FlowRep: Extracting Descriptive Curve Networks from Free-Form Design Shapes

by

Giorgio Gori

B. in Informatics, Università della Svizzera Italiana, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2017

© Giorgio Gori, 2017

Abstract

This thesis presents *FlowRep*, an algorithm for extracting descriptive compact 3D curve networks from meshes of free-form man-made shapes. FlowRep output networks provide a concise visual description of the underlying surface, and can be used as a compact proxy for shape compression, editing and manipulation. While artists routinely and successfully create descriptive curve networks to depict complex 3D shapes in 3D space or on 2D media, the method described here is the first to achieve this goal algorithmically. FlowRep infers the desired compact curve network from complex 3D geometries by using a series of insights derived from perception, computer graphics, and design literature which point to two sets of geometric properties that such networks should satisfy. 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. The algorithm leverages these observations in the construction of a compact descriptive curve network. Starting with a curvature aligned quad dominant mesh I first extract sequences of mesh edges that form long, well-shaped and reliable flowlines by leveraging directional similarity between nearby meaningful flowline directions. This process overcomes topological noise, and inaccuracies and singularities in the underlying curvature field. I then use the extracted flow-

lines and the model's sharp-feature, or trim, curves to form a projectable network which describes the underlying surface. Finally, I simplify this network while preserving its descriptive power to obtain the final result. My co-authors and I 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.

Preface

The ideas and algorithms described in this thesis, unless stated below, were developed by myself in consultation with Dr. Alla Sheffer, Dr. Nathan Carr and Dr. Tao Ju.

The content of Section 4.5 (Regularization) was developed by Nicholas Vining in consultation with Dr. Alla Sheffer and included here for completeness. The user studies mentioned in Section 8.3 (Figures A.1, A.2) were conducted by Enrique Rosales and Nicholas Vining with UBC approval (UBC BREB Number H16-02320).

All the ideas and algorithms described were submitted in the research paper:

- 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. [23]

Table of Contents

List of Figures

Acknowledgments

Many thanks to Nathan Carr from Adobe and Tao Ju from Washington University in St. Louis for helpful discussion, insight and feedback.

I would like to thank my supervisor Alla Sheffer for guidance, support and for being my academic sounding board in the past two years.

Thanks to Nicholas Vining, Enrique Rosales and Chenxi Liu for the help when was most needed and for complementing my skills.

Many thanks to University of British Columbia and the Department of Computer Science for offering me this fantastic opportunity and for the financial aid they provided.

Thanks especially to my parents, who encouraged and supported my choice of studying far from home.

Chapter 1

Introduction

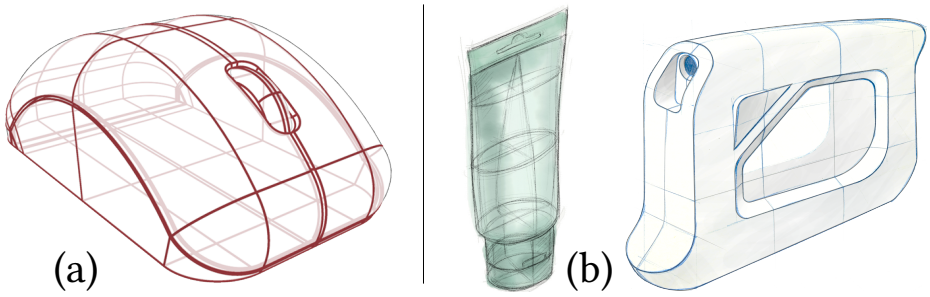

Figure 1.1: Artist generated 3D (a) and 2D (b) descriptive curve networks succinctly convey complex free-form shapes.

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 1.1). 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 [21]; facilitate compact shape representation and abstraction [33]; support shape-preserving mesh simplification [22]; and enable other high-level operations. This thesis proposes *FlowRep*, a method for computing visually descriptive compact curve networks of free-form

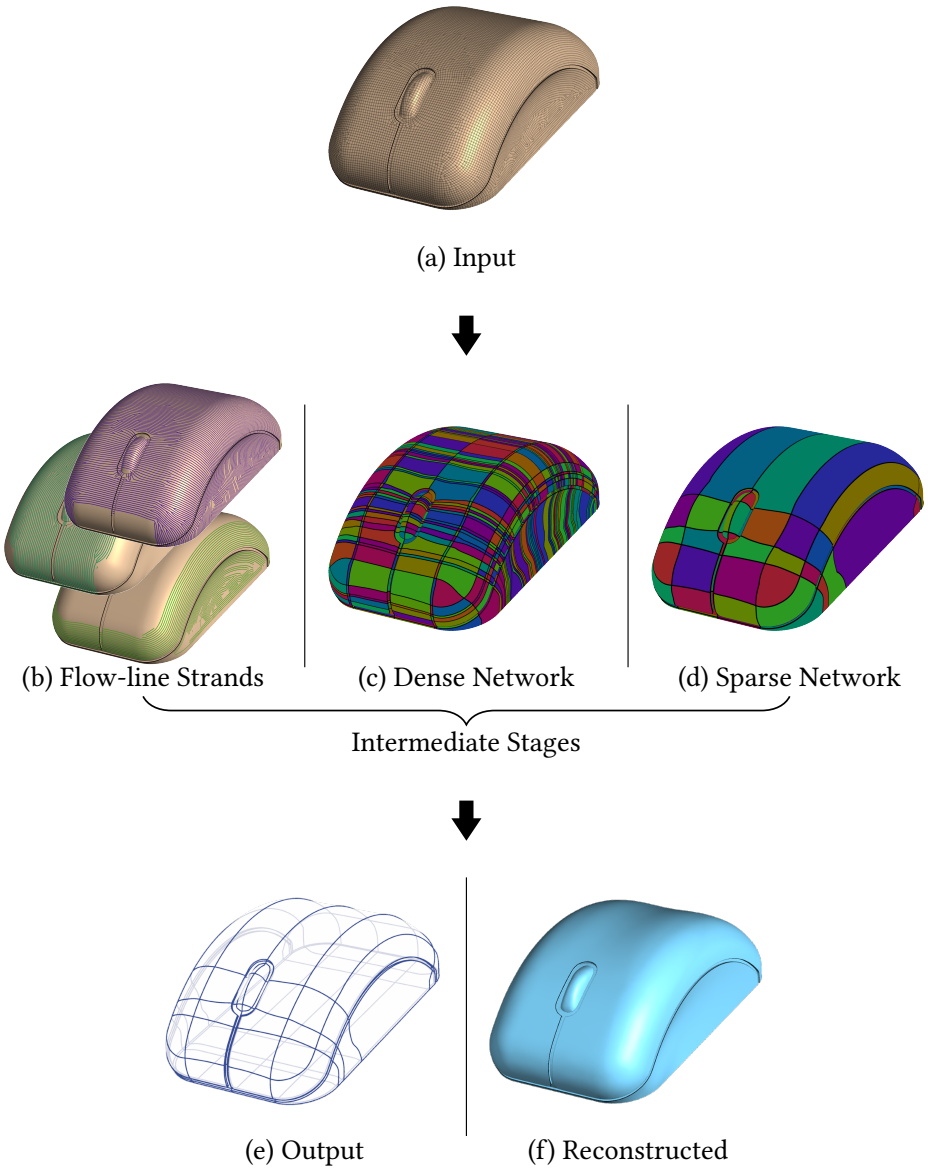

Figure 1.2: *FlowRep* describes complex free-form 3D geometries (a) by a compact network of descriptive and projectable curves (e) that can be used to both depict and reconstruct (f) the input (L_2 distance between (a) and (f) is 0.1% of bounding box diagonal). Given an input quad mesh (a) it extracts strands of dominant flowlines (b), uses those to computes a dense descriptive network (c) and then systematically simplifies it to obtain the desired compact net (d).

man-made, or designed, shapes from existing models (Figure 1.2). FlowRep networks effectively convey the shape of an input 3D object geometry to human observers and can be used for the range of applications described above. They accurately encode input geometry enabling both perceptual and geometric reconstruction; the mouse in Figure 1.2f was accurately reconstructed from our curve network alone using the method of [36].

While previous methods have addressed the related problems of surface partitioning [4, 10, 14, 20, 34] and extraction of sets or networks of different feature curves [17, 22, 33], the partition boundaries or sets of curves they produce do not provide a detailed description of complex man-made free-form shapes (Chapter 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 the comparisons presented later in this thesis, FlowRep 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 [3, 36].

The first challenge that I face in extracting the desired curve networks from input models is determining the properties these networks must possess to adequately describe a given shape. The design literature points to two sets of criteria that these curve configurations should satisfy (Chapter 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 [36, 45], 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 (Chapter 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 [45]. 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.

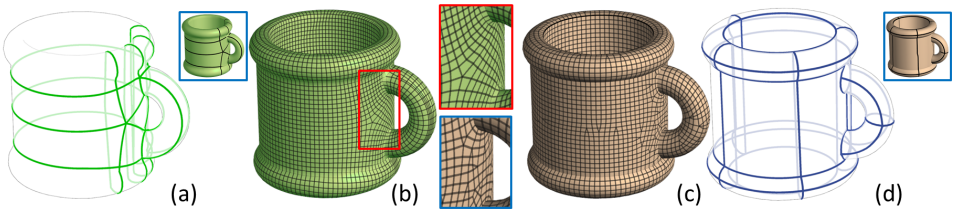

Figure 1.3: Quad-meshing methods that optimize for mesh regularity, such as [7], use quad partitions (a) as a starting point and often exhibit systemic misalignment with curvature directions as highlighted in (b). Meshing methods that seek to adhere to curvature directions more strictly often result 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).

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 [3, 36]. 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, the design literature indicates a strong preference for using *compact*, minimalist, cycle-descriptive networks to avoid visual clutter [18]. 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, I seek to compute a compact descriptive set of projective dominant flow lines on the input shapes. To extract this compact network, I leverage observations about the desired properties of such curves derived from design and modeling literature (Chapter 3) and use those to quantify dominance, descriptiveness and projectivity. I then employ these definitions in a network computation algorithm.

Tracing individual flowlines, especially across isotropic regions and near curvature-field singularities, is inherently unreliable. This thesis provides 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. [7, 11], often requires significant deviation from curvature directions (Figure 1.3b). In my 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 1.3c). I and my co-authors 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. My 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.

The contribution of this thesis is two-fold. I identify and enumerate the key properties of descriptive curve networks suitable for communicating complex designer shapes; I then propose the first curve network extraction algorithm that generates networks with these desired properties. I and my co-authors test the 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 (Chapters 8, 9). As this validation confirms, the output networks successfully capture and convey the essence of the input shapes both in 3D space, and when viewed from general viewpoints.

