# Design-Driven Quadrangulation of Closed 3D Curves

Mikhail Bessmeltsev[1] Caoyu Wang[1] Alla Sheffer[1] Karan Singh[2]

[1]University of British Columbia [2]University of Toronto

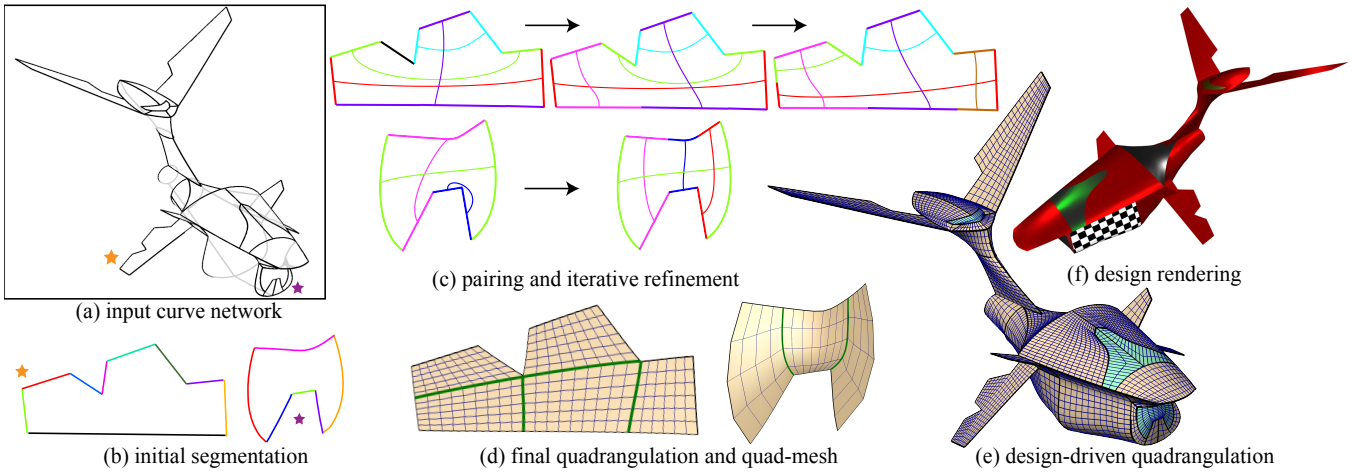



**Figure 1:** *Steps to quadrangulating a design network of closed 3D curves (a) : Closed curves are independently segmented (b) and iteratively paired and refined to capture dominant flow-lines as well as overall flow-line quality (c); final quadrangulation in green and dense quad-mesh (d); quadrangulations are aligned across adjacent cycles to generate a single densely sampled mesh (e), suitable for design rendering and downstream applications (f).*

## Abstract

We propose a novel, design-driven, approach to quadrangulation of closed 3D curves created by sketch-based or other curve modeling systems. Unlike the multitude of approaches for quad-remeshing of existing surfaces, we rely solely on the input curves to both conceive and construct the quad-mesh of an artist imagined surface bounded by them. We observe that viewers complete the intended shape by envisioning a dense network of smooth, gradually changing, *flow-lines* that interpolates the input curves. Components of the network bridge pairs of input curve segments with similar orientation and shape. Our algorithm mimics this behavior. It first segments the input closed curves into pairs of *matching* segments, defining dominant flow line sequences across the surface. It then interpolates the input curves by a network of quadrilateral cycles whose iso-lines define the desired flow line network. We proceed to interpolate these networks with all-quad meshes that convey designer intent. We evaluate our results by showing convincing quadrangulations of complex and diverse curve networks with concave, non-planar cycles, and validate our approach by comparing our results to artist generated interpolating meshes.

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—;

**Keywords:** curve-based modeling, flow-lines, quad-meshing

**Links:** DL PDF

## 1 Introduction

Curves are a quintessential primitive of visual communication dating back millenia to early cave drawings. In current 3D design practice, curves are universally used to depict both form and function, conveying the essence of a shape [Bordegoni and Rizzi 2011]. Sparse networks of closed 3D curves are indeed the foundation of shape in both, traditional CAD modeling [Farin and Hansford 1999] and increasingly popular sketch-based modeling interfaces [Bae et al. 2008; Nealen et al. 2007; Schmidt et al. 2009]. Unsurprisingly, recent research affirms that such 3D curve networks do effectively convey complex 3D shape [Mehra et al. 2009; de Goes et al. 2011; McCrae et al. 2011] (Figure 1 (a)). We aim to recover and compactly represent this conveyed shape (Figure 1 (f)), for designer-drawn curve networks, such as those generated by Abbasinejad et al. [2011] from sketched 3D curves [Bae et al. 2008].

