic regions, artists leverage the extra degree of freedom these regions provide to optimize both curve projectivity and descriptiveness. While dense flowline and trim networks adequately describe shape, design literature indicates a strong preference for using *compact*, minimalist, cycle-descriptive networks to avoid visual clutter [Eissen and Steur 2008]. When creating such compact networks, artists use *dominant* flowline curves to delineate regions with monotone curvature variation, whose geometry is consequently well described by their boundaries. Using these guidelines for creating the desired descriptive networks, we seek to compute a compact descriptive set of projective dominant flow lines on the input shapes. To extract this compact network, we leverage observations about the desired properties of such curves derived from design and modeling literature (Section 3) and use those to quantify dominance, descriptiveness and projectivity. We then employ these definitions in a network computation algorithm.

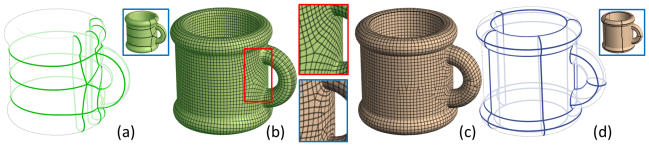


Fig. 3. Quad-meshing methods that optimize for mesh regularity, such as [Bommes et al. 2013], use quad partitions (a) as a starting point and often exhibit systemic misalignment with curvature directions as highlighted in (b). Meshing methods that seek to adhere to curvature directions more strictly often result in meshes with multiple sporadic singularities and non-quad elements (c). To avoid systemic curvature misalignment we use the latter type of meshes as a starting point for generating descriptive curve networks (d).

Tracing individual flowlines, especially across isotropic regions and near curvature-field singularities, is inherently unreliable. We provide global context for flowline computation by using a curvature aligned quad-dominant mesh as a starting point for our algorithm. Edge sequences on such meshes are largely aligned with curvature directions in anisotropic regions and typically smoothly extend across isotropic ones; such edge sequences provide a natural starting point for tracing an initial set of flowlines from which we can subsequently distill the desired dominant subset.

Using such meshes as a starting point, however, introduces different challenges. First, quad-meshing methods balance mesh quality and vertex regularity against curvature alignment. Generation of more regular meshes, e.g. [Bommes et al. 2013; Campen et al. 2016], often requires significant deviation from curvature directions (Figure 3b). In our setting, the requirement for curvature field alignment is paramount, necessitating the use of curvature aligned, but potentially highly irregular, meshes with singular vertices and non-quad faces (Figure 3c). We extrapolate projectable and dominant flowlines, overcoming misaligned edges and mesh irregularities, by leveraging directional affinity between adjacent meaningful flowlines. We note that dominant principal curvature directions are characterized by clusters, or *strands*, of adjacent similarly directed flowlines or sequences of mesh edges. Our flowline extraction method first clusters the mesh edges into strands, and then extracts individual flowlines from these strands. We use the extracted flowlines to compute the desired network. We first assemble a dense descriptive trim and flowline network that describes the input surface within a given tolerance, and then simplify and optimize this network using the dominance, projectivity and descriptiveness metrics identified above to obtain the desired compact solution.

Our contribution is two-fold. We identify and enumerate the key properties of descriptive curve networks suitable for communicating complex designer shapes; we then propose the first curve network extraction algorithm that generates networks with these desired properties. We test our method on a diverse range of inputs, showcasing its ability to generate the desired results on complex free-form models, and validating it through comparison to artist outputs, designer evaluation, and comparisons to prior art (Sections 8, 9). As this validation confirms, our networks successfully capture and convey the essence of the input shapes both in 3D space, and when viewed from general viewpoints.

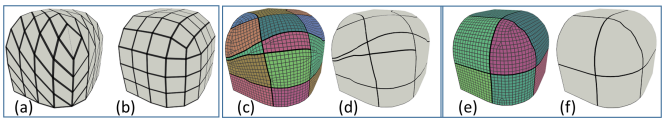


Fig. 4. Projectivity and cycle descriptiveness (all renders show the same 3D model in same view). Projection of an orthogonal quad mesh (b) conveys the underlying 3D geometry better than that of a non-orthogonal mesh (a). A flowline network (f) over a curvature aligned field (e) succinctly describes the surface, while a network (d) generated from an arbitrary smooth cross-field (c) does not.

2 RELATED WORK

While our focus on computing perceptually descriptive curve networks for design shapes is new, our method is related to a range of works that seek to either partition surfaces, or to extract different types of feature curves from an input mesh.

Curvature Aligned Meshes. At the finest level, curvature aligned polygonal meshes, e.g. [Alliez et al. 2003], provide two of the qualities we seek: descriptiveness and projectivity (Figure 4b). However, they are clearly not compact. We use such meshes as a starting point for our framework, which compacts them while maintaining these two key properties (Figure 4f).

Mesh Segmentation and Reverse Engineering. Methods for mesh segmentation, e.g. [Cohen-Steiner et al. 2004; Julius et al. 2005] and reverse engineering, e.g. [Bnire et al. 2013; Nieser et al. 2010; Wu and Kobbelt 2005] aim to segment models into regions with particular surface characteristics, for instance developable surfaces, planes, or conics. They pay minimal attention to the properties of the boundary networks that arise from the partition they produce, and at best optimize for boundary straightness or compactness. While these methods facilitate algorithmic reconstruction of approximate input geometry from the curve network and compactly encoded region descriptors (such as surface type, axis of revolution or radius), the curve networks they generate are often not cycle-descriptive and, when projected to 2D space, provide little information on the originating 3D curves (Figure 13). The key distinguishing feature of our method compared to these approaches is our focus on network, rather than region, properties, and consequently the ability to concisely and effectively describe input surfaces independent of their specific geometry using network curves alone.

Quad Patch Layout. A range of methods extract coarse quad patch layouts to facilitate parameterization and surface fitting. Early methods, such as [Bommes et al. 2008] generate such layouts semi-manually; this approach is still popular [Campen and Kobbelt 2014; Zhuang et al. 2014]. More recent frameworks use motorcycle graphs, starting at quad mesh singular points [Eppstein et al. 2008] to obtain singularity-free quad patches. Recent works [Bommes et al. 2011; Tarini et al. 2011] improve the patch layouts by simplifying spiraling patch boundaries and merging nearby singularities. Gunpinar et al. [2014a; 2014b] augment the graph tracing with geometric considerations, leading to better capture of prominent features. Alternative approaches use a curvature-aligned tensor field instead of a mesh as a starting point [Bommes et al. 2013; Campen et al. 2012; Myles et al. 2014; Razaflndrazaka et al. 2015] and trace separatrices from field singularities to form quad patches. Whether starting from a quad mesh or a field, the results of these methods are highly dependent on the location of singularities and can therefore be significantly misaligned with curvature directions (Figures 3,14). Even when aligned with curvature the separatrice boundaries used by such networks form irregular valence vertices, resulting in networks that contain few orthogonal intersections, making the mental leap from the network to the underlying surface challenging (Figure 14a). Similar to the first group of layout methods, we use meshes as a starting point; however, we seek a distinctly different set of network curves, one aimed at perceptual rather than purely geometric approximation of the input model.

Feature Curves and Curve Networks. A large body of research addresses detection of both sharp features and prominent curves, such as ridge and valley lines on meshes, e.g. [Hildebrandt et al. 2005; Lai et al. 2007]. These curves are often used to augment contours when generating sketches of input models, e.g. [DeCarlo et al. 2003], and are effective at quickly conveying the overall shape of objects [Cole et al. 2008; Eissen and Steur 2008] (Figure 12b). When aiming to convey proportions and geometric details, artists utilize more detailed descriptive, or "precise", drawings [Eissen and Steur 2008] which augment contours and sharp feature curves with dominant flowlines and typically do not include smooth ridges or valleys (Figure 2). Our work focuses on capturing the curve networks artists use in the latter context (Figures 11, 12c).

Mehra et al. [2009] represent 3D shapes using networks of sharp feature curves with specified normals along them, selecting a suitable curve subset to achieve a desired abstraction. To describe smooth regions, they augment those networks to include the boundaries of coarse planar segmentation regions [Cohen-Steiner et al. 2004] (Figure 13b). Gehre et al. [2016] allows for a more effective control of network density using a global scale parameter and support inclusion of other feature curves, such as ridges and valleys in the output networks. As noted earlier, feature curves alone are not sufficient to accurately describe smooth shapes, while segmentation boundary networks are rarely projectable and are therefore not suitable for our needs. Moreover, while the network resolution constraints these methods use are largely spatial, FlowRep network density is controlled by its descriptiveness - resulting in much denser spacing on prominent details, such as the mouse wheel, and sparser ones on more monotone areas such as the top of the mouse (Figure 1).

Shape Proxies. de Goes et al. [2011] propose a user-assisted method for coarse abstraction of natural shapes. They first segment models into roughly convex parts, and then partition those using an extension of VSA [Cohen-Steiner et al. 2004] that seeks to reduce T-junction count. Our automatic framework targets free-form man-made shapes and is designed to accurately capture their geometry (Figure 15, top).