Chapter 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. I briefly review these works, analyzing how well their outputs adhere to our three goals: descriptiveness, projectivity, and compactness.

2.1 Curvature Aligned Meshes.

At the finest level, curvature aligned polygonal meshes provide two of the qualities we seek: descriptiveness and projectivity (Figure 2.1b). For example, Alliez et al. [1] extract a curvature aligned quad-dominant mesh from the curvature

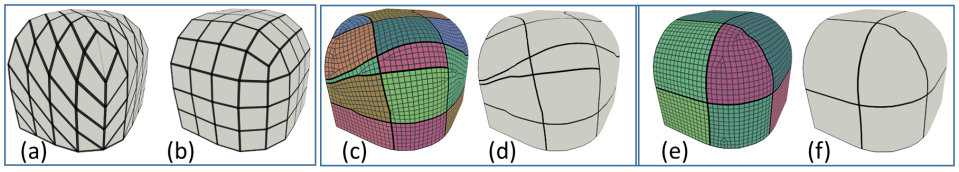

Figure 2.1: Projectivity and cycle descriptiveness (all renders show the same 3D model in same view). Projection of an orthogonal quad mesh (b) conveys the underlying 3D geometry better than that of a non-orthogonal mesh (a). A flowline network (f) over a curvature aligned field (e) succinctly describes the surface, while a network (d) generated from an arbitrary smooth cross-field (c) does not.

tensor field. However, such meshes are clearly not compact. We use them as a starting point for our framework, which compacts them while maintaining these two key properties (Figure 2.1f).

2.2 Mesh Segmentation and Reverse Engineering.

Methods for mesh segmentation, e.g. [14, 29] and reverse engineering, e.g. [4, 35, 44] aim to segment models into regions with particular surface characteristics, for instance developable surfaces, planes, or conics. Cohen-Steiner et al. [14] approximate the mesh with geometric proxies and Wu et al. [44] expand that method by using different primitives. Julius et al. [29] segment the mesh in quasi-developable charts. Nieser et al. [35] use feature lines and surface curvature to reverse engineer the surface in regions. Beniere et al. [4] detect various geometric primitives and their intersections to reverse engineer CAD models. All those methods 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 they 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 8.3). The key distinguishing feature of our method compared to these approaches is our focus on network properties rather than region properties, and consequently the ability to concisely and effectively describe input surfaces independent of their specific geometry using network curves alone.

2.3 Quad Patch Layout.

A range of methods extract coarse quad patch layouts to facilitate parameterization and surface fitting. Early methods, such as [5] generate such layouts semi-manually based on a coarse layout provided by the user. This approach is still popular; for example Campen et al. [9] let the user specify "elastica strip" (closed geodesic loops aligned with curvature) while Zhuang et al. [47] have the user choose segmentation boundaries by picking geodesics in a carefully crafted met-

ric. More recent frameworks use motorcycle graphs, starting at quad mesh singular points [20] to obtain singularity-free quad patches. Recent works [6, 42] improve the patch layouts by simplifying spiraling patch boundaries and merging nearby singularities. Gunpinar et al. [24, 25] 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 [7, 10, 34, 38] and trace separatrices from field singularities to form quad patches. Bommes et al. [7] use a mixed-integer approach to produce reliable quad-meshes. Campen et al. [10] construct patch layouts guided by the curvature field, then extract the dual graph and finally generate the quad mesh. Myles et al. [34] generate a parameterization aligned with a cross field by tracing along the field and then optimizing for consistent parametric edge lengths. Razafindrazaka et al. [38] formulate the problem of finding patch layouts as a global optimization on the cross field singularity graph. 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 1.3,8.4). 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 8.4a). 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.

2.4 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. Hildebrandt et al. [26] present a stable algorithm to produce visually pleasing feature lines, while Lai et al. [31] detect and classify features into ridges, valleys and prongs (a slender pointed part). These curves are often used to augment contours when generating sketches of input models, e.g. [17], and are effective at quickly conveying the overall shape of objects [15, 18] (Figure 8.2b). When aiming to convey proportions and geometric

details, artists utilize more detailed descriptive, or "precise", drawings [18] which augment contours and sharp feature curves with dominant flowlines and typically do not include smooth ridges or valleys (Figure 1.1). Our work focuses on capturing the curve networks artists use in the latter context (Figures 8.1, 8.2c).

Mehra et al. [33] 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 [14] (Figure 8.3b). Gehre et al. [22] 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 1.2).

2.5 Shape Proxies.

de Goes et al. [16] 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 [14] 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 8.5, 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 [13, 32]. This representation requires a leap of imagination to envision the intended surface, and performs worst when curvature streamlines are non-planar (Figure 8.5,e,h), leading Cignoni et al. [13] 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 8.5,f,i).

2.6 Analysis of Design Networks and Drawings.

There is a growing body of work on recovering 3D information from professional design drawings [27, 39, 45]. Iarussi et al. extrapolate line directions in designer drawings to form a cross field and then estimate the 3D surface normals. Shao et al. [39] use the intersections between sketch lines to infer the normals. Xu et al. [45] reconstruct 3D curve networks from design sketches using multiple insights from perception and design literature. Recent methods use such reconstructed or artist-created 3D networks to extract the 3D surface; Bessmeltsev et al. [3] interpolate the input curves with a set of quad patches whose iso-lines match the input network. Pan et al. [36] iteratively adapt a triangle mesh to optimize the agreement between its curvature field and the input. 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 Chapter 3.

Chapter 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 1.1). 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.

3.1 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 [40], validated by recent modeling research [3, 36], suggest that,

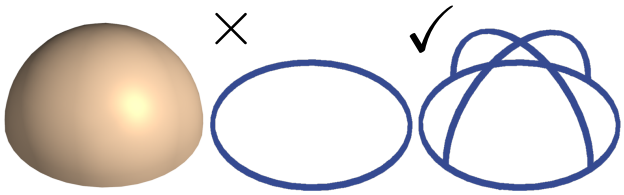

Figure 3.1: Descriptiveness; The curve network in the middle incorrectly conjures a flat surface, while the cycles on the right are descriptive of the originating surface on the left.

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 [8, 18] 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 Figure 3.1, 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.

3.2 Flowline Dominance

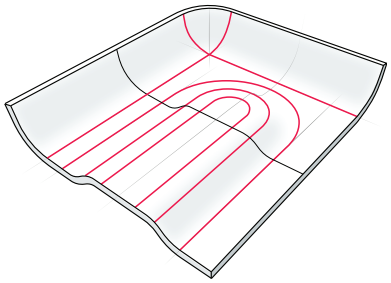

Figure 3.2: Examples of artist drawn lines to demarcate roundings.

Design literature [8, 18] 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 directions within these areas. As a specific example, Eissen

and Steur [2008] recommend that artists demarcate roundings (marked in red in Figure 3.2). 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.

3.3 Network Projectivity

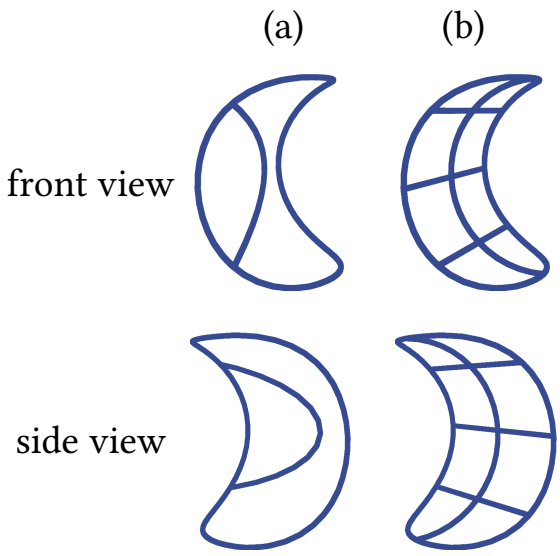

Figure 3.3: A less (a) and more (b) projectable curve network in two views. All figures show the same model.

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 [32]. In Figure 3.3 the network on the right satisfies this property, while the one on the left does not. Perception research [40] points out that smoothly crossing 2D curves in design drawings are universally perceived as *orthogonal*. As highlighted by [39, 45] 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 [45]. 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 [40, 43]. While restricting the set of network curves to strict planes reduces the set of models one can effectively describe [45], artists are strongly encouraged to use *planar* curves when depicting shapes [18], whenever possible.

Lastly, artists are strongly encouraged to draw local symmetry, or *geodesic* curves when depicting complex surfaces [18, 19]. The symmetry cue is known to be helpful in recovering the 3D shape of network curves from a 2D view [39, 45].

Chapter 4

Overview

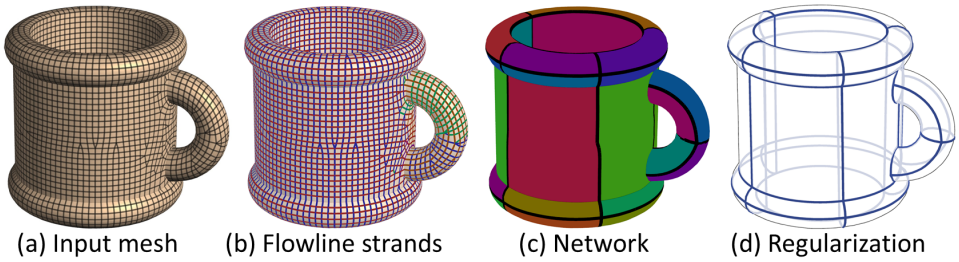

Figure 4.1: Algorithm stages: (a) Input quad-dominant mesh; (b) flowline strands; (c) compact descriptive network; (d) regularized final network.

Based on the criteria outlined in the previous chapter, I seek to construct a sparse cycle-descriptive network of trim curves and projective dominant flowlines. I 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, this thesis discretizes 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 [34, 37] and requires the consideration of subtle streamline seeding and termination choices [11, 34]. Existing methods for generating curvature-aligned quad-dominant meshes robustly address these issues, and their outputs can provide a suitable starting point for the method presented. Most of the edges in anisotropic regions on such meshes

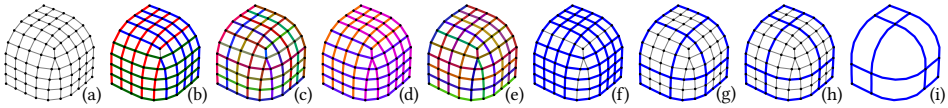

Figure 4.2: Detailed algorithm overview: (a) Input quad-dominant mesh; (b) initial flowline strands; (c) initial conservative flowlines; (d) final flowline strands; (e) reliable flowlines; (f) dense descriptive network; (g) simplified network; (h) network post local optimization; (i) regularized final network.