While arbitrary 3D curve cycles have highly ambiguous interpolating surfaces (Figure 2 (top)), designer created curve cycles, even when highly complex, typically convey a uniquely imagined surface (Figure 2 (bottom)). These curves are designed to serve as a visual proxy of the 3D object, with the expectation that every element of surface detail is explicitly captured by the network [Gahan 2010]. To this end, design texts repeatedly emphasize the significance of

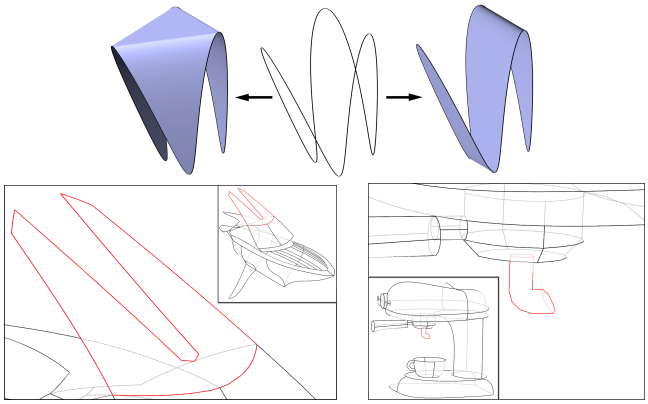
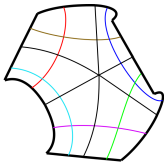


**Figure 2:** *Closed 3D curves: ambiguous hexagonal 3D curve (top) compared to complex curves with a clear design intent (bottom).*

using *representative flow-lines* of the object [Bordegoni and Rizzi 2011; Gahan 2010], as curve network elements. While design literature provides no precise mathematical definition of flow-lines, design and modeling references [Gahan 2010; Singh et al. 2004] suggest that flow-lines are strongly correlated to sharp features and lines of curvature but allow for artistic license at surface discontinuities, over fine details and in umbilic regions.

These observations, confirmed by perception studies [Stevens 1981], suggest that any additional flow-line on the surface must be expressible as a blend of the explicitly defined flow lines on the designer-created curve cycles. Viewers complete the intended shape by envisioning a dense network of such blended, gradually changing flow-lines. An examination of artist-drawn dense networks (e.g. Figure 3) confirms this observation; moreover, artists take advantage of this property by implicitly pairing opposite representative flow-lines, and constructing curve sequences that smoothly evolve from one input flow-line to its mate along the interior surface. The resulting surface is described by the union of these sequences, and forms a quad-dominant mesh. Consistent with this examination, popular CAD tools capture the geometry of four-sided curve cycles using Coons patches [Coons 1964] (Figure 4 (d)), whose iso-lines implicitly define a sequence of flow-lines that bridge the opposite sides of each cycle.

Our surface-fitting algorithm aims to replicate this behavior. Before formally describing the algorithm, we introduce some terminology. A *3D curve network* is a graph of connected 3D curves, where one or more curve cycles have been marked for surfacing by the designer. A *quadrangulated curve network* (left, black) requires that all curve cycles marked for surfacing be four-sided.



A single *quad-mesh* can be created from quadrangulated curve network cycles by sampling parametric four-sided patches. The *dual* of a quad network is a graph whose vertices correspond to quad cycles, and whose edges correspond to shared cycle sides. Each dual *poly-chord*, drawn on the left in a different color, is a sequence of dual edges that corresponds to a chain of quadrilaterals sharing opposite sides [Daniels et al. 2008].

If we can extract the flow-line pairings that artists use, we can then reconstruct the surface in a natural manner using a dual quadrangulation approach, described below. The grand challenge, therefore, is in obtaining a suitable segmentation of a curve cycle into pairs of matching opposite flow-lines. As we expect internal flow lines to change smoothly and gradually, these *bridged* segment pairs should have similar orientation and shape. When examining artist generated flow-networks, we observe that the preference for pairing segments becomes more pronounced as the degree of compatibility between them increases, often at the expense of sub-optimal pairing of other segments. This effect is evident in the highlighted regions in Figure 3. On the left, the strong preference for the blue pair enforces the far less obvious red one. On the right, the dominant yellow and blue matches enforce the far less attractive purple one. Such dominant preference order can be formally described as a *stable matching*, where a matching is considered stable when there are no two elements that prefer each other to their current match [Irving 1985]. We simultaneously compute the segmentation and its corresponding stable pairing using a tailored discrete optimization strategy which interleaves matching and segmentation steps.