Planar curves, or slices, aligned with major curvature direction or key symmetry planes are successfully used to fabricate real-life proxies of 3D shapes [Cignoni et al. 2014; McCrae et al. 2011]. This representation requires a leap of imagination to envision the intended surface, and performs worst when curvature streamlines are non-planar (Figure 15,e,h), leading Cignoni et al. [2014] to use

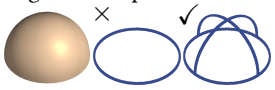
hundreds of slices to obtain recognizable representations of medium complexity shapes. Our alternative approach effectively describes shapes of similar complexity with just a few curves (Figure 15,f,i).

Analysis of Design Networks and Drawings. There is a growing body of work on recovering 3D information from professional design drawings [Iarussi et al. 2015; Shao et al. 2012; Xu et al. 2014] and surfacing of 3D designer networks [Bessmeltsev et al. 2012; Pan et al. 2015]. This line of work has been the catalyst for our exploration of the inverse problem - extracting a descriptive network from a given 3D geometry. We discuss the insights we derive from these papers and which we utilize in our work in Section 3.

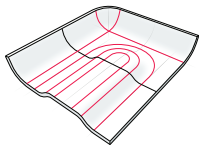
3 DESCRIPTIVE CURVE NETWORKS

Our goal is to compute a sparse network of curves on the surface of an arbitrary 3D model that succinctly describes the model's shape (Figure 2). Based on observations from design tutorials and relevant perception and computer graphics literature, we identify and formulate three major sets of geometric criteria that jointly determine the overall network effectiveness: cycle description, curve dominance, and network projectivity.

Cycle Description. A curve cycle is a collection of curve segments that demarcate the boundary of a single surface patch, whether that surface is a real surface or merely implied. Perceptual studies [Stevens 1981], validated by recent modeling research [Bessmeltsev et al. 2012; Pan et al. 2015], suggest that, when shown a curve cycle consisting of several smooth curve segments, human observers opt for a unique mental interpretation of the cycle's implied interpolating surface. Specifically, viewers tend to perceive most of the provided curves as representative curvature lines on an imaginary underlying surface. They subsequently imagine a surface whose principal curvature lines smoothly blend these curves. Since surface principal curvatures fully define its geometry, viewers consequently imagine a unique surface interpolating this cycle.



Observation of industrial design practices [Bordegoni and Rizzi 2011; Eissen and Steur 2008] indicates that artists leverage this property when creating 3D curve networks or depicting 3D geometry in 2D space. Their networks are dominated by flowline curves aligned with principal curvature directions in anisotropic areas and extended across isotropic regions and are augmented by *trimming* curves which demarcate open boundaries and sharp features. As demonstrated in the inset, for the surface on the left the trimming curve alone (middle) incorrectly conjures a flat surface, while the cycles of the rightmost network are descriptive of the originating surface on the left.



Flowline Dominance. Design literature [Bordegoni and Rizzi 2011; Eissen and Steur 2008] highlights the need to keep the number of network curves minimal for aesthetic reasons. It consequently provides helpful guidelines as to what subset of flowlines and trimming curves is *dominant*, or best at succinctly conveying surface geometry. This literature indicates a preference for using flowlines which delineate large areas of monotone curvature, and are representative of one of the curvature

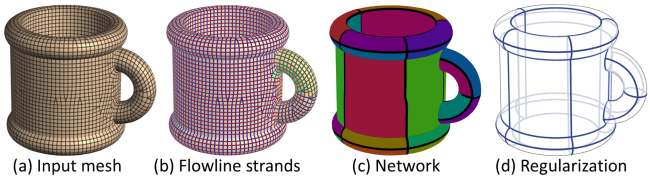
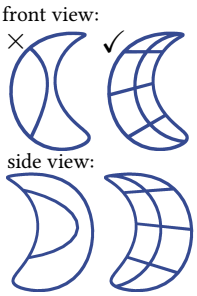


Fig. 5. Algorithm stages: (a) Input quad-dominant mesh; (b) flowline strands; (c) compact descriptive network; (d) regularized final network.

directions within these areas. As a specific example, Eissen and Steur [2008] recommend that artists demarcate roundings (marked in red in the inset). They also recommend using "*bigger sized* curves first" and adding more curves as necessary "to emphasize the transformation of the surface". This advice suggests a preference for hierarchical network constructions - first capturing major anisotropic regions by tracing their dominant principal curvature streamlines, and then refining the network to add finer details.



Network Projectivity. Artist-generated descriptive curve networks are designed to serve as a self-sufficient proxy of the 3D shape. Consequently, evidence indicates that artists construct networks whose 2D projections in many, if not all, views can be used by human observers to successfully predict their 3D shape [McCrae et al. 2011]. In the inset the network on the right satisfies this property, while the one on the left does not. Perception research [Stevens 1981] points out that smoothly crossing 2D curves in design drawings are universally perceived as *orthogonal*. As highlighted by [Shao et al. 2012; Xu et al. 2014] this cue is critical when extrapolating depth from line drawings. Artists ubiquitously employ such crossing orthogonal 3D curves in their networks. Curves meeting at T-junctions are similarly perceived as likely orthogonal, unless contradicted by the surrounding context [Xu et al. 2014]. There is no indication in research that any other type of intersection or curve ending contributes to viewer understanding of 3D network geometry given a 2D projection. These observations suggest a preference for orthogonal curve networks dominated by regular (valence-4) vertices, with no open-ended curves.

Human observers are more successful at inferring 3D curve shape from 2D projections of flatter curves [Stevens 1981; Todd and Reeichel 1990]. While restricting the set of network curves to strict planes reduces the set of models one can effectively describe [Xu et al. 2014], artists are strongly encouraged to use *planar* curves when depicting shapes [Eissen and Steur 2008], whenever possible.

Lastly, artists are strongly encouraged to draw local symmetry, or *geodesic* curves when depicting complex surfaces [Eissen and Steur 2008, 2011]. The symmetry cue is known to be helpful in recovering the 3D shape of network curves from a 2D view [Shao et al. 2012; Xu et al. 2014].

4 ALGORITHM OVERVIEW

Based on the criteria outlined above, we seek to construct a sparse cycle-descriptive network of trim curves and projective dominant

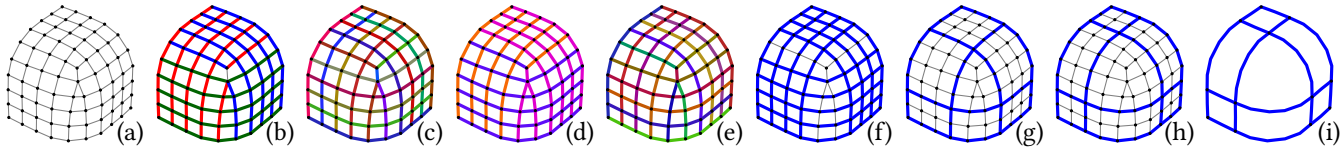
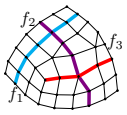


Fig. 6. Detailed algorithm overview: (a) Input quad-dominant mesh; (b) initial flowline strands; (c) initial conservative flowlines; (d) final flowline strands; (e) reliable flowlines; (f) dense descriptive network; (g) simplified network; (h) network post local optimization; (i) regularized final network.

flowlines. We seek flowlines that are aligned with curvature directions in anisotropic areas, and which smoothly extend across features and isotropic regions.

To make the problem tractable, we discretize the solution domain by starting from a finite set of potential flowlines. While one could start from a network constructed by directly tracing on a smooth curvature-aligned tensor field, tensor field tracing raises numerous accuracy issues [Myles et al. 2014; Ray and Sokolov 2014] and requires the consideration of subtle streamline seeding and termination choices [Campen et al. 2016; Myles et al. 2014]. We note that existing methods for generating curvature-aligned quad-dominant meshes robustly address these issues, and that their outputs can provide a suitable starting point for our method. Most of the edges in anisotropic regions on such meshes are, by design, aligned with curvature directions, and most edges in isotropic regions are aligned with a smooth extension of the curvature field. Moreover such meshes, when dense enough, satisfy both projectivity and cycle-descriptiveness (Figure 3c, Figure 4e); our task can therefore be formulated as extracting a compact subset of the mesh edges which maximally retains both properties. We control the trade-off between compactness and descriptiveness by imposing a bound on the cycle-descriptiveness error, and optimizing for the most compact dominant and projective edge network that satisfies this bound.

In this section, we give a high-level formulation of the problem and overview the key components of our method. Details of the formulation and method will be discussed in the next three sections.



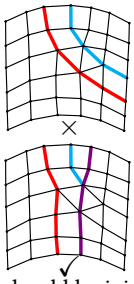
Problem Statement. Given an input quad-dominant mesh M (Figure 6a), we formulate the computation of our target network N as selection of a subset of mesh edges with the following properties. We define a *flowline* as a (possibly closed) path made up of a sequence of vertex-adjacent edges (see inset; three flowlines f_1, f_2, f_3 are identified by three different colors). We associate positive projectivity and dominance costs p_f and d_f with each flowline f. These costs are designed to decrease as a flowline's projectivity or dominance increases. We also associate a descriptiveness error d_c with each network cycle c The exact formulations of p_f, d_f and d_c are detailed in Section 5. Using these measures, our discrete optimization goal can then be formulated as computing a connected set of network flowlines f that minimizes network cost while satisfying a descriptiveness threshold:

$$\begin{aligned} \min E_N &= \sum (p_f + d_f) \\ \text{subj. to } &\max d_c < d_{\max} \ \forall c \in N \end{aligned} \tag{1}$$

Here $d_{\max}$ is a user specified descriptiveness threshold that controls the network sparsity.