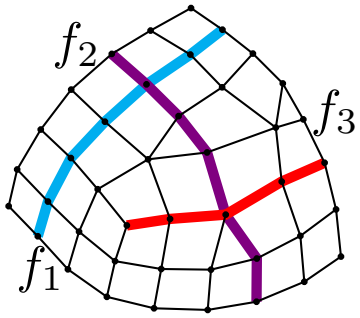

Figure 4.3: Examples of flowlines.

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 1.3c, Figure 2.1e); the task can therefore be formulated as extracting a compact subset of the mesh edges which maximally retains both properties. I 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 chapter, I 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 chapters.

4.1 Problem Statement

Given an input quad-dominant mesh M (Figure 4.2a), I formulate the computation of our target network N as selection of a subset of mesh edges with the following properties. Define a *flowline* as a (possibly closed) path made up of a sequence of vertex-adjacent edges (see Figure 4.3; three flowlines f_1, f_2, f_3 are identified by three different colors). Each flowline f is associated with positive projectivity and dominance costs p_f and d_f. These costs are designed to decrease as a flowline's projectivity or dominance increases. Each network cycle c is also associated with a descriptiveness error d_c. The exact formulations of p_f, d_f and d_c are detailed in Chapter 5. Using these measures, the 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:

$$\begin{aligned} \min E_N &= \sum (p_f + d_f) \\ \text{subect to } \max d_c &< d_{\max} \quad \forall c \in N \end{aligned} \tag{4.1}$$

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, the network computation operates on two distinct sets of entities. While the optimization function is defined on flowlines, or sequences of mesh edges, the constraints are defined on the cycles, or patches of mesh faces, that the flowlines bound. Second, unlike classical segmentation frameworks, the function 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.

4.2 Solution Framework

The algorithm developed in this thesis obtains 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 the final

network. Therefore I first group sequences of edges into flowlines (Figure 4.2b-e) and then select a subset of these flowlines, in combination with the surface trim curves, to assemble the desired network (Figure 4.2f-h). I contrast our approach against classical segmentation methods in Chapter 8.

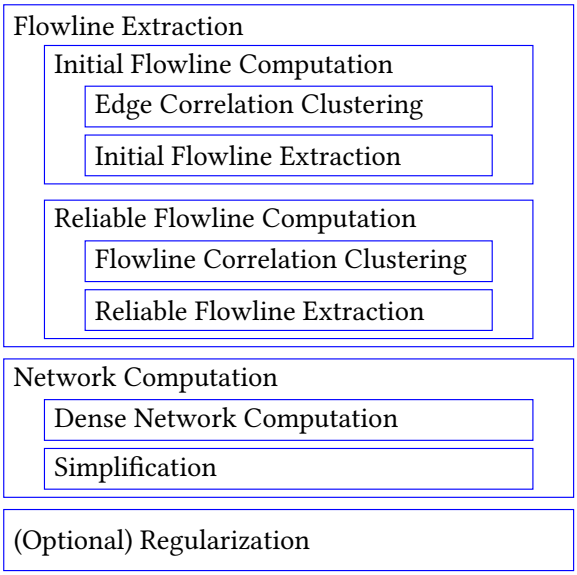

Figure 4.4: Overview of the method steps.

The following sections contain overviews of the solution framework steps (Figure 4.4), then Chapter 6 describes in detail flowline extraction and Chapter 7 presents network computation.

4.3 Strand-based Flowline Extraction

In general I 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. I 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

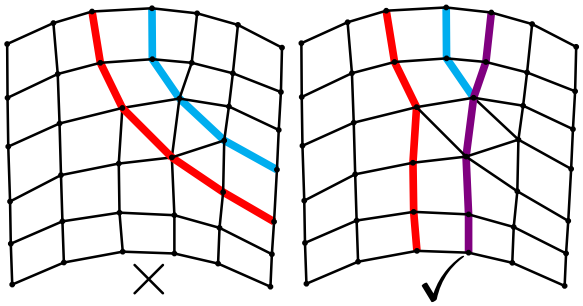

Figure 4.5: Flowlines extracted with just local information (left) and with global information (right).

(Figure 4.5, left) can introduce flowlines misaligned with dominant flow directions.

I obtain flowlines aligned with dominant flow (Figure 4.5, right) by taking global context into account. I 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*. FlowRep leverages this behavior by first extracting similarly directed flowline strands (Figure 4.2b-d) and then using these strands to extract individual, reliable flowlines (Figure 4.2e). I do not know *a priori* the number of strands we seek; however, as noted earlier, I 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. I use these observations to formulate strand extraction as a correlation clustering problem [2] (Figure 4.2d). I then use these strands to extract *individual, reliable* flowlines (Chapter 6, Figure 4.2e). Figure 4.6 demonstrates the differences between the local and global approaches on real-life inputs.

4.4 Network Computation

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

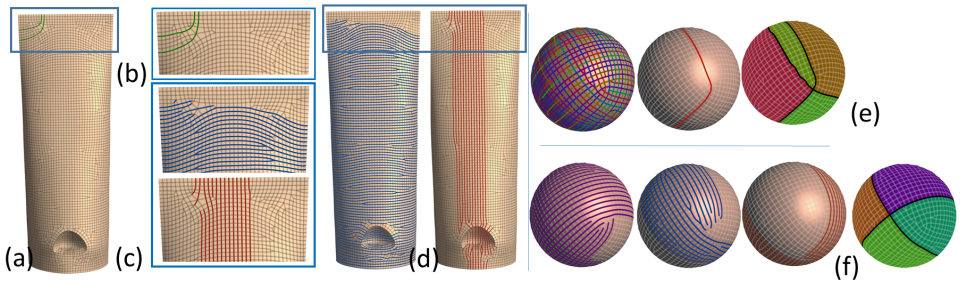

Figure 4.6: Independently formed flowlines (a,b,e) can be sub-optimal and may occasionally persist through network computation (e). Strand computation (c,d,f) correctly splits edges between different strands overriding purely local alignment and resulting in better final networks (f).

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 I adopt a mixed top-down/bottom-up strategy. The network construction process starts from a minimal network of only trim curves (Figure 7.1a) and progressively refines the inadequately described cycles on this network (Figure 7.1, Section 7.1). At each refinement stage I add all flowlines that span, or cross, these cycles into the network (Figure 7.1b), 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 4.2f).

At this point I have sufficient context to proceed with the simplification process, and can remove redundant flowlines (Section 7.2, Figures 4.1c, 4.2g, 7.1c). I greedily remove less dominant and less projectable flowlines while enforcing the descriptiveness threshold. I then further reduce the network energy E_N, subject

to the descriptiveness constraints, by reassessing local flowline selection (Figure 4.2h). 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.

4.5 Regularization

The work presented in this section (Section 4.5) was developed by Nicholas Vining in consultation with Alla Sheffer and is provided here for completeness.

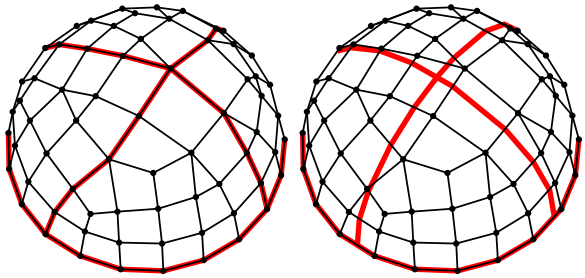

Figure 4.7: Regularization; Before (left) and after (right).

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 (Figure 4.7, 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 [41]. 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 4.2i).

Chapter 5

Measuring Network Properties

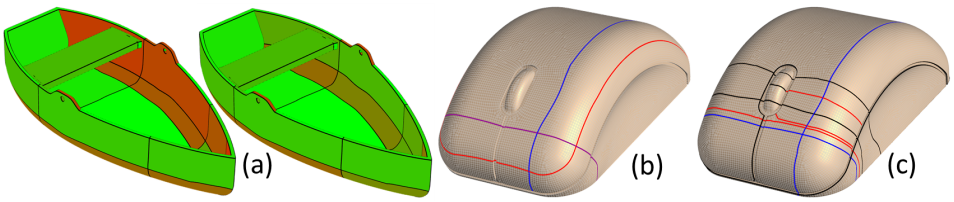

Figure 5.1: (a) More (green) and less (red) well described cycles on a rowboat, before and after local optimization. (b) Flowlines colored by decreasing projectivity (blue to red). (c) More (blue) and less (red) dominant flowlines.

The framework of this thesis seeks to solve the optimization problem described by Equation 4.1. I now derive the formulas used to measure the optimized energy and constraints. To enable meaningful combination of different values I express all quantities as angles (measured in degrees).

5.1 Cycle Descriptiveness

My descriptiveness metric assesses how well a given network curve cycle describes the region it surrounds (Figure 5.1a). Intuitively, I 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 re-

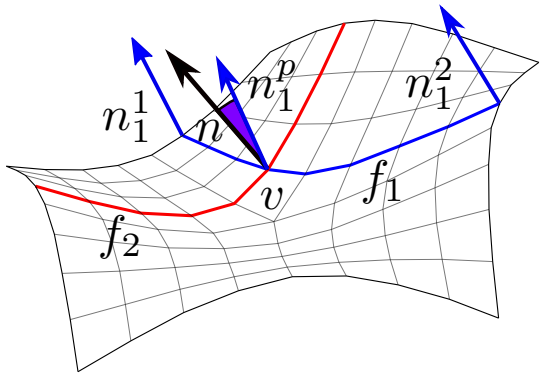

Figure 5.2: Coverage; At the vertex v, interpolating along the flowline f_1 the boundary normals n_1^1 and n_1^2 yield the prediction n_1^p. The error is the difference between n_1^p and the actual normal n. Similarly, another error is computed interpolating along f_2. FlowRep aggregates all of those errors for every vertex in the region.

gions. To enable conservative and robust descriptiveness computation I use differences in normals as proxy for curvature changes: in particular, I measure the angle differences between real normals across the patch and ones predicted based on normals along the cycles.