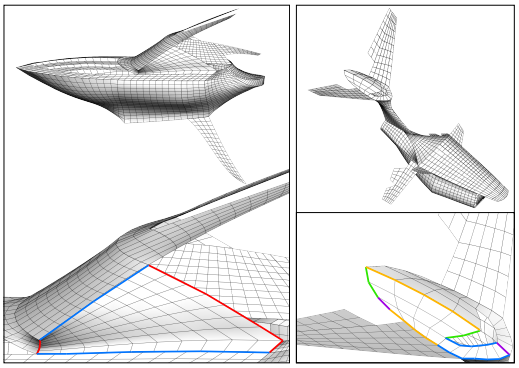


**Figure 3:** *Artist designed interpolating quad-meshes.*

Given the computed segmentation and pairing (Figure 1 (c)), we construct a network of quadrilateral cycles (Figure 1 (d)) whose dual poly-chords connect the matched flow-line curve segments and interpolate those with tensor-product surface patches. Using this construction, the iso-lines of the patches naturally align with the matched curves, forming a dense flow-line network conveying the intended surface. An arbitrarily dense quad-mesh describing the target shape is then created by tracing patch iso-lines (Figure 1 (f)).

We demonstrate the quad meshes created by our method on a variety of challenging inputs, including both synthetic models and curve networks created by different modeling softwares, comparing our outputs against those manually created by design professionals (Section 5).

**Contribution:**

We present the first solution to constructing the imaginary surface interpolating a general 3D design curve network. We represent this surface using a quad mesh whose iso-lines capture the design flow inherent in the network. Lacking a mathematical model of human perception, we distill perception studies and guidelines from design literature into a mathematical formulation of flow-line matching and segmentation. We evaluate this formalism by showing results that match both viewer expectation and artist created surfaces. Our key technical innovation is a simultaneous segmentation and pairing algorithm that locates suitable end segments for the dual poly-chords of the interpolating quad mesh based on analysis of the input curve geometry.

Quad-remeshing techniques often strive to generate rectangular quad elements. We note that our primary objective is to capture flow-lines; since these lines are often related to lines of curvature, we will typically generate well-shaped quads. However, when flow-lines conflict with quad orthogonality, we focus on capturing the flow at the expense of irregularly shaped quads (see Figure 1). This ensures that our output is consistent with designer expectations (Figure 3).

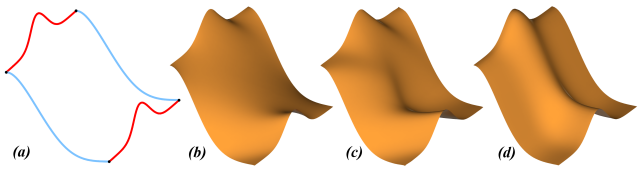



**Figure 4:** *Using Laplacian diffusion (b) or Thin-Plate Splines [Finch and Hoppe 2011](c) to surface a four-sided cycle leads to unintuitive results. (d) In contrast the flow lines on an interpolating Coons patch, by construction, bridge opposite cycle sides.*

## 2 Related Work

We build on previous research in the areas discussed below.

**Quad Meshing:** Our work draws on ideas from coarse-to-fine planar meshing approaches, such as sub-mapping [Owen 1998; Ruiz-Gironés and Sarrate 2010]. But in contrast to those it supports irregular quad connectivity, automatically introducing irregular interior vertices when warranted by the boundary shape (e.g. Figure 12). More significantly it operates on 3D curves, without the benefit of a well defined planar domain. While planar meshing methods focus on element quality or shape, our goal is to recover and quadrangulate a surface enclosed by designer-drawn curves.

Many recent publications addresses quad meshing of existing 3D surfaces [Bommes et al. 2011; Daniels et al. 2009; Kälberer et al. 2007; Levy and Liu 2010; Tong et al. 2006; Bommes et al. 2010; Marinov and Kobbelt 2006]. These methods aim to align the output quad meshes with the principal curvature directions in anisotropic regions generating smooth orthogonal families of flow-lines. In our setup no underlying surface is available. Instead we aim to align the output meshes with the flow-line directions conveyed by the input designer curves, which as noted above strongly correlate to curvature lines.

**Surface Fitting to Curve Cycles:** There is a large body of work on interpolating closed curves with smooth surfaces, much of it in the context of hole filling [Malraison 2000]. While a small portion of the methods [Levy 2003; Das et al. 2005; Finch and Hoppe 2011; Nealen et al. 2007; Rose 2007; Abbasinejad et al. 2011] can operate on arbitrarily shaped curves, the majority assume that the cycle is pre-segmented into $n$ sub-curves and has a low-distortion mapping to a convex planar $n$-sided polygon, e.g. [Coons 1964; Gao and Rockwood 2005; Várady et al. 2011].