Two key properties make this problem distinct from those addressed by traditional discrete mesh segmentation frameworks. First, our network computation operates on two distinct sets of entities. While our optimization function is defined on flowlines, or sequences of mesh edges, our constraints are defined on the cycles, or patches of mesh faces, that the flowlines bound. Second, unlike classical segmentation frameworks, the function we seek to optimize is essentially independent of the number of edges within each selected flowline, but highly dependent on the *a priori* unknown number of network flowlines and their overall properties.

Solution Framework. We obtain the desired output network, by using the observation that an assignment of mesh edges to flowlines can be made largely independently of the choice of which flowlines will be eventually used in our final network. We therefore first group sequences of edges into flowlines (Figure 6b-e) and then select a subset of these flowlines, in combination with the surface trim curves, to assemble the desired network (Figure 6f-h). We contrast our approach against classical segmentation methods in Section 8.



Strand-based Flowline Extraction. In general we expect pairs of similarly directed edges that share a common vertex to belong to the same flowline and orthogonal adjacent edges to belong to different flowlines. We use these criteria to determine when flowlines should terminate, i.e. under which conditions adjacent edges should belong to different flowlines, and when local geometry allows for multiple alternatives to determine which pairs of adjacent edges should be joined into the same flowline. Due to meshing artifacts, singularities, misaligned edges, and inaccuracies in curvature field computation, making these choices based on purely local geometry around the edges in question (inset, top) can introduce flowlines misaligned with dominant flow directions.

We obtain flowlines aligned with dominant flow (inset, bottom) by taking global context into account. We note that dominant and meaningful curvature cross-field directions on the surface are characterized by groups of multiple long, adjacent, similarly directed streamlines, or streamline *strands*. We leverage this behavior by first extracting similarly directed flowline strands (Figure 6b-d) and then using these strands to extract individual, reliable flowlines (Figure 6e). We do not know *a priori* the number of strands we seek; however, as noted earlier, we generally expect consecutive edges across vertices and opposite edges within quads to belong to the same strand if they have similar directions, and expect orthogonal edges and intersecting flowlines to belong to different strands. We use these observations to formulate strand extraction as a correlation clustering problem [Bansal et al. 2004] (Figure 6d). We then use these strands to extract *individual, reliable* flowlines (Section 6, Figure 6e).

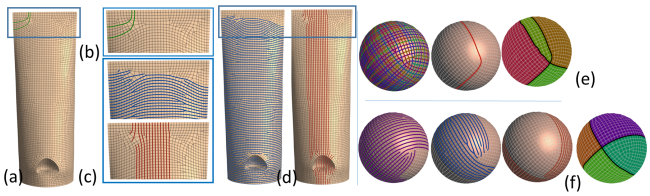


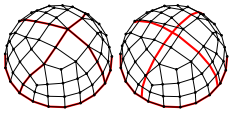
Fig. 7. Independently formed flowlines (a,b,e) can be sub-optimal and may occasionally persist through network computation (e). Strand computation (c,d,f) correctly splits edges between different strands overriding purely local alignment and resulting in better final networks (f).

Figure 7 demonstrates the differences between the local and global approaches on real-life inputs.

Network Assembly. Selecting an optimal subset of the computed flowlines requires solving a discrete constrained optimization problem within a large solution space. Adding flowlines into a network individually is problematic. While humans can easily identify dominant curvature streamlines i.e. surface curves across which curvature changes non-linearly, algorithmically identifying such locations on a mesh is error prone. Absent this information, dominance is best assessed in the context of an existing network, where it can be directly evaluated by comparing the impact on cycle-descriptiveness of removing individual flowlines from the network. Intuitively, keeping more dominant flowlines results in a more cycle-descriptive network. Using all flowlines at once to first form a dense network, and then simplifying it by gradually removing flowlines, provides an adequate solution but is computationally expensive as it involves multiple redundant insertion and removal operations.

To efficiently compute the desired network we adopt a mixed top-down/bottom-up strategy. Our network construction process starts from a minimal network of only trim curves (Figure 10a) and progressively refines the inadequately described cycles on this network (Figure 10, Section 7.1). At each refinement stage we add all flowlines that span, or cross, these cycles into the network (Figure 10b), delaying the decision on which of them are best until the network is sufficiently descriptive. Refinement terminates once all network cycles are sufficiently described; specifically, when the description error d_c for each cycle is below our threshold $d_{\max}$ (Figure 6f).

At this point we have sufficient context to proceed with the simplification process, and can remove redundant flowlines (Section 7.2, Figures 5c, 6g, 10c). We greedily remove less dominant and less projectable flowlines while enforcing the descriptiveness threshold. We then further reduce the network energy E_N, subject to our descriptiveness constraints, by reassessing local flowline selection (Figure 6h). While this combined process is not guaranteed to converge to a global minimum, it works well in practice, resulting in networks of similar complexity to those produced by artists.



Post-Processing. The network produced by the framework discussed so far is constrained by the underlying mesh discretization. This can lead to sub-optimal wiggles along flowlines and some approximately orthogonal, rather than strictly orthogonal, flowline intersections (inset, left). As a post-processing step, we eliminate these artifacts by directly optimizing flowline geometry. We first use an iterative Gauss-Seidel smoother which straightens flowlines while maintaining and improving flowline orthogonality at their intersections. Specifically, for each interior flowline vertex we project its neighbors to its tangent plane and the vertex toward the average of these projections. We relocate flowline intersections by applying the angle equalizing mesh formula proposed by [Surazhsky and Gotsman 2004]. The newly computed positions are projected to the input surface at each step. Finally, we detect all near-planar flowlines (a flowline is near-planar if every point on the flowline is less than half of the average mesh edge length from its best-fit plane, computed via least squares) and make them strictly planar, and straighten all near-linear flowlines (using the same distance threshold, but tested against the best-fit line) (Figure 6i).

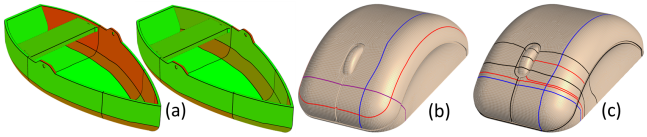


Fig. 8. (a) More (green) and less (red) well described cycles on a row-boat, before and after local optimization. (b) Flowlines colored by decreasing projectivity (blue to red). (c) More (blue) and less (red) dominant flowlines.

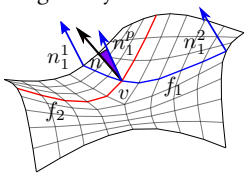
The following sections describe in detail the measures used in our problem formulation (Section 5), and the two stages of our solution framework (Sections 6, 7). Pseudocode for our solution framework is given in the appendix.

5 MEASURING NETWORK PROPERTIES

Our framework seeks to solve the optimization problem described by Equation 1. We now derive the formulas used to measure the optimized energy and constraints. To enable meaningful combination of different values we express all quantities as angles (measured in degrees).

5.1 Cycle Descriptiveness.

Our descriptiveness metric assesses how well a given network curve cycle describes the region it surrounds (Figure 8a). Intuitively, we wish to measure how well the curvature directions and magnitudes in the interior of a region can be reproduced from the magnitudes and directions along the boundary. However, estimating curvature differences is sensitive to different scales in high versus low curvature regions. To enable conservative and robust descriptiveness computation we use differences in normals as proxy for curvature changes: in particular, we measure the angle differences between real normals across the patch and ones predicted based on normals along the cycles.



We predict interior normals based on the boundary by locally mimicking the method of Bessmeltsev et al. [2012]. Specifically, for each vertex v within a region we locate flowlines that cross this

vertex and intersect the boundary curves of the region (see inset). For each flowline f_i we use the normals at the cycle intersections n_i^1 and n_i^2 to obtain a normal prediction at the vertex

$$n_i^p = \frac{d_1}{d_1 + d_2} n_i^1 + \frac{d_2}{d_1 + d_2} n_i^2$$

where d_i is the distance along the flowline from v to respective intersection point. To be conservative we measure local descriptiveness error as the maximum normal deviation between per flowline predictions and the actual normal

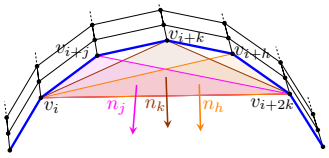
$$d(v) = \max_i(\angle(n_i^p, n)).$$

In theory we could extend this mechanism to handle vertices with no crossing flowlines, e.g. by computing some directed paths from them to the boundary. However, given that the regions we process are sufficiently well spanned by flowlines, and given that the normals within them change gradually, we found that it is safe to simply omit all such vertices from our region descriptiveness computation.

We seek a conservative descriptiveness estimate. Thus we refrain from using vertex descriptiveness average as the per cycle value, yet we also wish to avoid using the worst value as it may be an outlier. To discard outliers, rather than using the largest angle $d(v)$ as the per-cycle error d_c, we set d_c to the 90th percentile value. Lastly, if a non-planar region has more than one boundary cycle , we by default consider it poorly described, setting $d_c = 90°$.

5.2 Flowline Cost.

We assign a cost $p_f + d_f$ to each flowline based on how its presence impacts the descriptiveness d_f and projectivity p_f of the network N.