I predict interior normals based on the boundary by locally mimicking the method of Bessmeltsev et al. [2012]. Specifically, I follow each flowline f_i fully traversing the region from boundary to boundary. I use the normals at the cycle intersections n_i^1 and n_i^2 to obtain a normal prediction for each vertex v that belongs to the flowline and is inside the region (Figure 5.2):

$$n_i^p = \frac{d_1}{d_1+d_2} n_i^1 + \frac{d_2}{d_1+d_2} n_i^2 \tag{5.1}$$

Where d_i is the distance along the flowline from v to respective intersection point. I consider as the error the angle difference between the predicted normal and the actual surface normal n, which is defined as the average of the normals of the faces adjacent to the vertex *that belong to the region*:

$$d(v_i) = \angle(n_i^p, n). \tag{5.2}$$

In theory I 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 normals within the regions change gradually, if the region is sufficiently well spanned by flowlines, I found that it is safe to simply omit all such vertices from my region descriptiveness computation.

I seek a conservative descriptiveness estimate. Thus I refrain from using vertex descriptiveness average as the per cycle value, yet I also wish to avoid using the worst value as it may be an outlier. Thus, rather than using the largest angle $d(v)$ as the per-cycle error d_c, I collect all errors for the vertices along a flowline strand and compute their 90^{th} percentile value, then set d_c to the maximum of those *per strand* errors.

It is possible that a region does not have enough spanning flowlines, or even none at all. During the previous step I keep track of how many vertices were processed, and if more than a third of the total were missed, I then fallback to a simple planarity test. I compute the average of all face normals of the region, then set d_c as the maximum difference of each normal to the average normal.

5.2 Flowline Cost.

This thesis assigns 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.

5.2.1 Flowline Projectivity

I measure raw flowline projectivity by locally evaluating its planarity, its deviation from local geodesics, and its connections within the network. For each flowline f I 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 $k = 5$). I 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:

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

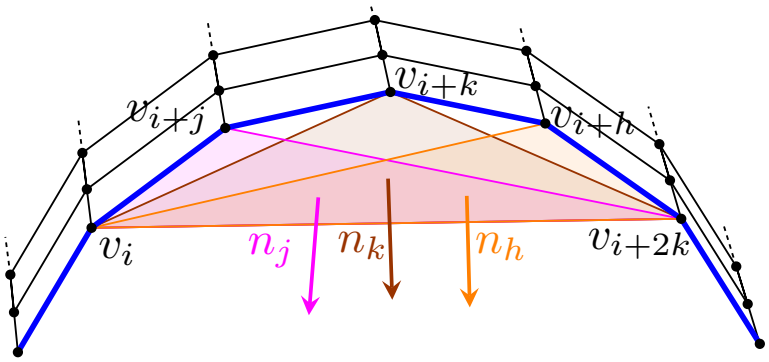

Figure 5.3: Projectivity; Three planes are fit to a sliding sequence of vertices from v_i to $v_i + 2k$. Their normals n_j, n_k, and n_h are used to compute planarity (as the difference between them) and geodesicity (comparing them against the vertex normals).

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

$$P(f) = \frac{1}{|i \in s|} \sum_{i \in s} P_i \tag{5.3}$$

Recall that a curve is locally considered a geodesic if its local fitting plane contains the surface normal. I 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 the angle measure becomes meaningless above a certain value, leading us to compute geodesicity as

$$G(f) = \frac{1}{|i \in s|} \sum_{i \in s} \min(\angle n_k n(v_{i+k}), G_m) \tag{5.4}$$

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. I 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. I

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.

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

$$p_f = P(f) + \frac{d_{\max}}{G_m} G(f) + E_1(f) + E_2(f) \tag{5.5}$$

5.2.2 Flowline Dominance

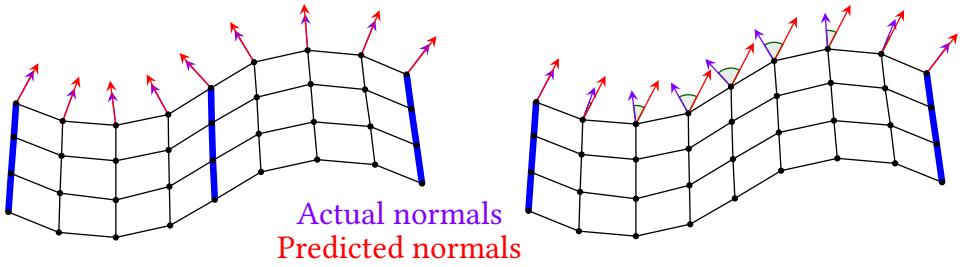

Figure 5.4: Normal predictions match the actual normals (left). Removing the middle flowline affects the normal predictions (right), that then diverge from the actual normals, leading to a high dominance cost.

Assessing dominance by measuring curvature continuity, as suggested by design literature, is unreliable in a discrete setting. Instead, I 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, I note that removing a network flowline impacts only one of these directions (Figure 5.4). Accordingly, to better pinpoint the impact of each network flowline, I use a modified cycle descriptiveness metric when assessing the impact of the removal. For each cycle resulting from removing a flowline f, I only consider the differences between the predicted and actual normal for normals predicted using flowlines crossing f, and ignore the differences along other flowlines. I then use the maximum of the cycle errors as a dominance estimate $D(f)$. Since I want the

cost to be smaller the more dominant a flowline is, we use

$$d_f = 360 - D(f) \tag{5.6}$$

as the cost. Figure 5.1 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, I require proxy values to predict flowline properties, as discussed in the next chapter.

Chapter 6

Flowline Computation

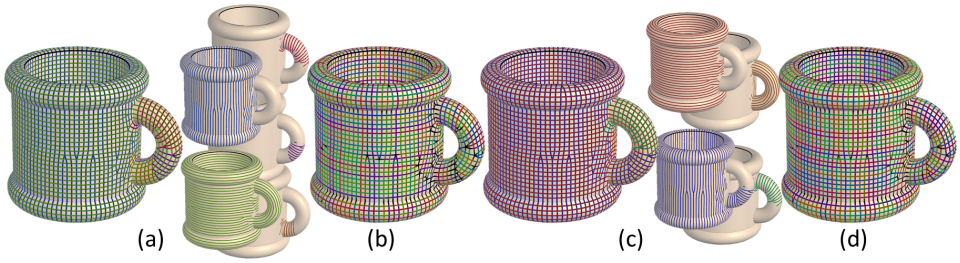

Figure 6.1: Dominant flowline strands on the mug: (a) initial edge strands and (b) extracted conservative flowlines; (c) final strands and (d) flowlines.

The goal of the flowline computation stage is to generate a set of *reliable* flowlines which we can use to assemble the network (see Chapter 7). Since purely local reasoning about individual flowline formation is unreliable I employ a global approach that leverages directional similarity between edges associated with adjacent flowlines (Figure 4.6). First I form strands, or clusters, of similarly directed edges, and then extract individual flowlines from these strands.

When forming strands I 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 (Figure 6.2).

They are expected to belong to different strands if they share common vertices

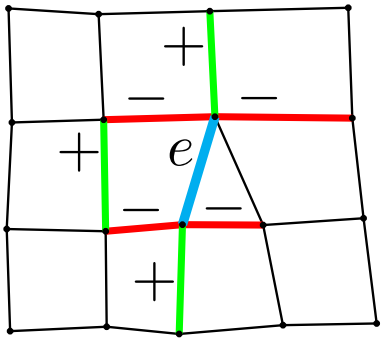

Figure 6.2: Positive and negative cluster associations of edges around edge e

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, I cannot apply it to our raw input consisting of individual edges. To take advantage of this cue, I employ a two-stage process: I first form *initial* strands using only cues applicable to edges and extract *initial* conservative flowlines from those; I 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 I use for strand formation naturally feeds into a correlation clustering framework [2]. I 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. I use the method of [30] to compute the clusters; while not optimal, it performs well in practice. The specific edge weights used in my two clustering steps are defined below.

6.1 Initial Strands and Flowlines.

6.1.1 Initial Strands

To form initial strands using correlation clustering (Figure 4.2b) mesh edges are treated as graph nodes, and I 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, I define the arc weight c_{ij} as

$$c_{ij} = \begin{cases} e^{-\left(\frac{\alpha_{ij}}{\sigma_1}\right)^2}, & \text{if } \alpha_{ij} \leq 45 \\ w_n e^{-\left(\frac{90-\alpha_{ij}}{\sigma_2}\right)^2}, & \text{otherwise} \end{cases} \tag{6.1}$$

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, I define the weight as

$$c_{ij} = w_p e^{-\left(\frac{\alpha_{ij}}{\sigma_3}\right)^2}. \tag{6.2}$$

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

6.1.2 Initial Flowline Extraction

I generate flowlines from strands by segmenting the connected components of each strand into individual flowline edge sequences (Figure 4.2c). I avoid making any potentially ambiguous choices by using all irregular (non valence 4) 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.

6.2.1 Reliable Strands

The initial flowlines allow for more global reasoning about, and consequently extraction of, more reliable strands and flowlines (Figure 6.1). I use a similar process for extracting these flowlines as for the initial ones, but incorporate additional information provided by the current segments. I 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, I 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. I then compute $c_{f,f'}$ as a function of the angle between these tangents using Equation 6.1, but with a more tolerant value, $\sigma_1 = 7.5$. For flowlines that share two endpoints I 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, I 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:

$$c_{(f,f')} = \frac{2}{l} \sum_{i,j} e^{-\left(\frac{\alpha_{ij}}{\sigma_3}\right)^2}. \tag{6.3}$$

As observed earlier, crossing flowlines should not belong to the same cluster. I 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 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.

6.2.2 Reliable Flowline Extraction

I use the obtained strands to extract extended reliable flowlines (Figure 4.2e). The connected components of flowline strands can form a range of graph configurations, allowing for multiple individual flowline configurations. I form our reliable flowlines using a greedy process which prioritizes extraction of more projectable and longer flowlines within each component. I 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 connected component I extract all closed loop flowlines first, prioritizing more projectable closed loops when given multiple options. I 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, I detect spirals in the same way as for conservative flowlines, and split them at the point where they complete a full cycle but have yet to become parallel. If there are multiple such points, I select the one(s) which result in the most projectable flowlines.