The former, more generic, approaches typically utilize a diffusion process that optimizes surface fairness. They do not explicitly align flow-lines or curvature directions with the input curves, failing to capture designer intent on structured inputs (Figure 4 (b,c)). Using pre-defined normals along the input curves [Levy 2003; Mehra et al. 2009] improves the output, but as the normals are diffused uniformly, dominant flow-line directions may be lost. Moreover such normals are not part of a typical curve-based modeler output [Bae et al. 2008; Nealen et al. 2007; Schmidt et al. 2009]. An alternative approach of Rose et al [2007] fits developable surfaces to the input cycles. This approach is too restrictive for a general modeling setup, where many inputs, including the cycle in Figure 4 (a), aim to convey non-developable surfaces.

**Fitting to $n$-sided curve cycles** A variety of popular techniques are available for interpolating and approximating networks of regular quad or triangular patches [Farin 1992], see [Orbay and Kara 2011] for a recent sketching motivated approach. These methods, including the well-known Coons patches [Coons 1964], and their discrete extension [Farin and Hansford 1999] (Figure 4, (d)) provide an effective solution which naturally aligns the surface iso-lines with the flow-line sequences indicated by the boundary curves. Design and perception literature indicate that designers expect the curve cycle boundaries to correspond to representative flow-lines implying surface curvature directions, a behavior captured by Coons interpolation (Figure 4, (d)), but not the other fitting approaches. These methods are widely used by modelers and designers as the resulting surfaces closely reflect designer intent.

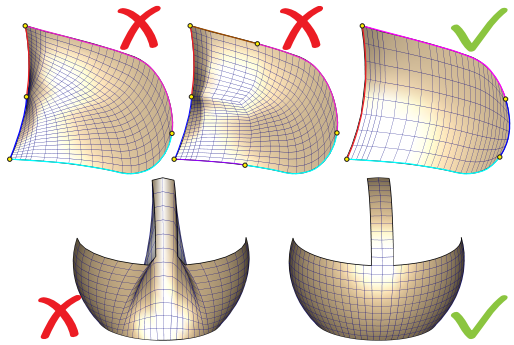


**Figure 5:** *(top) Using a purely topological approach and applying mid-point subdivision (forming either four or six sides) generates a quad mesh with poor flow line layout (left and center). Our method (right) uses geometry driven segmentation and matching to generate smooth flow lines and a predictable surface. (bottom) On a concave cycle, parameterization onto a convex domain (a rectangle) leads to foldovers (left), our method automatically segments the cycle into convex quadrilaterals leading to a fair surface (right).*

For cycles with $n > 4$ existing approaches can be classified into single surface fitting, e.g. [Gao and Rockwood 2005; Várady et al. 2011], or subdivision into quad or triangular cycles, e.g. [Schaefer et al. 2004; Nasri et al. 2009]. The first category of methods interpolate the cycles with a single surface patch by utilizing suitable $n$-sided convex 2D polygons as parameter domains. As acknowledged by Varady [2011] the fitted surface quality is strongly dependent on the quality of the 2D parameterization.

Subdivision approaches, e.g. [Schaefer et al. 2004; Nasri et al. 2009], quadrangulate the input cycles, and then use available techniques to interpolate or approximate the resulting quad network. In the basic midpoint scheme a single vertex is placed in the center of a patch and then connected to the middle of each side. To generate a watertight surface across heterogeneous networks, Schaefer et al. [2004] and Nasri et al [2009] introduce more sophisticated quadrangulation schemes that maintain a fixed number of intervals, or sub-segments along each side while aiming to control both the number and valence of the added extraordinary vertices [Nasri et al. 2009].

All methods in this category require the $n$-sides to be pre-defined and use either network junctions or sharp corners on the cycle as side end-points. As shown by figure 5 (top) using the actual shape of the curves to determine the end-point locations and induced topology as done by our method can significantly improve both the flow line layout and the resulting surface shape. More significantly, contrary to all the approaches above our method can operate on curve cycles with large concavities (Figures 5 (bottom) and 1 (b)).