Projectivity. We measure raw flowline projectivity by locally evaluating its planarity, its deviation from local geodesics, and its connections within the network. For each flowline f we use short odd-length sliding sequences of vertices $v_i, \ldots v_{i+k}, \ldots, v_{i+2k}$ to assess planarity and geodesicity (we use $k = 5$). We fit planes to triplets of vertices v_i, v_{i+j}, v_{i+2k} $j \in [1, 2k-1]$. We then measure sequence planarity as the average angle between the normal of the central plane n_k and those of the other planes n_j fitted to the sequence:

$$P_i = \frac{1}{k-1} \sum_{j \in [1,2k-1], j \neq k} \angle n_j n_k.$$

The planarity of the entire flowline is in turn the average of local sliding sequence planarity costs:

$$P(f) = \frac{1}{|i \in s|} \sum_{i \in s} P_i \tag{2}$$

We recall that a curve is locally considered a geodesic if its local fitting plane contains the surface normal. We thus measure geodesicity by evaluating the angle between the normal to the surface n at v_{i+k} and the plane p_k. Strictly speaking, geodesicity is a boolean property - in a continuous setting, a curve is either a geodesic or it is not. Thus our measure becomes meaningless above a certain angle, leading us to compute geodesicity as

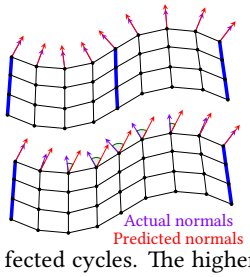
$$G(f) = \frac{1}{|i \in s|} \sum_{i \in s} \min(\angle n_k n(v_{i+k}), G_m) \tag{3}$$

where $G_m = 15°$.

As observed earlier, the interaction between network curves, specifically their intersections, plays a major role in the projectivity of the overall network. When interpreting the geometry conveyed by the network, human observers leverage orthogonal crossings both between flowlines and between flowlines and trimming curves. We therefore prioritize retaining flowline curves that form such crossings by associating *endiness* costs $E_1(f)$ and $E_2(f)$ with the ends of open flowlines. We use an endiness value of 30 for flowline/trimming-curve T-junctions, and 60 for all others. Both values are set to zero for a closed loop.

We set flowline projectivity p_f as follows, weighing geodesicity by $d_{\max}/G_m$ to bring all values to a common scale,

$$p_f = P(f) + \frac{d_{\max}}{G_m} G(f) + E_1(f) + E_2(f) \tag{4}$$



Flowline Dominance. Assessing dominance by measuring curvature continuity, as suggested by design literature, is unreliable in a discrete setting. Instead, we observe that a flowline's dominance within a network context can be evaluated by measuring the impact of removing this flowline on the descriptiveness of the affected cycles. The higher the resulting descriptiveness error, the more dominant the flowline. Recalling that the dominance error is computed as a maximum along two spanning flowline directions, we note that removing a network flowline impacts only one of these directions (see inset). Accordingly, to better pinpoint the impact of each network flowline, we use a modified cycle descriptiveness metric when assessing the impact of the removal. For each cycle resulting from removing a flowline f, we only consider the differences between the predicted and actual normal for normals predicted using flowlines crossing f, and ignore the differences along other flowlines. We then use the maximum of the cycle errors as a dominance estimate $D(f)$. Since we want the cost to be smaller the more dominant a flowline is, we use

$$d_f = 360 - D(f) \tag{5}$$

as the cost. Figure 8 visualizes some flowlines colored based on projectivity and dominance. Note that these costs are only meaningful in the context of a network. Thus when computing flowlines prior to network construction, we require proxy values to predict flowline properties, as discussed in the next section.

6 FLOWLINE COMPUTATION

The goal of the flowline computation stage is to generate a set of *reliable* flowlines which we can use to assemble our network (see Section 7). Since purely local reasoning about individual flowline formation is unreliable we employ a global approach that leverages directional similarity between edges associated with adjacent

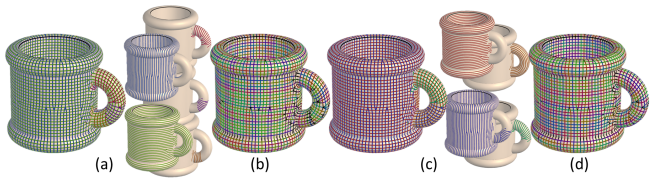
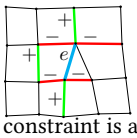


Fig. 9. Dominant flowline strands on the mug: (a) initial edge strands and (b) extracted conservative flowlines; (c) final strands and (d) flowlines.

flowlines (Figure 7). We first form strands, or clusters, of similarly directed edges , and then extract individual flowlines from these strands. When forming strands we employ the following positive (likely to be in same strand) and negative (likely to be in different strands) cues. Edges are expected to belong in the same strand if they are roughly parallel, and either share common vertices or lie on opposite sides of common quads (see inset).



They are expected to belong to different strands if they share common vertices and are roughly orthogonal. Once formed, flowlines belonging to the same cluster should not cross one another. While the no-crossing constraint is a very strong clustering cue, we cannot apply it to our raw input consisting of individual edges. To take advantage of this cue, we employ a two-stage process: we first form *initial* strands using only cues applicable to edges and extract *initial* conservative flowlines from those; we then use those *initial* flowlines to generate a set of *reliable* strands using the full set of clustering cues; and finally use those strands to extract *reliable* extended flowlines.

The combination of positive and negative cues that we use for strand formation naturally feeds into a correlation clustering framework [Bansal et al. 2004]. We employ the version of correlation clustering that maximizes $\sum_e c_e Y_e$, where $Y_e \in 0, 1$ is 0 if the end nodes of the arc e are in different clusters and 1 if they are in the same cluster. While the general correlation clustering problem is NP-hard, it has a number of efficient approximation methods. We use the method of [Keuper et al. 2015] to compute the clusters; while not optimal, it performs well in practice. The specific edge weights used in our two clustering steps are defined below.

6.1 Initial Strands and Flowlines.

Initial Strands. To form initial strands using correlation clustering (Figure 6b) we treat mesh edges as graph nodes, and associate non-zero weights with the arc connecting edge nodes i, j when these either share a common vertex, or form opposite sides in a mesh quad. When a pair of mesh edges i, j share a common vertex, we define the arc weight c_{ij} as

$$c_{ij} = \begin{cases} e^{-\left(\frac{\alpha_{ij}}{\sigma_1}\right)^2}, & \text{if } \alpha_{ij} \le 45 \\ w_n e^{-\left(\frac{90-\alpha_{ij}}{\sigma_2}\right)^2}, & \text{otherwise} \end{cases} \tag{6}$$

α_{ij} is the angle difference between the unsigned edge directions projected to the vertex tangent plane, $w_n = -5$, $\sigma_1 = 5$, and $\sigma_2 = 15$. This weight is positive when edge directions are closer to parallel than orthogonal, and negative otherwise. For arcs connecting opposite edges within each quad, we define the weight as

$$c_{ij} = w_p e^{-\left(\frac{\alpha_{ij}}{\sigma_3}\right)^2}. \tag{7}$$

We set $\sigma_3 = 5^\circ$ and use a very small parallel coefficient $w_p = 0.05$, as at this stage we desire clusters dominated by local flowline smoothness.

Initial Flowline Extraction. We generate flowlines from strands by segmenting the connected components of each strand into individual flowline edge sequences (Figure 6c). We avoid making any potentially ambiguous choices by using all irregular and trim curve vertices within such components as flowline termination points and define each resulting one-dimensional edge sequence as a flowline.

6.2 Reliable Strands and Flowlines.

The initial flowlines allow for more global reasoning about, and consequently extraction of, more reliable strands and flowlines (Figure 9).

Reliable Strands. We obtain reliable strands using our correlation clustering setup by treating the initial flowlines as graph nodes, and associating arcs with pairs of flowlines f, f' that share common end vertices or contain edges on opposite sides of mesh quads. At each shared end vertex, we first compute the tangent vectors for the emanating flowlines f, f' (using the average across a local neighborhood set to five average mesh edge lengths). We then compute $c_{f,f'}$ as a function of the angle between these tangents using the formula in Equation 6, but with a more tolerant $\sigma_1 = 7.5$. For flowlines that share two endpoints we sum up the values obtained at both ends.

For each pair of flowlines f and f' that contain opposite edges $i \in f, j \in f'$ on shared quad mesh faces, we compute the arc weight as a function of both the angles between such pairs of opposite edges, and the proportion of such opposite edges as a function of the length l of the shorter flowline (intuitively the bigger this proportion, the more likely the flowlines are to be in the same strand):

$$c_{(f,f')} = \frac{2}{l} \sum_{i,j} e^{-\left(\frac{\alpha_{ij}}{\sigma_3}\right)^2}. \tag{8}$$

As observed earlier, crossing flowlines should not belong to the same cluster. We therefore associate a large constant negative arc weight $c_{(f,f')} = -25$ with each pair of crossing flowlines (f, f').

The overall score function that our strand computation seeks to maximize is $\max \sum c_{(f,f')} Y_{(f,f')}$ where $Y_{(f,f')}$ is 1 if the two flowlines are in the same cluster and 0 if not.