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. 6.1)
    for each edge j face-opposite to i **do**
      $G_1[i, j] \leftarrow c_{ij}$ (Eq. 6.2)
  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. 6.1, 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. 6.3)
  C_1 = CorrelationClustering(G_2) ▷ maximize $\sum_e c_e Y_e$
  Join connected flowlines in each $c \in C_2$ into *reliable flowlines*

Chapter 7

Network Computation

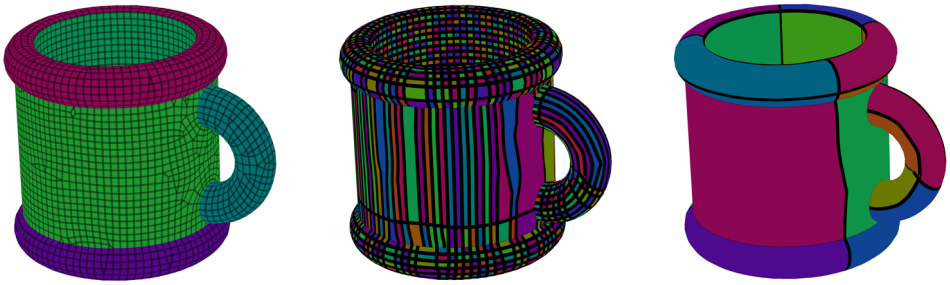

Figure 7.1: Network computation: (a) Initial trim curve network; (b) descriptive dense network; (c) final network.

I 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, I first progressively add flowlines to obtain a dense network that describes the input model within the given descriptiveness threshold $d_{\max}$ (Figure 4.2f, 7.1b); I then simplify it by removing redundant flowlines (Figure 4.2g, 7.1c).

7.1 Top-Down: Dense Descriptive Network Computation

7.1.1 Network Initialization

I compute feature curves on the input model using ridge and valley detection [46] with conservative settings designed to capture only sharp features. The boundary and extracted feature curves form the initial network, which I use to partition the input surface into a set of cycles surrounded by network curves (Figure 7.1a). I 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 $d_{\max}$. 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.

7.1.2 Network Refinement

I define a flowline as *spanning* a region if it splits it into two or more separate regions. I refine the network by iteratively incorporating flowlines that span currently uncovered regions: for each uncovered region, I detect all flowlines that span it; I add all located spanning flowlines into the network, shortening them as described below to avoid forming undesirable network topology. I then compute the cycle-descriptiveness error (Section 5.1) for all newly formed regions. The algorithm repeats the above step until all regions are covered or there are no more flowlines that can be added to the network.

7.1.3 Flowline Shortening

As noted in Chapter 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.

I maximize the projectivity of each open flowline we embed in the network, by collapsing open end-points to T-junctions (Figure 7.2, right) and collapsing purely flowline T-junctions to flowline-trim curve T-junctions (Figure 7.2, left), while constraining the shortened flowline to span the same set of uncovered regions. I first

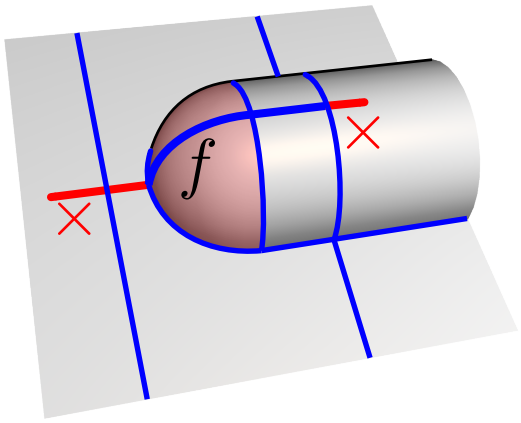

Figure 7.2: A flowlines is shortened to terminate at trim curves when possible (left) or to other flowlines (right).

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, I shorten it to the closest such intersection to its current end point. Similarly, if it ends at a valence 1 or 2 vertex, I shorten it to end at the nearest flowline intersection to the current end point.

7.2 Bottom-Up: Network Simplification

I 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 Chapter 4. In contrast to the original formulation, which operated by assigning edges to flowlines, I keep the flowlines computed in the previous stage largely fixed, and focus on selecting the optimal subset among them (Figure 4.2g).

My 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, my co-authors and I found that a greedy approach that mimics classical mesh sim-

plification, followed by local flowline movement, works sufficiently well for our needs.

7.2.1 Simplification

I 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 the descriptiveness constraint from the network. After each flowline is removed, I 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 the constraints. A typical simplification output is shown in Figure 7.1.

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

7.2.2 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. It is important to penalize valence reduction, but not completely prevent it. Prior to starting the simplification, I 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. I merge these sequences into *composite flowlines*. A composite flowline is a proxy for the flowline formed by its components, shares the same score and properties of a regular flowline and is placed in the same simplification queue. For each flowline within a composite flowline 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. The result is that the simplification algorithm can

remove a whole composite flowline in a single step, but it can also potentially remove a part of it, if appropriate. During simplification, when any such interior flowline is removed, I update the composite and the cost of the other flowlines interior to this composite accordingly.

7.2.3 Post-process Local Optimization

Post-simplification I locally further optimize the network as follows (Figure 4.2h, 5.1a). First, for each flowline, I 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. I perform the substitution if it decreases the network energy and the resulting network still satisfies the descriptiveness threshold. I repeat these steps as long as the energy decreases.

After local optimization, I 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, 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 the threshold. To avoid visual clutter, I 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. I select as a candidate a flowline whose removal increases the error the least. I remove this flowline if the increase in the descriptiveness error is less than 20%.

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) $\mid d_c < d_{\max}$ **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. 5.5, 5.6)
    $Q \leftarrow Q \cup \{f\}$
  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**
        $N \leftarrow ((N \setminus \{f\}) \cup \{f_n\})$

Chapter 8

Validation

The user studies presented in this chapter were conducted by Enrique Rosales and Nicholas Vining, in consultation with Alla Sheffer and myself. They are described here for completeness.

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.

8.1 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 8.1). 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.

To provide further comparison to artist-created networks, we qualitatively com-

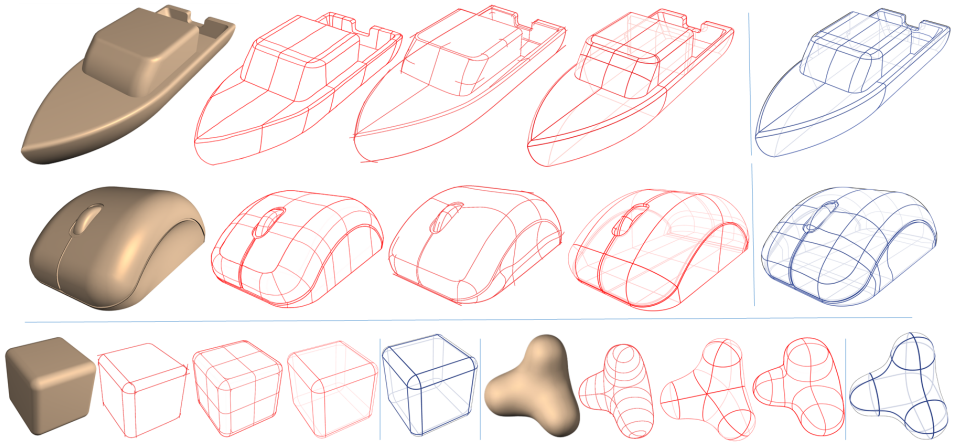

Figure 8.1: Comparison against artist generated networks: (left to right) input model, artist generated 2D design drawings and 3D network (red), and our algorithmic result (blue).

pared our output to the inputs used by [3, 36] (Figure 8.8, 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).

8.2 Comparison to Prior Art

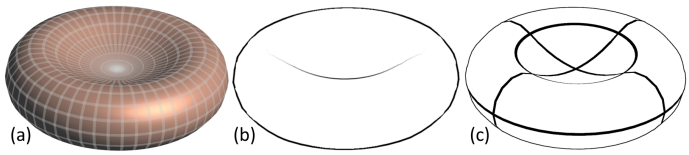

Figure 8.2: Suggestive contours combined with ridge and valley lines [17] (b) convey the overall input shape (a); a FlowRep descriptive network (c) provides a more detailed and accurate description of the input geometry.

Figures 8.2, 8.3, 8.4, and 8.5 show representative comparisons of our outputs against prior art. As Figure 8.2 demonstrates, while suggestive contours combined with ridge/valley lines convey the overall input shape, our networks provides a

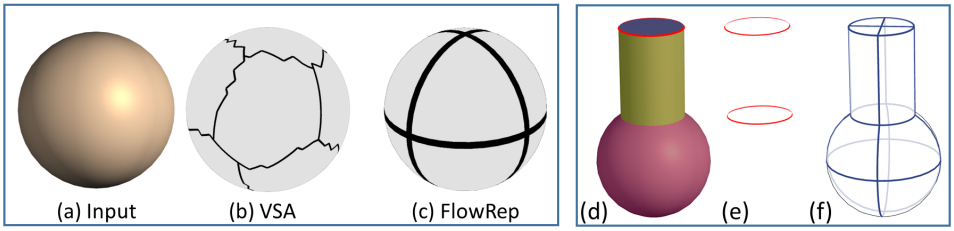

Figure 8.3: Surface segmentation (e.g. VSA [14]) (b) and reverse engineering methods (d,e) are not designed for, and do not produce, projectable curve networks; their output is often not descriptive to a human observer. FlowRep networks (c,f) satisfy both criteria.

more precise description of the input. Figure 8.3 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.

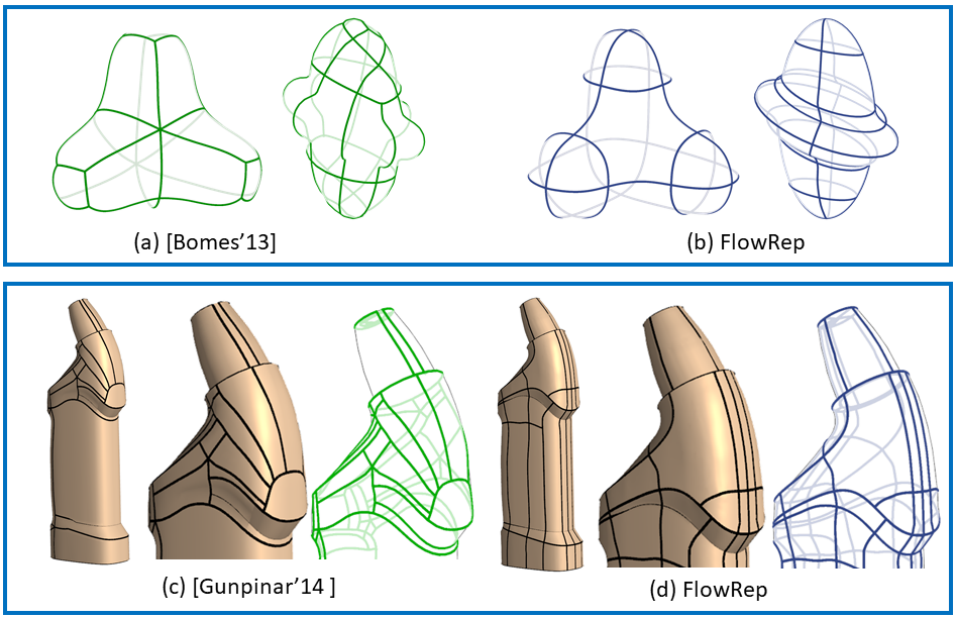

Figure 8.4: (a) Quad-partition [7] of the treball and ellipsoid (a) compared to our network (b). Quad partitions, here [24], are highly dependent on the singularity locations in the initial mesh drifting from curvature directions (c). FlowRep result on same model (d).