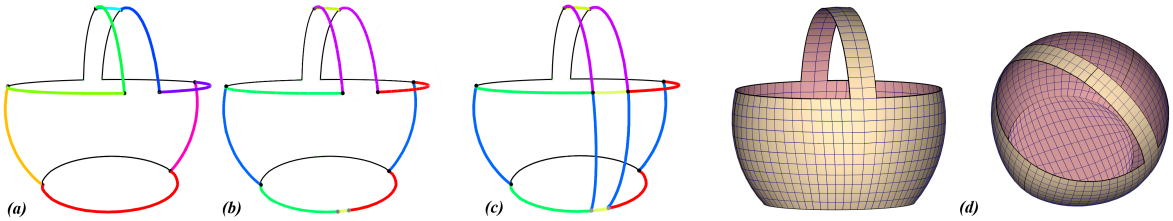
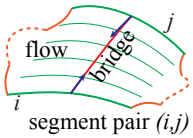



**Figure 6:** *After the initial segmentation (a), we alternate matching and refinement steps to obtain a pair-based curve segmentation which is converted into a quadrilateral network (c) . To minimize T-junction count (d) we compute global interval assignment, and use it to sample iso-lines on discrete Coons patches.*

## 3 Quadrangulating a closed 3D curve

This section describes our approach for quadrangulating the interior of a closed curve such that the iso-lines induced by the 4-sided curve cycles capture designer intended flow-lines. The extension of this method to networks of curves is discussed in Section 4. We use a dual based quadrangulation approach, where we first compute the dual graph of the quadrangulation (Section 3.1), and then use it to induce the primal quad connectivity and geometry (Section 3.2). This workflow is illustrated in Figure 6.



segment pair *(i,j)*

To assemble the dual, we segment the input curve into a small number of matching segment pairs that serve as opposite ends of dual graph poly-chords and corresponding primal quad-chains. In this respect, paired segments are analogous to river banks that both bound and define the flow between them; the poly-chord represents a bridge across the flow, connecting the paired segments.

Simultaneously computing this segmentation and pairing is an ambitious problem; we want to explicitly minimize the average matching cost, while avoiding outlier matches with very high cost. We consider the average, rather than the sum, so that the cost is not affected by the number of segments. To render this problem tractable, we use a discrete iterative optimization strategy that interleaves matching and segmentation. Given an existing segmentation and an appropriate cost metric, the right pairing strategy is not simply one that minimizes an overall cost, but instead one that prioritizes strongly compatible segment pairs that define dominant flow-lines. As noted in the Introduction, this can be mathematically formulated using the concept of a *stable matching*; we can find such a stable matching using the method of Irving [1985].

Once we have obtained such a pairing, we can then refine our segmentation by looking for a subdivision that maximally decreases our average matching cost without increasing the worst match cost (Section 3.1.5). To find the optimal splitting point(s), we examine the pairings in the current stable matching and consider strategies that improve the current high-cost matches. This new segmentation can then be fed back into the matching stage. To generate the desired segmentation and pairing , we start from an initial segmentation and interleave segmentation and matching steps. Since we aim for a compact quadrangulation, we use a coarse to fine segmentation update strategy, starting with the minimal segmentation for which the notion of opposite segments, or bridging directions, is well defined. To avoid over-segmentation we stop the refinement process once the improvement to the average match cost becomes insignificant.

The final segmentation induces a poly-chord graph, which we use to generate quad network connectivity. The generated interior curves are positioned using an extension of the quadrangulation scheme of Nasri et al. [2009] (Figure 6 (c)). A mesh of the entire network is then computed as discussed in Sections 3.2 and 4 (Figure 6,(d,e)).

### 3.1 Segmentation and Matching

The pseudocode below describes the flow of our iterative segmentation and matching process. Every iteration, we subdivide one or more segments to maximally reduce the average matching cost, without increasing the worst-match cost (Section 3.1.4). While the number of curve segments, at intermediate steps of the algorithm may be odd, each iterative refinement increments the number of segments, typically by one, admitting a perfect segment matching after one or two iterations. We continue to iterate until there is no significant drop in the average matching cost, rolling back to the last even segmentation when significant improvement is no longer possible (Figure 7). While this algorithm does not guarantee a globally minimal average match cost, it captures our design goals admirably in that it finds and preserves dominant segment pairs early and then refines segments as necessary to reduce the matching cost of poorly paired segments.