Reliable Flowline Extraction. We use the obtained strands to extract extended reliable flowlines (Figure 6e). The connected components of our flowline strands can form a range of graph configurations, allowing for multiple individual flowline configurations. We form our reliable flowlines using a greedy process which prioritizes extraction of more projectable and longer flowlines within each component. We measure projectivity using the metric in Section 5.2. Flowline endiness costs E_1, E_2 (Section 5.2) dominate all other projectivity components and are lowest for closed loops. Thus for each

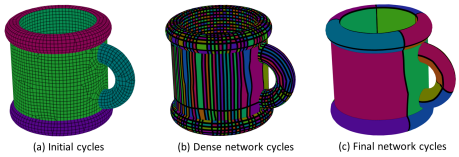


Fig. 10. Network computation: (a) Initial trim curve network; (b) descriptive dense network; (c) final network.

connected component we extract all closed loop flowlines first, prioritizing more projectable closed loops when given multiple options. We then use a greedy process to extract the longest open flowlines that cannot be extended, i.e. ones that start and end at a valence one vertex, again prioritizing more projectable ones, given multiple same length alternatives. Mesh artifacts can result in spiraling flowlines, where one edge is directly or indirectly parallel to another edge. While sometimes these spiral flowlines need to be included in the final network for description purposes, one cycle is often sufficient to describe the surrounding geometry. To facilitate processing of individual cycles, we detect spirals, and split them at the point where they complete a full cycle but have yet to become parallel. If there are multiple such points, we select the one(s) which result in the most projectable flowlines.

7 NETWORK COMPUTATION

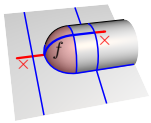
We use the computed flowlines to form a descriptive network by employing a mixed top-down/bottom-up strategy. Starting with a network of trim curves only, we first progressively add flowlines to obtain a dense network that describes the input model within the given descriptiveness threshold $d_{\max}$ (Figure 6f, 10b); we then simplify it by removing redundant flowlines (Figure 6g, 10c).

7.1 Top-Down: Dense Descriptive Network Computation

Network Initialization. We compute feature curves on the input model using ridge and valley detection [Yoshizawa et al. 2005] with conservative settings designed to capture only sharp features. The boundary and extracted feature curves form our initial network, which we use to partition the input surface into a set of cycles surrounded by network curves (Figure 10a). We evaluate the descriptive error d_c of each cycle and classify these cycles as either *covered* or *uncovered* depending on whether it is below, or above, the descriptiveness threshold $d_{\max}$. We note that the trim curve network may not form any cycles, and may even be empty; in this case, the initial cycle set contains one uncovered region which spans the entire input surface.

Network Refinement. We say that a flowline spans a region if it splits it into two or more separate regions. We refine the network by iteratively incorporating flowlines that span currently uncovered regions: for each uncovered region, we detect all flowlines that span it; we add all located spanning flowlines into the network, shortening them as described below to avoid forming undesirable network topology. We then compute the cycle-descriptiveness error (Section 5.1) for all newly formed regions. If uncovered regions remain, we perform another iteration; otherwise the algorithm terminates. In the rare case where uncovered regions have no spanning flowlines, we repeat the reliable flowline extraction step (Section 6.2) with the parameter σ_1 increased by 1°. We restrict this computation to flowlines that currently intersect the uncovered regions in question. We repeat this stage if necessary until coverage is achieved.

Flowline Shortening. As noted in Section 3, network projectivity depends on its connectivity. In particular, flowlines are least projectable when terminating at valence one or two end-points and T-junctions between flowlines and trim curves are more projectable than T-junctions between flowlines.



We maximize the projectivity of each open flowline we embed in the network, by collapsing open end-points to T-junctions (inset, right) and collapsing purely flowline T-junctions to flowline-trim curve T-junctions (inset, left), while constraining the shortened flowline to span the same set of uncovered regions. We first identify the section of each flowline that spans this set of regions and the excess sections on either end of it. If an excess section does not end at a trimming curve, but does intersect one, we shorten it to the closest such intersection to its current end point. Similarly, if it ends at a valence 1 or 2 vertex, we shorten it to end at the nearest flowline intersection to the current end point.

7.2 Bottom-Up: Network Simplification

We formulate the extraction of a compact network out of the dense descriptive network produced by the previous step as a direct optimization of the constrained problem formulated in Section 4. In contrast to the original formulation, which operated by assigning edges to flowlines, we keep the flowlines computed in the previous stage largely fixed, and focus on selecting the optimal subset among them (Figure 6g). Our discrete optimization goal can consequently be formulated as computing the subset of flowlines $f \in N$ that minimizes E_N, subject to the constraint that $\max d_c < d_{\max}$ over all cycles c in the resulting network. While such discrete constrained minimization problems often require sophisticated machinery to optimize, we found that a greedy approach that mimics classical mesh simplification, followed by local flowline movement, works sufficiently well for our needs.

Simplification. We place the flowlines in a priority queue ordered by their cost (Section 5.2), then repeatedly remove the highest cost flowlines whose removal does not violate our descriptiveness constraint from the network. After each flowline is removed, we update the cost of its neighboring flowlines and compute the descriptiveness error for the newly formed, merged, regions. This process terminates when no flowlines can be removed without violating our constraints. A typical simplification output is shown in Figure 10.

Removing a flowline from a network can result in undesirable valence 2 or 1 joints at the endpoints of remaining flowlines. To avoid these, we shorten such remaining flowlines by removing the sections between the undesirable endpoints and their nearest network intersections. Before shortening the flowlines we check if doing so would violate our descriptiveness threshold. If the threshold would be violated we abort the precipitating flowline removal; otherwise we remove the flowline and perform the shortening step.

Connectivity Optimization. When two or more flowlines end at a common valence four vertex, removing one of these flowlines reduces network projectivity by reducing the valence of the intersection. To penalize undesirable valence reduction, prior to starting the simplification, we locate all sequences of compatibly oriented flowlines that share common endpoints (flowlines are deemed compatible if the angle between their tangents at the shared point is obtuse). We merge these sequences into composite flowlines with their own score and place them into the simplification queue. For each flowline within the sequence we set the endiness E_i cost (Section 5.2) for the endpoint within the composite to zero, decreasing the likelihood of these flowlines being targeted for removal, before the composite. During simplification when any such interior flowline is removed, we update the composite and the cost of the other flowlines interior to this composite accordingly.

Post-process Local Optimization. Post-simplification we locally further optimize the network as follows (Figure 6h, 8a). First, for each flowline, we locate its immediately adjacent left and right flowlines in the same strand, and test the impact of removing the current flowline from the network and adding its immediate neighbor instead. We perform the substitution if it decreases the network energy and the resulting network still satisfies the descriptiveness threshold. We repeat these steps as long as the energy decreases.

After local optimization, we once again shorten open network flowlines, removing end-sections if doing so replaces valence 1 or 2 endpoints with T-junctions or replaces pure flowline T-junctions by flowline-trim curve ones, and does not violate our cycle-descriptiveness threshold.

As in many other segmentation setups, our output networks may occasionally end up with close by parallel flowlines none of which can be removed without lifting the descriptiveness error for a neighboring cycle just above our threshold. To avoid visual clutter, we detect such pairs of adjacent parallel flowlines (using a distance bound of two average mesh edge lengths) and examine the impact on the descriptiveness error of removing either one or the other. We select as a candidate a flowline whose removal increases the error the least. We remove this flowline if the increase in the descriptiveness error is less than 20%.

8 VALIDATION

We validate the results produced by our method in a number of ways: by comparing our outputs against artist-generated networks and prior art, by performing a qualitative evaluation study, and by demonstrating that the networks generated by our method can be used to closely reproduce the originating models.

Comparison to Artist Generated Networks. We selected 4 models (rounded cube, treball, mouse, boat) ranging from simple (cube) to highly complex (mouse, boat) and provided them to three industrial designers/artists. We asked one artist to manually generate descriptive 3D curve networks for these models, and asked two artists to create design drawings of them from given descriptive views. While the artist results are, as expected, not identical, they all share common characteristics, which are similarly captured by our outputs (Figure 11). It took the artist over three hours to generate the 3D curve networks (two hours for the mouse, one for the boat, and about half an hour total for the simpler two models). Our fully automatic method takes a fraction of this time, generating all four results in under five minutes.

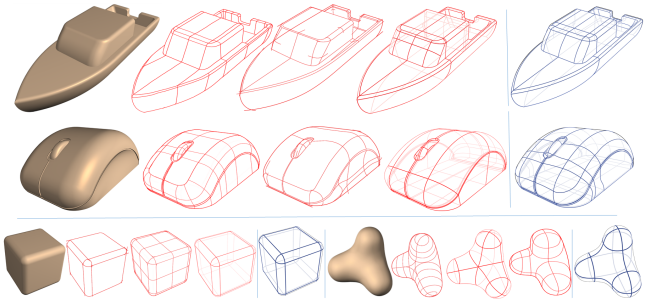


Fig. 11. Comparison against artist generated networks: (left to right) input model, artist generated 2D design drawings and 3D network (red), and our algorithmic result (blue).

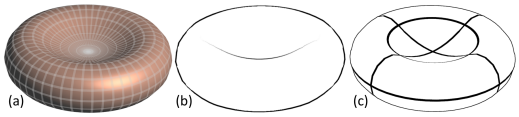


Fig. 12. Suggestive contours combined with ridge and valley lines [DeCarlo et al. 2003] (b) convey the overall input shape (a); a FlowRep descriptive network (c) provides a more detailed and accurate description of the input geometry.