As demonstrated in Figures 1.3 and 8.4a, quad partition boundaries (e.g. [7]) 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 8.4c (network courtesy of [24]). Such misalignment results in non-descriptive cycles. Our networks are strictly aligned with curvature directions (Figure 8.4d) and enable viewers to envision the originating shape from a 2D projection.

While exoskeletons [16] reduce the number of T-junctions formed by unconstrained VSA [14], this semi-automatic method is only suitable for coarse abstraction, while our automatic framework captures much finer details on the same geometry (Figure 8.5, top). As highlighted by Figure 9.2, mimicking their bottom-up approach using an automatic process results in fragmented flowlines. Lastly, as shown in Figure 8.5 (bottom) planar slice proxies [32], while descriptive of multi-part simple shapes, compare poorly to our networks on typical design shapes.

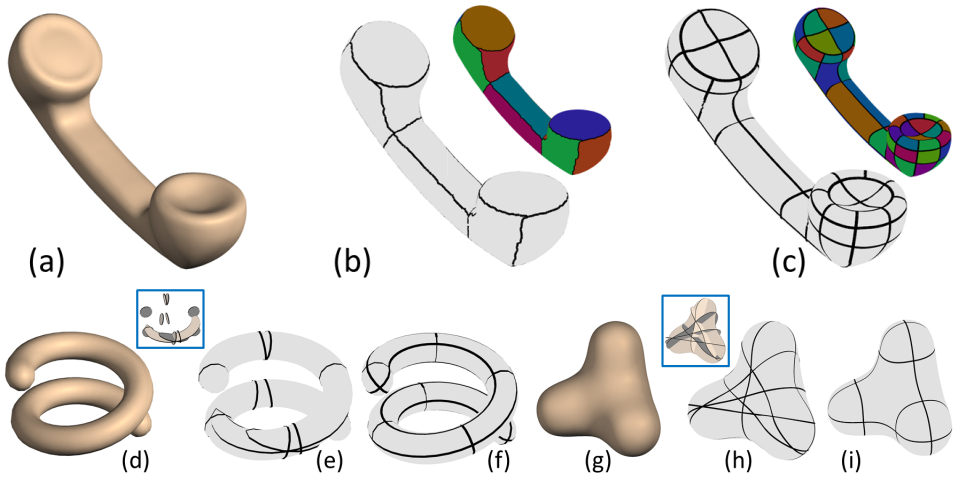

Figure 8.5: While exoskeletons [16] only roughly capture coarse part structures of shapes (b), our method describes the geometry in more detail (c). Planar slices [32] are restricted in their ability to convey free-form shape (e,h), while FlowRep networks are well suited for this task (f,i).

8.3 Qualitative Evaluation

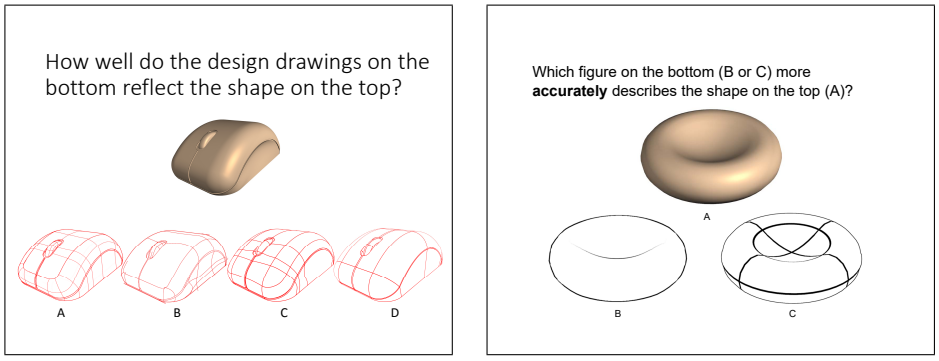

Figure 8.6: Study questionaire layouts: FlowRep compared to artists (left) and FlowRep compared to previous work (right).

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 (Figure 8.6, left). We then asked the designers "How well do the design drawings on the bottom reflect the shape on the top?". 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 [17], quad patch layouts generated by [7] and [24], variational shape approximation [14], exoskeletons [16], and planar slices [32], totaling eight queries.) (Figure 8.6, right). 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 [7] on the ellipsoid (Figure 8.4). 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.

8.4 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 [36] on a subset of our outputs (Figure 8.7). 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 [12] 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 en-

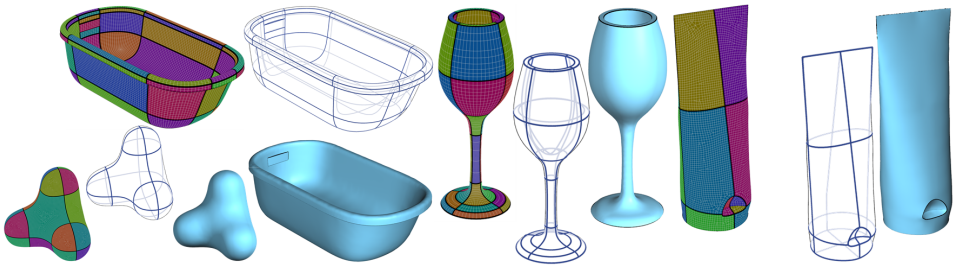

Figure 8.7: Reproduction. Pairs of input models with computed FlowRep networks (wireframes) and these networks resurfaced using [36] (blue).

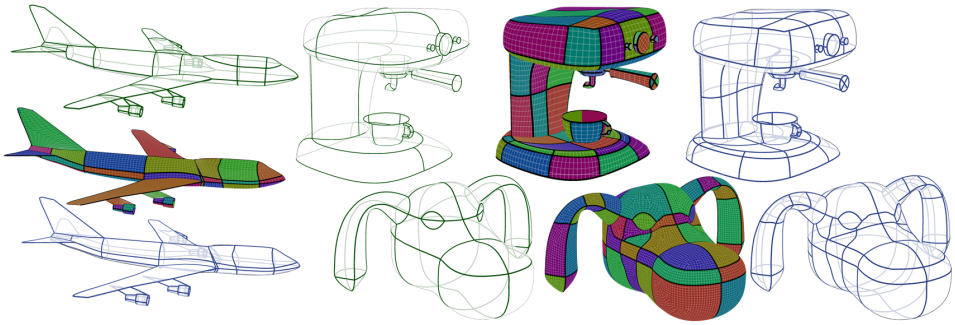

Figure 8.8: Reproduction. Input curve networks (green), surfaces produced by [3, 36], and our networks (blue) computed from these surfaces.

code their geometry.

Chapter 9

Results

This thesis demonstrated my method on a large set of inputs, ranging from simple (rounded cube, spiral, torus, bump) to highly complex (shuttle, boats, mouse, toilet). I tested various models including open surfaces (beetle, bathtub), ones with sharp features (mouse, row-boat), and smoother, more organic ones (treball, spiral, ellipsoid, phone handle, dog-head, big buck bunny). I tested my method on coarsely meshed complex shapes (beetle, at 4K triangles) as well as fine meshed ones (mouse, boat, toilet, at 30K triangles each). My results on all these models are consistent with human expectations.

9.1 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.

9.2 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 9.1. By default we use a threshold of 20°. For the rowboat and wineglass we used thresholds of

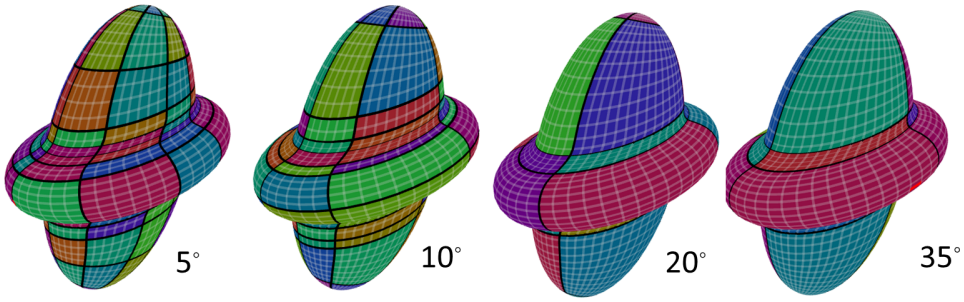

Figure 9.1: Networks with different descriptiveness thresholds, shown on the ellipsoid model.

30° and 10° to obtain more aesthetically pleasing results. We use 30° for more organic models (doghead, phone).

9.3 Impact of Design Choices

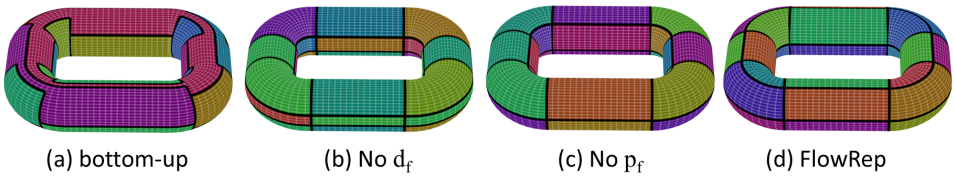

Figure 9.2: Alternative network extraction methods: (a) bottom-up cycle clustering results in poorly descriptive networks with highly irregular connectivity; network optimization using only flowline projectivity (b) or only their dominance (c), versus full optimization (d).

Figures 4.6 and 9.2 demonstrate the impact of the key algorithmic choices on the final results. Figure 4.6 demonstrates the importance of using my strand formation process to obtain reliable flowlines. This process contributes to my success in processing highly irregular meshes, with singularities and numerous edges which are misaligned with curvature directions. Figure 9.2 shows a comparison against a number of alternative network formation strategies. The first mimics [16] as it uses a bottom up cycle growth strategy directly optimizing E_N (Equation 4.1) 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 9.2b 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 the descriptiveness threshold, but select different representative flowlines within individual strands. Absent dominance (Figure 9.2b) the result has more geodesic flowlines but has some less well described cycles; in contrast in Figure 9.2c the cycles are well described but the interior symmetry curve is not included. My solution (Figure 9.2d) balances both sets of criteria, and is more reflective of traditional drawings of such toroidal shapes.