**Notation:** The above steps are described succinctly using notation and pseudo-code as follows: Given a curve segmentation $\sigma = 1, .., n$, we refer to $(i, j)$ as a distinct segment pair with a matching cost $c_{i,j}$ ($(i, j)$ and $c_{i,j}$ are symmetric). $c_{i,j}$ captures the compatibility of any two curve segments to form opposite sides of dual poly-chord in our target quadrangulation. $M(\sigma)$ is a perfect matching of $\sigma$, where each segment is uniquely paired, barring a solitary unmatched segment when the number of segments $||\sigma||$ is odd. We define the average cost of a matching $M(\sigma)$ as $cost(M, \sigma) = (\sum_{(i,j)\in M(\sigma)} c_{ij}^2)/(2 \cdot \lfloor||\sigma/2||\rfloor)$. A constant $drop = 1.25$ captures the factor of average cost reduction below which the iterative algorithm terminates.

```
1  σ= initial segmentation (Sec. 3.1.1);
2  M(σ)= stable matching of segment pairs (i, j) using match cost c_{i,j} (Sec. 3.1.3);
3  U_b^* = ∞;
4  cost^* = ∞;
5  Repeat
6        if ||σ|| is even:
7          then σ^* = σ; M^* = M; cost^* = cost(M, σ)
8               U_b^* = max_M c_{i,j};
9        σ'=refine σ (Sec. 3.1.4);
10       M'(σ')= stable matching of σ';
11       σ = σ';
12 Until (||σ'|| is even) and
13       (drop * cost(M', σ') > cost^* or U_b^* < max_{M'} c_{i,j});
14 create internal quadrangulation curves from poly-chord graph of M^*(σ^*);
```

We now elaborate on the rationale and details of each step.

#### 3.1.1 Initial Segmentation

As described in the Introduction we expect the flow-lines induced between any pair of segments to be smooth. Motivated by this con-

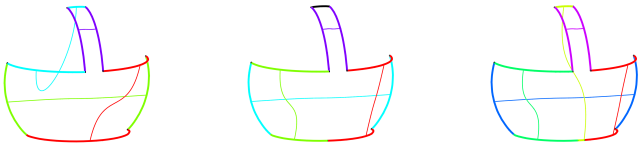
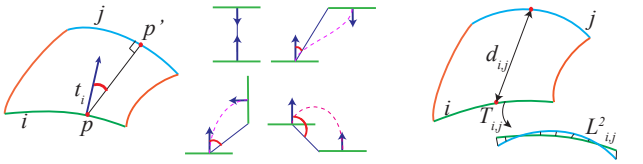


**Figure 7:** *Iterative segmentation refinement: (a) initial segmentation where the matching highlights correct dominant side matches. The match quality is drastically improved by segmenting the bottom curve (b), and repeating the process (c) to obtain an even segment count. Further refinement has no real impact on matching cost.*




(a) bridge curvature $b_{ij}$ using bridge directions $t_i$ (b) distance $d_{ij}$, similarity $s_{ij}$

**Figure 8:** *Estimated bridge curvature for different segment layouts measured as angle (red) between bridge direction $t_i$ and $p - p'$ (at a point p). The dashed lines visualize representative intersecting flow-lines (a). Shape similarity and distance cost terms (b).*

tinuity property of flow-lines, we can use any robust corner finding technique, such as computing discontinuities of discrete curvature along the curve [McCrae and Singh 2009], for our initial segmentation. We further refine this segmentation to ensure that the line segments connecting curve end-points are near linear using a technique similar to [McCrae et al. 2011]. This property helps define coherent bridge directions for matching cost evaluation, described next.

#### 3.1.2 Segment Pairing Cost

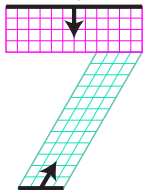
Paired segments have a two-fold impact on the final flow-line network. They explicitly define the sequence of flow-lines evolving from one segment to its mate. They also impact the family of flow-lines intersecting this sequence. Since the pairing defines a chain of quadrilaterals in the final quad network, these intersecting flow lines connect the two segments by evolving from one pair of end-points to another (see Figure 8 (a)). To generate the designer-expected flow-line network, the matching cost must satisfy the following criteria. First, to minimize the variation of flow-lines that evolve from one segment to the next we aim for the segments to be *similar*. Matching impacts the shape of the intersecting family of curves, or *bridge*, which in general we want to be as straight as possible, minimizing its *curvature*. Internal flow-lines should reflect input curve geometry, thus we would like the bridge to be aligned with intersecting flow lines evolving from input curve chains connecting the two segments, or, since these chains can be very complex, to at least *align* with intersecting sequences evolving from neighboring segments. Lastly, to best capture the general correlation between flow-lines and lines of curvature of the imagined surface, we expect intersecting sequences of flow lines to be *orthogonal*. We capture the last two requirements through a per-segment preferred *bridge direction*, which depends on the segment and its two neighbors. We use these directions to define bridge curvature $b_{i,j}$. Our matching cost combines bridge curvature, a term measuring similarity between the segments $s_{i,j}$, and a weak distance term $d_{i,j}$ used to prioritize more close-by matches

$$c_{i,j} = w_b b_{i,j} + w_s s_{i,j} + w_d d_{i,j}.$$

As the segments typically have fairly similar shape, bridge curvature dominates the cost with $w_b = 0.8$ and $w_s = w_d = 0.1$.