To provide further comparison to artist-created networks, we qualitatively compared our output to the inputs used by [Bessmeltsev et al. 2012; Pan et al. 2015] (Figure 16, right). We directly processed quad-meshes produced by Bessmeltsev et al. [2012] (coffee-machine, airplane). We also quad-meshed and processed the dog-head surface produced by Pan et al. [2015]. Our output networks (blue) are very similar to their inputs (green).

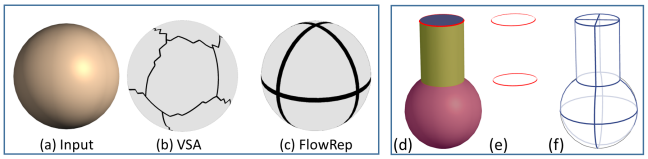


Fig. 13. Surface segmentation (e.g. VSA [Cohen-Steiner et al. 2004]) (b) and reverse engineering methods (d,e) are not designed for, and do not produce, projectable curve networks; their output is often not descriptive to a human observer. FlowRep networks (c,f) satisfy both criteria.

Comparison to Prior Art. Figures 12, 13, 14, and 15 show representative comparisons of our outputs against prior art. As Figure 12 demonstrates, while suggestive contours combined with ridge/valley lines convey the overall input shape, our networks provides a more precise description of the input. Figure 13 contrasts our networks with those generated by planar segmentation and reverse engineering approaches; as demonstrated FlowRep outputs support better mental visualization of the input than such approaches.

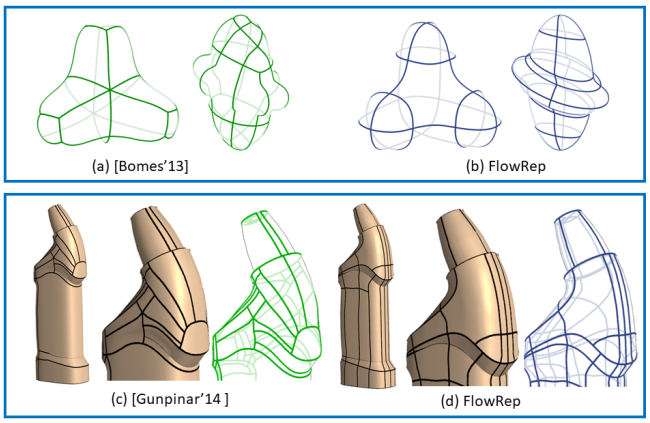


Fig. 14. (a) Quad-partition [Bommes et al. 2013] of the treball and ellipsoid (a) compared to our network (b). Quad partitions, here [Gunpinar et al. 2014a], are highly dependent on the singularity locations in the initial mesh drifting from curvature directions (c). FlowRep result on same model (d).

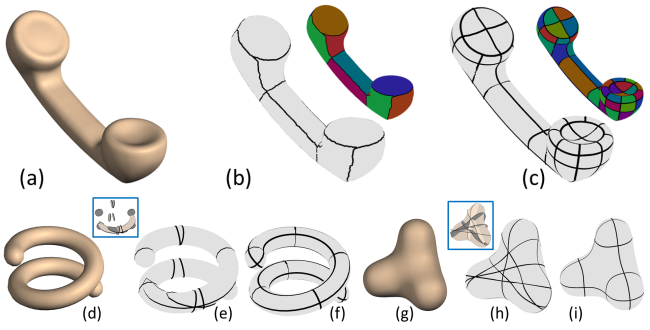


Fig. 15. While exoskeletons [De Goes et al. 2011] only roughly capture coarse part structures of shapes (b), our method describes the geometry in more detail (c). Planar slices [McCrae et al. 2011] are restricted in their ability to convey free-form shape (e,h), while FlowRep networks are well suited for this task (f,i).

As demonstrated in Figures 3 and 14a, quad partition boundaries (e.g. [Bommes et al. 2013]) are not projective, making it hard for viewers to envision the originating geometry. They are also often misaligned with curvature directions as highlighted by the mismatch between shading and curve directions in Figure 14c (network courtesy of [Gunpinar et al. 2014a]). Such misalignment results in non-descriptive cycles. Our networks are strictly aligned with curvature directions (Figure 14d) and enable viewers to envision the originating shape from a 2D projection.

While exoskeletons [De Goes et al. 2011] reduce the number of T-junctions formed by unconstrained VSA [Cohen-Steiner et al. 2004], this semi-automatic method is only suitable for coarse abstraction, while our automatic framework captures much finer details on the same geometry (Figure 15, top). As highlighted by Figure 18, mimicking their bottom-up approach using an automatic process results in fragmented flowlines. Lastly, as shown in Figure 15 (bottom) planar slice proxies [McCrae et al. 2011], while descriptive of multipart simple shapes, compare poorly to our networks on typical design shapes.

Qualitative Evaluation. We showed seven designers (including the two who produced the drawn networks) images of our results and of the results produced by the artists, without telling them which is which. To ensure uniform style we used identical views, line style and color, and generated non-transparent renders for the 3D models. The model was shown in top row and the renders in bottom row. We then asked the designers "How well do the design drawings on the bottom reflect the shape on the top?" (see supplementary material for exact questionnaire). All seven designers assessed all the shown networks as reflective of the input, ranking ours on par with other renders; three of them commented on our mouse and boat as being most descriptive. The positive comments included "easier to understand", "precise and simple", "great sense of depth". On the negative side one commented that some of our curves were not smooth, and one designer felt that the curves on the mouse were too close to one another.

We also conducted a study to compare our outputs to previous work. We asked 34 nonexpert users to compare our outputs to curve networks generated by alternative methods. Each query in this study included an input model (A, top) and two curve networks (B and C, bottom), arranged in random order and presented side by side: one generated by our algorithm, and one generated by an alternative method (suggestive contours [DeCarlo et al. 2003], quad patch layouts generated by [Bommes et al. 2013] and [Gunpinar et al. 2014a], variational shape approximation [Cohen-Steiner et al. 2004], exoskeletons [De Goes et al. 2011], and planar slices [McCrae et al. 2011], totaling eight queries.) Users were asked the question, "Which figure on the bottom (B, or C) more *accurately* describes the shape on the top (A)?". Across all queries participants preferred our outputs to the alternatives 92% of the time. The individual comparison with lowest majority preference for our method (80%) was against quad partition [Bommes et al. 2013] on the ellipsoid (Figure 14). The network drawings used for the evaluation were generated using descriptive views, and the same view was used for all models. The exact questionnaires used in the evaluation are included in our supplementary material.

Reproduction. Our method aims to create networks that can be used to reproduce the input shapes both mentally and algorithmically. To validate the algorithmic reconstruction feature we ran the surfacing framework of [Pan et al. 2015] on a subset of our outputs (Figure 16, left). As shown, the surfaced networks look very similar to the input models. The L_2 distance between the inputs and the re-surfaced networks measured using [Cignoni et al. 1998] was under 0.3% of the bounding box diagonal, leading us to conclude that our networks do not only describe the models perceptually but also compactly and accurately encode their geometry.

9 RESULTS

We have demonstrated our method on a large set of inputs, ranging from simple (rounded cube, spiral, torus, bump) to highly complex (shuttle, boats, mouse, toilet). Our tests included open surfaces (beetle, bathtub), ones with sharp features (mouse, row-boat), and

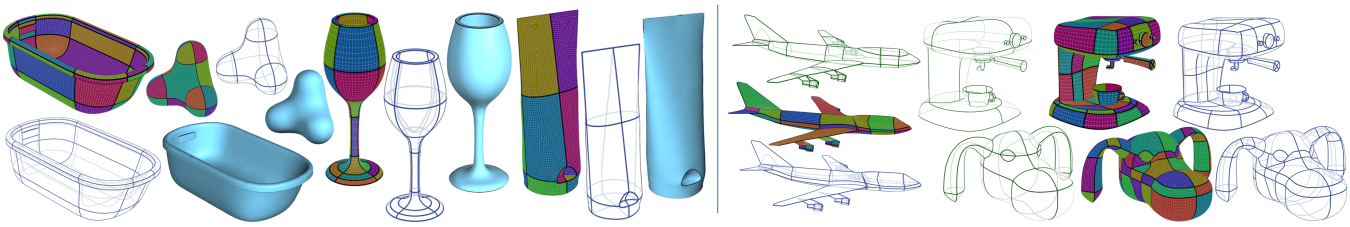


Fig. 16. Reproduction. Left: pairs of input models with computed FlowRep networks (wireframes) and these networks resurfaced using [Pan et al. 2015] (blue). Right: Input curve networks (green), surfaces produced by [Bessmeltsev et al. 2012; Pan et al. 2015], and our networks (blue) computed from these surfaces.

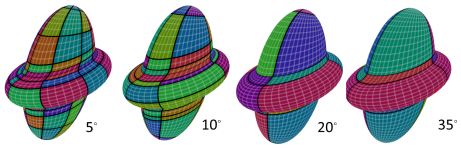


Fig. 17. Networks with different descriptiveness thresholds , shown on the ellipsoid model.

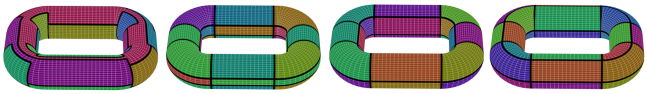


Fig. 18. Alternative network extraction methods: (a) bottom-up cycle clustering results in poorly descriptive networks with highly irregular connectivity; network optimization using only flowline projectivity (b) or only their dominance (c), versus full optimization (d).