9.4 Input Mesh Impact

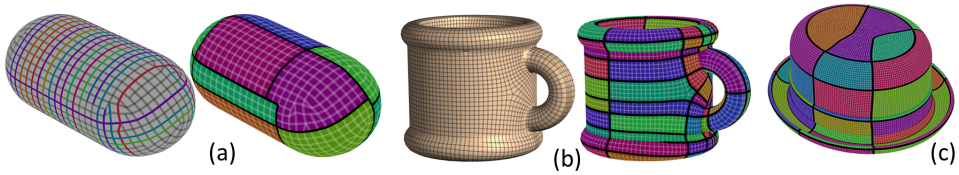

Figure 9.3: Impact of systemic mesh vs curvature tensor misalignment: (a) Interleaved spiral flowlines resulting from curvature drift may result in undesirable T-junctions (mesh from [28]); (b) more significant misalignment predictably distorts the network orientation (mesh from [7]); (c) large patches of randomly aligned quads (top of the hat) similarly lead to artifacts.

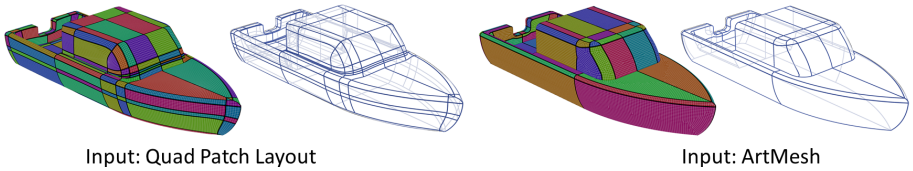

Figure 9.4: Curve networks generated from different meshes of same model: (a) [7], (b) ArtMesh. The input mesh in (a) exhibits feature drift (sharp features migrating between mesh flowlines); leading FlowRep to preserve these flowlines generating visual redundancy.

I tested my 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 me in quad-mesh format (spiral, glass, bathtub, beetle). The plane and coffee-machine in Figure 8.8 were generated by [3]. I also tested inputs generated using quad-patch layout [7] (Figures 9.4, 9.3b) and Instamesh [28] (phone handle, and Figure 9.3a). As the examples demonstrate, FlowRep is largely agnostic to mesh artifacts, such as non-quad elements, irregular connectivity, and uneven element sizing, e.g. Figure 1.3b. The method is more sensitive to consistent curvature misalignment and irregular mesh flow in isotropic areas (hat, Figure 9.3c). Input meshes whose edges form long spirals due to systemic edge direction drift [28] may lead to extra T-junctions in the final output (Figure 9.3a). 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. [7]), the results in this case are, as expected, less reflective of human expectations (Figure 9.3b).

9.4.1 Curvature re-alignment

ArtMesh meshes are generated with quad shape and size in mind, often sacrificing edge alignment to curvature for those properties. For this thesis the regularity of the quads is not as important as the alignment to the flow, so I need to undo that process and optimize for better alignment.

Consecutive segments of the same flowline at any given point on the surface tend to share the same tangent and the tangents of intersecting streamlines to be orthogonal, up to discretization accuracy. I therefore wish for all angles on the mesh to be as close as possible to either 90° or 180°.

For each vertex v, consider the angle α described by one of its adjacent faces f. We find whether α is closer to 90° or 180° and compute its difference ω to that.

$$\omega = \begin{cases} \alpha - 90^\circ & \text{if } \alpha < 135^\circ \\ \alpha - 180^\circ & \text{otherwise} \end{cases} \tag{9.1}$$

Then, I simply push or pull v away or to the center of the face, according to ω.

$$v = v + \omega(v - \text{center}(f)) \tag{9.2}$$

I repeat the above for every face of every vertex with valence 4 or less. Higher valence vertices are often misleading, meaning that the procedure would improve the wrong angle, and are best left untouched.

9.5 Parameters and Runtimes

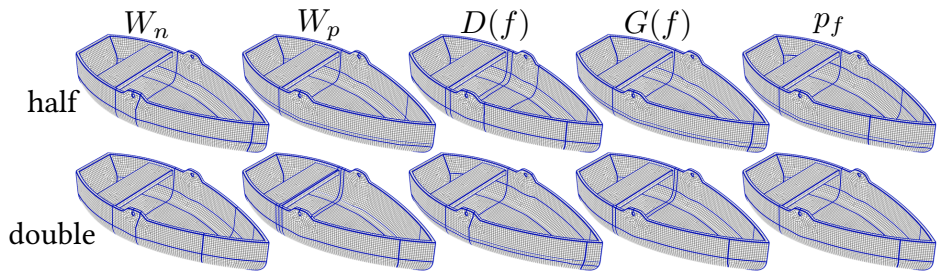

Figure 9.5: Impact of halving/doubling default algorithm parameters (weights used for correlation clustering; weights of different elements of E_n (Eq. 4.1)).

With the exception of the descriptiveness threshold $d_{\max}$ all other algorithm parameters are fixed across all inputs, and as shown in Figures 9.2, 9.5 changing them has fairly minimal impact on the outcome.

FlowRep 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.

9.6 Limitations

My 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 1.3) but is affected by systemic misalignment and uneven mesh sizing (Figure 9.3). The

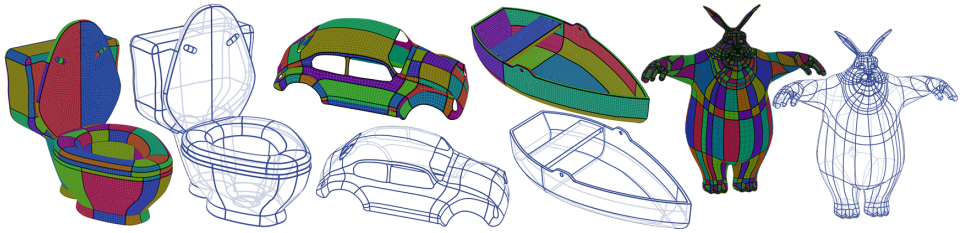

Figure 9.6: Additional results.

trade-off I require is supported by many research and commercial meshers. My 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.

Chapter 10

Conclusions

This thesis presented FlowRep, the first algorithm for computing descriptive compact curve networks from an input mesh. My output networks succinctly describe complex free-form shapes and are suitable for both perceptual and algorithmic reconstruction. When computing the networks my 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. I 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, the results are contingent on the quality of the input quad meshes. Producing initial meshes tuned to optimally adhere to my requirements can both simplify the algorithm and improve its results. Mine 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 I have identified.

Bibliography

[1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic Polygonal Remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003. → pages 6

[2] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004. ISSN 0885-6125. → pages 19, 29

[3] M. Bessmeltsev, C. Wang, A. Sheffer, and K. Singh. Design-driven quadrangulation of closed 3d curves. *ACM Trans. Graph.*, 31(6): 178:1–178:11, 2012. ISSN 0730-0301. → pages xi, 3, 4, 10, 11, 23, 40, 45, 49

[4] R. Bnire, G. Subsol, G. Gesquire, F. L. Breton, and W. Puech. A comprehensive process of reverse engineering from 3d meshes to CAD models. *Computer-Aided Design*, 45(11):1382 – 1393, 2013. ISSN 0010-4485. → pages 3, 7

[5] D. Bommes, T. Vossemer, and L. Kobbelt. Quadrangular parameterization for reverse engineering. volume 5862 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2008. → pages 7

[6] D. Bommes, T. Lempfer, and L. Kobbelt. Global structure optimization of quadrilateral meshes. *Computer Graphics Forum*, 30(2):375–384, 2011. → pages 8

[7] D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, and L. Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.*, 32(4):98:1–98:12, 2013. ISSN 0730-0301. → pages viii, xi, xii, 4, 5, 8, 42, 44, 48, 49

[8] M. Bordegoni and C. Rizzi. *Innovation in product design*. Springer, 2011. → pages 12

[9] M. Campen and L. Kobbelt. Dual strip weaving: Interactive design of quad layouts using elastica strips. *ACM Trans. Graph.*, 33(6), 2014. → pages 7

[10] M. Campen, D. Bommes, and L. Kobbelt. Dual loops meshing: Quality quad layouts on manifolds. *ACM Trans. Graph.*, 31(4):110:1–110:11, 2012. ISSN 0730-0301. → pages 3, 8

[11] M. Campen, M. Ibing, H.-C. Ebke, D. Zorin, and L. Kobbelt. Scale-Invariant Directional Alignment of Surface Parametrizations. *Computer Graphics Forum*, 2016. → pages 5, 15

[12] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 1998. → pages 44

[13] P. Cignoni, N. Pietroni, L. Malomo, and R. Scopigno. Field-aligned mesh joinery. *ACM Trans. Graph.*, 33(1):11:1–11:12, 2014. ISSN 0730-0301. → pages 9

[14] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004. ISSN 0730-0301. → pages xi, 3, 7, 9, 41, 42, 44

[15] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. Where do people draw lines? *ACM Trans. Graph.*, 27(3), 2008. → pages 8

[16] F. De Goes, S. Goldenstein, M. Desbrun, and L. Velho. Exoskeleton: Curve network abstraction for 3d shapes. *Comput. Graph.*, 35(1):112–121, 2011. ISSN 0097-8493. → pages xi, 9, 42, 43, 44, 47

[17] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003. → pages xi, 3, 8, 40, 44

[18] K. Eissen and R. Steur. *Sketching: Drawing Techniques for Product Designers*. Bis Publishers, 2008. → pages 4, 8, 9, 12, 13, 14

[19] K. Eissen and R. Steur. *Sketching: The Basics*. Bis Publishers, 2011. → pages 14

[20] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. In *Proceedings of the Symposium on Geometry Processing*, pages 1477–1486, 2008. → pages 3, 8

[21] R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*, 28 (3):#33, 1–10, 2009. → pages 1

[22] A. Gehre, I. Lim, and L. Kobbelt. Adapting Feature Curve Networks to a Prescribed Scale. *Computer Graphics Forum*, 2016. → pages 1, 3, 9

[23] G. Gori, A. Sheffer, N. Carr, T. Ju, N. Vining, and E. Rosales. Flowrep: Descriptive curve networks for free-form design shapes. *ACM Transaction on Graphics*, 36(4), 2017. doi:http://dx.doi.org/10.1145/3072959.3073639. → pages iv