**Bridge Curvature:** To estimate the curvature of the anticipated intersecting flow lines, or bridge, between segments $i$ and $j$, we use the predicted bridge directions $t_i$ and $t_j$ for both ends of the bridge. As illustrated in Figure 8, the flow-line shape depends both on these directions and the relative location of the segments. As start and end positions, plus directions, allow for fitting of multiple flow-line curves, explicitly evaluating flow-line curvature is problematic. Instead we use an angle based curvature predictor defined as follows. Let $p$ be a point on the segment $i$, and let $p'$ be the point where the angle between the vectors $t_i$ and $p - p'$ is minimal on the segment $j$, i.e. $p' = \text{argmin}_{x \in j} |\angle(t_i, x - p)|$. Then, for a given point, the angle $\angle(t_i, \overline{p' - p})$ measures the angular difference between the shortest bridge between the segments and the one taken when using the estimated bridge direction $t_i$. To compute deviation across the segment $i$, $a_{i \to j}$, we average the angular difference over all points. Finally, we set the bridge curvature to the maximum of the per-segment deviations, namely $b_{i,j} = \max(a_{i \to j}, a_{j \to i})$.

**Bridge directions:** The bridge direction $t_i$ is the predicted optimal tangent direction for the flow-lines intersecting the segment $i$. As such, it depends both on the segment orientation, and on the bridge directions at neighboring segments.



Designer flow-line properties as seen in the inset figure attempt to align a bridge direction perpendicular to its segment's orientation (magenta), as well as parallel or orthogonal to the bridge directions of its neighboring segments (cyan). As the segments and their pairings evolve, so do their bridge directions.

The initial bridge direction $t_i$, for any segment $i$, is estimated from the initial segmentation (Figure 9(a-c)) and then refined in every subsequent algorithmic iteration (Figure 9(d)). The initial bridge direction $t_i = n_i$, is set to capture a direction orthogonal to the segment and lying on the imaginary surface emanating from it. Specifically, we define $n_i$ as the perpendicular to the straight line $f_i$ connecting its end-points, in the best-fit plane of the segments $i$ and its neighbors. Neighboring segments can also strongly influence bridge direction. An adjacent segment $m$ is considered to influence the bridge direction of $i$ if it is of reasonable arc-length $l$ ($1.5 * l_m > l_i$), and if its general flow direction $f_m$ is likely to form flow-lines intersecting those emanating from $i$ ($\angle(f_m, f_i) \leq 135^\circ$). The bridge direction $t_i$ is refined to be the average $f$ of its influential neighbours (Figure 9(a)(b)), or left as $n_i$ if none exist (Figure 9(c)).

Then, at every algorithmic iteration, we update bridge directions (Figure 9(d)), using *dominant pairs*, i.e. pairs $(i, j)$ such that $c_{ij} < dom$, where $dom = 0.15$. First, we refine the bridge direction of the dominant pairs. We update $t_i$ and $t_j$ of all dominant pairs $(i, j)$ to their current average (thus implicitly lowering their bridge curvature estimate $b_{i,j}$). Next, for any segment $i$ that is not dominantly matched but has a neighbor $m$ that is part of a dominant pair, we use $t_m$ to update $t_i$. Specifically, we attempt to set $t_i$ to either align, or to be orthogonal to, $t_m$. if the angle between $t_i$ and $t_m$ is less than $135^\circ$, we set $t_i$ to be orthogonal to $t_m$ in the plane defined by $n_i$. If $135^\circ \leq \angle(t_i, t_m) \leq 225^\circ$, we set $t_i = t_m$. If $t_i$ has two dominant neighbors, we use the one with lower matching cost for the update. The remaining bridge directions are left unchanged in this iteration.

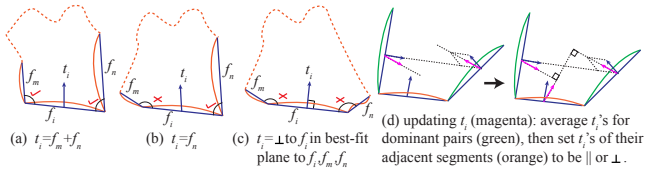



**Figure 9:** *Initial bridge direction $t_i$ of segment $i$ is determined by adjacent segment flow directions $f_m$ , $f_n$ and its normal.*