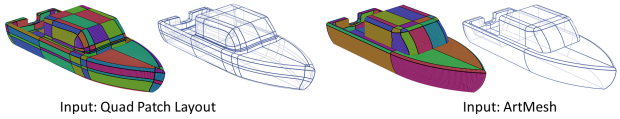


Fig. 19. Curve networks generated from different meshes of same model: (a) [Bommes et al. 2013], (b) ArtMesh. The input mesh in (a) exhibits feature drift (sharp features migrating between mesh flowlines); leading FlowRep to preserve these flowlines generating visual redundancy.

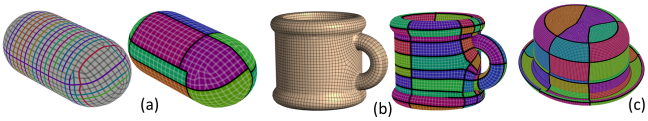


Fig. 20. Impact of systemic mesh vs curvature tensor misalignment: (a) Interleaved spiral flowlines resulting from curvature drift may result in undesirable T-junctions (mesh from [Jakob et al. 2015]); (b) more significant misalignment predictably distorts the network orientation (mesh from [Bommes et al. 2013]); (c) large patches of randomly aligned quads (top of the hat) similarly lead to artifacts.

smoother, more organic ones (treball, spiral, ellipsoid, phone handle, dog-head, big buck bunny). We tested our method on coarsely meshed complex shapes (beetle, at 4K triangles) as well as fine meshed ones (mouse, boat, toilet, at 30K triangles each). Our results on all these models are consistent with human expectations.

Symmetry. Most quad-meshers do not detect or enforce global symmetry. To produce globally reflectively symmetric results we use external code to detect global reflective symmetries, quad-mesh one half of a symmetric model, and use a reflected mesh as input. We used this approach for the boat, mouse, bottle, phone-handle, big buck bunny, and dog-head.

Network Resolution. Our framework allows users to control the density of the output curve networks by changing the descriptiveness threshold $d_{\max}$, as illustrated in Figure 17. By default we use a threshold of 20°. For the rowboat and wineglass we used thresholds of 30° and 10° to obtain more aesthetically pleasing results. We use 30° for more organic models (doghead, phone).

Impact of Design Choices. Figures 7 and 18 demonstrate the impact of our key algorithmic choices on the final results. Figure 7 demonstrates the importance of using our strand formation process to obtain reliable flowlines. This process contributes to our success in processing highly irregular meshes, with singularities and numerous edges which are misaligned with curvature directions. Figure 18 shows a comparison against a number of alternative network formation strategies. The first mimics [De Goes et al. 2011] as it uses a bottom up cycle growth strategy directly optimizing E_N (Equation 1) while aiming to keep consecutive flow-lines or edges together to minimize the appearance of T-junctions. Even on simple models, this strategy results in highly irregular networks with redundant T-junctions and fragmented flowlines.

Figures 18b and c show the impact of using a network simplification strategy that takes only flowline projectivity (b) or only its dominance (c) into account. Both networks satisfy our descriptiveness threshold, but select different representative flowlines within individual strands. Absent dominance (Figure 18b) the result has more geodesic flowlines but has some less well described cycles; in contrast in Figure 18c the cycles are well described but the interior symmetry curve is not included. Our solution (Figure 18d) balances both sets of criteria, and is more reflective of traditional drawings of such toroidal shapes.

Input Mesh Impact. We tested our framework on quad-meshes produced by a range of sources. Most of our inputs were generated using ArtMesh (http://www.topologica.org) or were *a priori* given to us in quad-mesh format (spiral, glass, bathtub, beetle). The plane and coffee-machine in Figure 16 were generated by [Bessmeltsev et al. 2012]. We also tested inputs generated using quad-patch layout [Bommes et al. 2013] (Figures 19, 20b) and Instamesh [Jakob et al. 2015] (phone handle, and Figure 20a). As our examples demonstrate, FlowRep is largely agnostic to mesh artifacts, such as non-quad elements, irregular connectivity, and uneven element sizing, e.g. Figure 3b. Our method is more sensitive to consistent

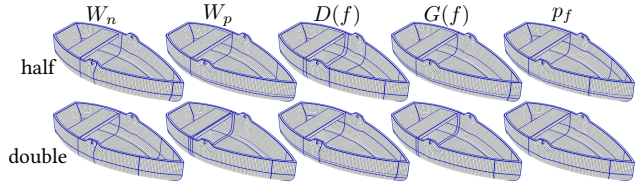


Fig. 21. Impact of halving/doubling default algorithm parameters (weights used for correlation clustering; weights of different elements of E_n (Eq. 1)).

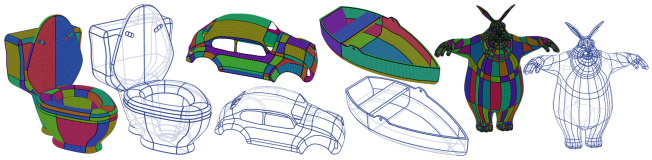


Fig. 22. Additional results.

curvature misalignment and irregular mesh flow in isotropic areas (hat, Figure 20c). Input meshes whose edges form long spirals due to systemic edge direction drift [Jakob et al. 2015] may lead to extra T-junctions in the final output (Figure 20a). While FlowRep can process meshes whose edge directions consistently and significantly deviate from curvature directions, such as those sometimes produced by quad-patch layout (e.g. [Bommes et al. 2013]), the results in this case are, as expected, less reflective of human expectations (Figure 20b).

Runtimes and Parameters. Our runtimes range from 3 seconds for simple models such as the wineglass (5K faces) to 48 and 111 seconds for complex ones such as the mouse (30K faces) and the toilet (28K faces) respectively. Roughly 75% of the time is spent performing edge correlation clustering for flowline initialization. Once initial flowlines are formed, the simplification stage dominates, taking another 20%. Overall, the dominating runtime factors are the original mesh resolution and geometric complexity. With the exception of the descriptiveness threshold $d_{\max}$ all other algorithm parameters are fixed across all inputs, and as shown in Figures 18, 21 changing them has fairly minimal impact on the outcome.

Limitations. Our method relies on curvature-aligned quad-dominant meshes to serve as a reliable proxy of a curvature aligned tensor field smoothly extended across isotropic areas. It successfully overcomes sporadic topological noise (e.g. Figure 3) but is affected by systemic misalignment and uneven mesh sizing (Figure 20). The trade-off we require is supported by many research and commercial meshers. Our framework does not explicitly account for symmetries, beyond global reflection, even when these are present in the mesh. Detecting such symmetries at the mesh level using existing methods, and then enforcing similar processing on symmetric edges and flowlines, would alleviate this concern.

10 CONCLUSIONS

This paper presented FlowRep, the first algorithm for computing descriptive compact curve networks from an input mesh. Our output networks succinctly describe complex free-form shapes and are suitable for both perceptual and algorithmic reconstruction. When computing the networks our method optimizes two main criteria: cycle-descriptiveness, or the network's ability to describe the underlying surface, and network projectivity — the ability to perceive the network's 3D shape from its 2D projection. We use these properties to first trace reliable flowline curves on the surface, then use the flowlines to extract an initial projectable and descriptive network, and finally coarsen the network while maintaining a given descriptiveness threshold. Combined together, these steps result in descriptive networks comparable with those produced by artists, and which have been extensively validated to show that they agree with viewer perception.

This work opens many avenues for future research. As noted, our results are contingent on the quality of the input quad meshes. Producing initial meshes tuned to optimally adhere to our requirements can both simplify our algorithm and improve its results. Ours is the first attempt to evaluate network suitability for the task at hand. Additional perception research may be able to improve this formulation, and machine learning algorithms based on perceptual studies might better pinpoint the necessary balance between the key geometric properties we have identified.

ACKNOWLEDGEMENTS

This work has been funded by NSERC and by a gift from Adobe Inc. We are grateful to Marcel Campen, Fernando De Goes, Erkan Gunpinar, Mikhail Bessmeltsev and James McCrae for providing comparison data, and Hao Pan for surfacing our outputs. We would like to thank Yixin Zhuang for his help with the project. The "big buck bunny" model is provided courtesy of the Blender Institute, licensed CC-Attribution. Mouse, bowler cap and wine glass are provided courtesy of Microsoft, licenced CC BY 4.0. "miniature row boat" and "bathtub" are provided by thingiverse.com, licensed CC BY-SA 3.0.