[24] E. Gunpinar, M. Moriguchi, H. Suzuki, and Y. Ohtake. Feature-aware partitions from the motorcycle graph. *Comput. Aided Des.*, 47:85–95, 2014. ISSN 0010-4485. → pages xi, 8, 42, 44

[25] E. Gunpinar, M. Moriguchi, H. Suzuki, and Y. Ohtake. Motorcycle graph enumeration from quadrilateral meshes for reverse engineering. *Comput. Aided Des.*, 55:64–80, 2014. ISSN 0010-4485. → pages 8

[26] K. Hildebrandt, K. Polthier, and M. Wardetzky. Smooth feature lines on surface meshes. In *Proc. SGP 2005*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. ISBN 3-905673-24-X. → pages 8

[27] E. Iarussi, D. Bommes, and A. Bousseau. Bendfields: Regularized curvature fields from rough concept sketches. *ACM Trans. Graph.*, 2015. → pages 10

[28] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph*, 34(6), 2015. → pages xii, 48, 49

[29] D. Julius, V. Kraevoy, and A. Sheffer. D-Charts: Quasi-Developable Mesh Segmentation. *Computer Graphics Forum*, 2005. → pages 7

[30] M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proc. ICCV*, 2015. → pages 29

[31] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, J. Wallner, and H. Pottmann. Robust feature classification and editing. *IEEE Trans. Vis. Comp. Graphics*, 13(1): 34–45, 2007. → pages 8

[32] J. McCrae, K. Singh, and N. J. Mitra. Slices: A shape-proxy based on planar sections. *ACM Trans. Graph.*, 30(6), 2011. → pages xi, 9, 13, 42, 43, 44

[33] R. Mehra, Q. Zhou, J. Long, A. Sheffer, A. Gooch, and N. J. Mitra. Abstraction of man-made shapes. *ACM Trans. Graph.*, 28(5), 2009. → pages 1, 3, 9

[34] A. Myles, N. Pietroni, and D. Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):135:1–135:14, 2014. ISSN 0730-0301. → pages 3, 8, 15

[35] M. Nieser, C. Schulz, and K. Polthier. Patch layout from feature graphs. *Comput. Aided Design*, 42(3), 2010. → pages 7

[36] H. Pan, Y. Liu, A. Sheffer, N. Vining, C. Li, and W. Wang. Flow aligned surfacing of curve networks. *ACM Trans. Graph.*, 34(4), 2015. → pages xi, 3, 4, 10, 11, 40, 44, 45

[37] N. Ray and D. Sokolov. Robust polylines tracing for n-symmetry direction field on triangulated surfaces. *ACM Trans. Graph.*, 33(3):30:1–30:11, 2014. ISSN 0730-0301. → pages 15

[38] F. H. Razaflndrazaka, U. Reitebuch, and K. Polthier. Perfect matching quad layouts for manifold meshes. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 219–228, 2015. → pages 8

[39] C. Shao, A. Bousseau, A. Sheffer, and K. Singh. Crossshade: shading concept sketches using cross-section curves. *ACM Trans. Graph.*, 31(4), 2012. → pages 10, 13, 14

[40] K. A. Stevens. The visual interpretation of surface contours. *Artificial Intelligence*, 17, 1981. → pages 11, 13, 14

[41] V. Surazhsky and C. Gotsman. High quality compatible triangulations. *Eng. Comput. (Lond.)*, 20(2):147–156, 2004. → pages 21

[42] M. Tarini, E. Puppo, D. Panozzo, N. Pietroni, and P. Cignoni. Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph.*, 30(6): 142:1–142:12, 2011. ISSN 0730-0301. → pages 8

[43] J. Todd and F. Reeichel. Visual perception of smoothly curved surfaces from double-projected contour patterns. *J. of Exp. Psych.: Human Percep. and Performance*, 16:665–674, 1990. → pages 14

[44] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005. → pages 7

[45] B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. Mccrae, and K. Singh. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Trans. Graph.*, 33(4), 2014. → pages 3, 10, 13, 14

[46] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. Fast and robust detection of crest lines on meshes. In *Proc. Symp. on Solid and Physical Modeling*, SPM '05, pages 227–232, 2005. → pages 34

[47] Y. Zhuang, M. Zou, N. Carr, and T. Ju. Anisotropic Geodesics for Live-wire Mesh Segmentation. *Computer Graphics Forum*, 2014. ISSN 1467-8659. → pages 7

Appendix A

Supporting Materials

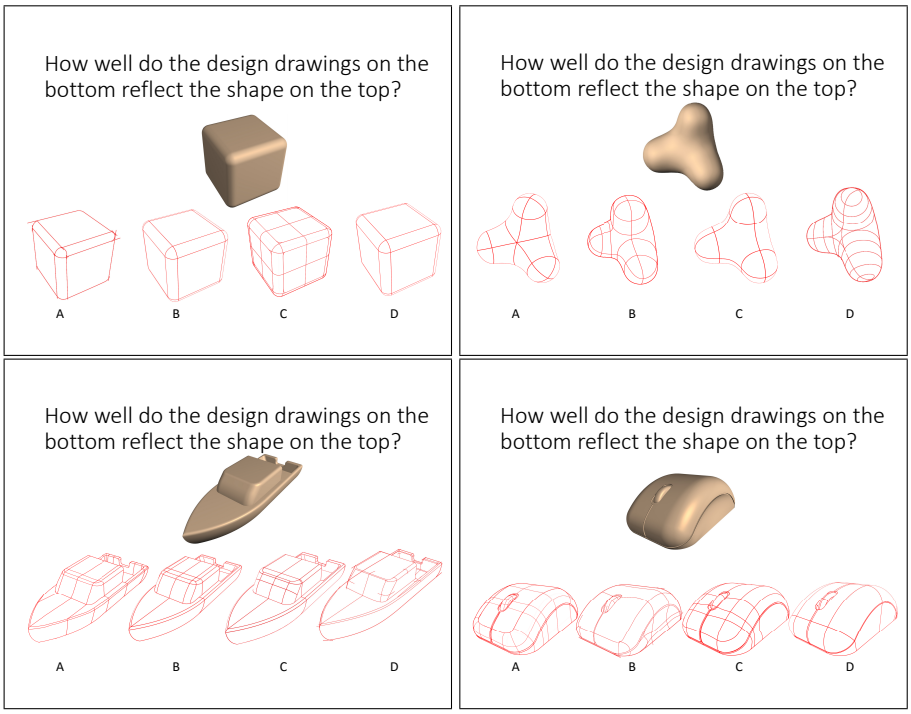

Figure A.1: User Study 1.

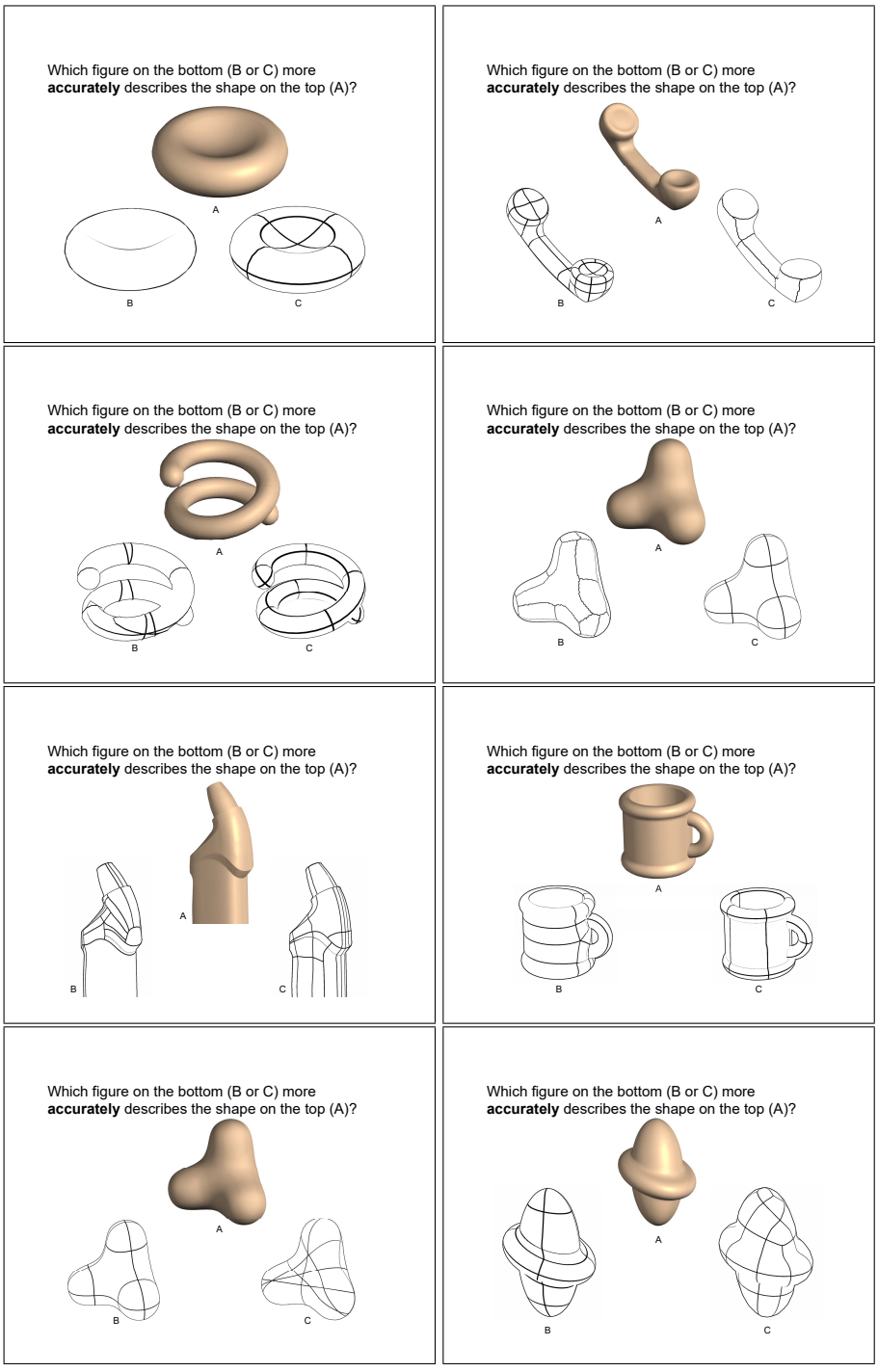

Figure A.2: User Study 2.