REFERENCES

Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. 2003. Anisotropic Polygonal Remeshing. *ACM Trans. Graph.* 22, 3 (2003), 485–493.
Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation Clustering. *Mach. Learn.* 56, 1-3 (2004), 89–113.
Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. 2012. Design-driven Quadrangulation of Closed 3D Curves. *ACM Trans. Graph.* 31, 6, Article 178 (2012), 178:1–178:11 pages.
Roseline Bnire, Grard Subsol, Gilles Gesquire, Franois Le Breton, and William Puech. 2013. A comprehensive process of reverse engineering from 3D meshes to CAD models. *Computer-Aided Design* 45, 11 (2013), 1382 – 1393.
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), 98:1–98:12 pages.
David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30, 2 (2011), 375–384.
David Bommes, Tobias Vossemer, and Leif Kobbelt. 2008. Quadrangular Parameterization for Reverse Engineering. *(Lecture Notes in Computer Science)*, Vol. 5862. Springer, 55–69.
Monica Bordegoni and Caterina Rizzi. 2011. *Innovation in product design.* Springer.
Marcel Campen, David Bommes, and Leif Kobbelt. 2012. Dual Loops Meshing: Quality Quad Layouts on Manifolds. *ACM Trans. Graph.* 31, 4, Article 110 (2012), 110:1–110:11 pages.
Marcel Campen, Moritz Ibing, Hans-Christian Ebke, Denis Zorin, and Leif Kobbelt. 2016. Scale-Invariant Directional Alignment of Surface Parametrizations. *Computer Graphics Forum* (2016).
Marcel Campen and Leif Kobbelt. 2014. Dual Strip Weaving: Interactive Design of Quad Layouts Using Elastica Strips. *ACM Trans. Graph.* 33, 6 (2014).
Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. 2014. Field-aligned Mesh Joinery. *ACM Trans. Graph.* 33, 1, Article 11 (2014), 11:1–11:12 pages.
P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* (1998).

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational Shape Approximation. *ACM Trans. Graph.* 23, 3 (2004), 905–914.
Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. 2008. Where Do People Draw Lines? *ACM Trans. Graph.* 27, 3 (2008).
Fernando De Goes, Siome Goldenstein, Mathieu Desbrun, and Luiz Velho. 2011. Exoskeleton: Curve Network Abstraction for 3D Shapes. *Comput. Graph.* 35, 1 (2011), 112–121.
Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. 2003. Suggestive Contours for Conveying Shape. *ACM Trans. Graph.* 22, 3 (2003), 848–855.
Koos Eissen and Roselien Steur. 2008. *Sketching: Drawing Techniques for Product Designers.* Bis Publishers.
Koos Eissen and Roselien Steur. 2011. *Sketching: The Basics.* Bis Publishers.
David Eppstein, Michael T. Goodrich, Ethan Kim, and Rasmus Tamstorf. 2008. Motorcycle Graphs: Canonical Quad Mesh Partitioning. In *Proceedings of the Symposium on Geometry Processing*. 1477–1486.
Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. iWIRES: An Analyze-and-Edit Approach to Shape Manipulation. *ACM Trans. Graph.* 28, 3 (2009), #33, 1–10.
Anne Gehre, Isaak Lim, and Leif Kobbelt. 2016. Adapting Feature Curve Networks to a Prescribed Scale. *Computer Graphics Forum* (2016).
Erkan Gunpinar, Masaki Moriguchi, Hiromasa Suzuki, and Yutaka Ohtake. 2014a. Feature-aware Partitions from the Motorcycle Graph. *Comput. Aided Des.* 47 (2014), 85–95.
Erkan Gunpinar, Masaki Moriguchi, Hiromasa Suzuki, and Yutaka Ohtake. 2014b. Motorcycle Graph Enumeration from Quadrilateral Meshes for Reverse Engineering. *Comput. Aided Des.* 55 (2014), 64–80.
Klaus Hildebrandt, Konrad Polthier, and Max Wardetzky. 2005. Smooth Feature Lines on Surface Meshes. In *Proc. SGP 2005 (SGP '05)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, Article 85.
Emmanuel Iarussi, David Bommes, and Adrien Bousseau. 2015. BendFields: Regularized Curvature Fields from Rough Concept Sketches. *ACM Trans. Graph.* (2015).
Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Trans. Graph* 34, 6 (2015).
Dan Julius, Vladislav Kraevoy, and Alla Sheffer. 2005. D-Charts: Quasi-Developable Mesh Segmentation. *Computer Graphics Forum* (2005).
Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoué, Thomas Brox, and Bjoern Andres. 2015. Efficient Decomposition of Image and Mesh Graphs by Lifted Multicuts. In *Proc. ICCV*.
Yu-Kun Lai, Qian-Yi Zhou, Shi-Min Hu, Johannes Wallner, and Helmut Pottmann. 2007. Robust Feature Classification and Editing. *IEEE Trans. Vis. Comp. Graphics* 13, 1 (2007), 34–45.
James McCrae, Karan Singh, and Niloy J. Mitra. 2011. Slices: A Shape-proxy Based on Planar Sections. *ACM Trans. Graph.* 30, 6 (2011).
Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. 2009. Abstraction of Man-Made Shapes. *ACM Trans. Graph.* 28, 5 (2009).
Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-aligned Global Parametrization. *ACM Trans. Graph.* 33, 4, Article 135 (2014), 135:1–135:14 pages.
Matthias Nieser, Christian Schulz, and Konrad Polthier. 2010. Patch Layout from Feature Graphs. *Comput. Aided Design* 42, 3 (2010).
Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Changjian Li, and Wenping Wang. 2015. Flow Aligned Surfacing of Curve Networks. *ACM Trans. Graph.* 34, 4 (2015).
Nicolas Ray and Dmitry Sokolov. 2014. Robust Polylines Tracing for N-Symmetry Direction Field on Triangulated Surfaces. *ACM Trans. Graph.* 33, 3, Article 30 (2014), 30:1–30:11 pages.
Faniry H. Razaflndrazaka, Ulrich Reitebuch, and Konrad Polthier. 2015. Perfect Matching Quad Layouts for Manifold Meshes. In *Proceedings of the Eurographics Symposium on Geometry Processing*. 219–228.
Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: shading concept sketches using cross-section curves. *ACM Trans. Graph.* 31, 4 (2012).
Kent A. Stevens. 1981. The visual interpretation of surface contours. *Artificial Intelligence* 17 (1981). Issue 1-3.
Vitaly Surazhsky and Craig Gotsman. 2004. High quality compatible triangulations. *Eng. Comput. (Lond.)* 20, 2 (2004), 147–156.
Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. 2011. Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Trans. Graph.* 30, 6, Article 142 (2011), 12 pages.
J Todd and F. Reeichel. 1990. Visual perception of smoothly curved surfaces from double-projected contour patterns. *J. of Exp. Psych.: Human Percep. and Performance* 16 (1990), 665–674.
Jianhua Wu and Leif Kobbelt. 2005. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum* 24, 3 (2005), 277–284.
Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James Mccrae, and Karan Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Trans. Graph.* 33, 4 (2014).
Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. 2005. Fast and Robust Detection of Crest Lines on Meshes. In *Proc. Symp. on Solid and Physical Modeling (SPM '05)*. 227–232.
Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. 2014. Anisotropic Geodesics for Live-wire Mesh Segmentation. *Computer Graphics Forum* (2014).

APPENDIX

Algorithm 1 Compute Flowlines

procedure ComputeFlowlines(mesh = (V, E))
  Construct a graph G_1 where each node is an edge of mesh
  for each edge $i \in E$ **do**
    for each edge j connected to i **do**
      $G_1[i, j] \leftarrow c_{ij}$ (Eq. 6)
    for each edge j face-opposite to i **do**
      $G_1[i, j] \leftarrow c_{ij}$ (Eq. 7)
  C_1 = CorrelationClustering(G_1) ▷ maximize $\sum_e c_e Y_e$
  Join connected edges in each $c \in C_1$ into *initial flowlines*
  Construct a graph G_2 where each node is a flowline
  for each flowline f **do**
    for each flowline f' sharing an end-point v with f **do**
      $G_2[f, f'] \leftarrow c_{ij}$ (Eq. 6, with i and j being tangential vectors of f and f' at v.)
    for each flowline f' intersecting f **do**
      $G_2[f, f'] \leftarrow (-25)$
    for each flowline f' parallel to f **do**
      $G_2[f, f'] \leftarrow c_{f,f'}$ (Eq. 8)
  C_1 = CorrelationClustering(G_2) ▷ maximize $\sum_e c_e Y_e$
  Join connected flowlines in each $c \in C_2$ into *reliable flowlines*

Algorithm 2 Network Computation

procedure ComputeNetwork(network N)
  for each trimming curve t **do** ▷ Network Refinement
    $N \leftarrow (N \cup \{t\})$
  while $\exists$(cycle c) | $d_c < d_{\max}$ **do**
    for each cycle c with $d_c > d_{\max}$ **do**
      for each flowline f splitting c in two parts **do**
        $N \leftarrow (N \cup \{f\})$
  Construct all composite flowlines f_c ▷ Network Simplification
  Let C be the set of all composite flowlines
  Let Q be a priority queue
  for each flowline $f \in (N \cup C)$ **do**
    Compute flowline cost $p_f + d_f$ of f (Eq. 4, 5)
    $Q \leftarrow Q \cup \{f\}$
  while Q is not empty **do**
    $f \leftarrow$ maximal cost flowline in Q
    $Q \leftarrow (Q \setminus \{f\})$
    if $\forall$cycles $c \in (N \setminus \{f\})$, $d_c < d_{\max}$ **then**
      $N \leftarrow (N \setminus \{f\})$
      Update costs $p_{f_n} + d_{f_n}$ of neighbors f_n of f
  for each flowline $f \in N$ **do** ▷ Local Optimization
    for each neighbor f_n of f **do**
      if $p_{f_n} < p_f \wedge \forall$cycle $c \in ((N \setminus \{f\}) \cup \{f_n\})$, $d_c < d_{\max}$ **then**
        $N \leftarrow ((N \setminus \{f\}) \cup \{f_n\})$