**Distance and Similarity:** The distance $d_{i,j}$ is simply the Euclidean distance between the segment centers (Figure 8b). Given two curve segments $i$, $j$, we measure their similarity in terms of shape and scale. We measure scale as the difference in curve length $\|l_i - l_j\|$. To compare shape, we first compute a best-fit affine transform $T_{i,j}$ from $i$ to $j$. We do this by resampling the curves by arc-length using the same number of points, 50 for all our experiments. We then use a linear least squares formulation to find the affine transformation which minimizes the $L_2$ distance between the two point-sets. We use a generic affine transform instead of a rigid one to allow for non-uniform scale and shear. We then measure similarity as the $L_2$ closest-point distance between the transformed curve and its mate $L_{i,j}$. All distances are normalized by the *diameter* of the processed curve, i.e. by the maximal distance between two points on the curve. Similarity between curves is then set to $s_{i,j} = 0.5\|l_i - l_j\| + 0.5(1 - e^{-L_{i,j}^2/\sigma^2})$. The second term measures the affine invariant shape difference of two curve segments. Specifically, we define a function that is zero if the curves are identical and 1 if they are maximally different. We achieve this mapping using a Gaussian fall-off function applied to the $L_2$ distance between the curves segments. Normalizing this distance by the diameter of the curve loop and setting $\sigma = 1/3$, set using the three-sigma rule, results in the desired shape difference function.

#### 3.1.3 Stable matching of segment pairs

Given a curve segmentation and a cost of pairing any two curve segments to form opposite sides of a poly-chord, this step aims to match segment pairs in a manner that maximally satisfies the dominant pairing preferences producing a stable matching.

The standard algorithm for computing a stable matching [Irving 1985] consists of two phases. First, each segment "proposes" to all other segments in order of pairing preference, continuing to the next segment if and when its current proposal is rejected. A segment rejects a proposal if it already holds, or subsequently receives, a proposal from a segment it prefers. In our setup, since matching costs are symmetric, if the number of segments is even this step ends with each segment holding a proposal from another segment. If the number of segments is odd, one segment is left out by the process and is ignored by the subsequent step.

Held proposals form a set $S$ of ordered segment pairs $(i, j)$, where $i$ holds a proposal from $j$ ($j$ is $i$'s current favorite). $S$ is a stable matching if $(j, i) \in S$ whenever $(i, j) \in S$. A second phase of repeated co-rotations, described below, transforms $S$ into a stable matching. Suppose that $(i, j) \in S$, but not $(j, i)$. For each such $i$ we identify the current second favorite to be the first successor of $j$ in $i$'s preference list who would reject their held proposal in favor of $i$. A rotation relative to $S$ is a sequence $(i_0, j_0), (i_1, j_1), ..., (i_{k-1}, j_{k-1})$ such that $(i_m, j_m) \in S$ for each $m$, and $j_{m+1}$ is $i_m$'s current second favorite (all indices are modulo $k$). A co-rotation replaces pairs $(i_m, j_m)$, with $(i_m, j_{m+1})$in $S$.

The standard method [Irving 1985] is proven to provide a stable match for an even number of participants, unless an *odd party* is found [Tan 1991], i.e. a rotation such that $k$ is odd, and $p_i = q_{i+(k+1)/2}$ for all $i$. In that case no stable matching exists. In the rare case of an odd party, we have an odd-length cycle of segments with equal pairwise costs, e.g. an equilateral triangle or three perfectly symmetric curves (Figure 12). This case can be seen as a generalization of the standard midpoint splitting, and is resolved by splitting each segment in the cycle into two. Once the split is performed, a clear difference in cost emerges and the matching is repeated.

#### 3.1.4 Segmentation Refinement

The refinement process looks for a segment, or segments, to subdivide so as to maximally decrease the average matching cost. Our refinement examines two segmentation strategies, first searching for a single edge refinement and then a global mid-edge split. Since the number of segments is typically very small, a stable matching computation is practically instantaneous. Using the first approach, we quickly iterate over all segments, segmenting each one and evaluating the cost of the match computed with the refined segmentation. We then select the segmentation that maximally lowers the cost. Using this strategy, the one question we need to address is where to place the split, as the location can impact the subsequent segmentation cost.

The basic strategy of splitting the segment in half is tested first, then a more targeted strategy that leverages the computed matching is applied to the currently matched segments. Given a current segment $i$ which is matched to $j$ we search for all segments $k$ that are either unmatched, or that prefer to be matched to $i$ rather than their current mate $l$, i.e. $c_{k,i} < c_{k,l}$. In such situations, for instance the bottom curve on the basket (Figure 7), splitting the curve strategically into $i_1$ and $i_2$ can often satisfy this preference by generating matches $(i_1, j)$ and $(i_2, k)$. To minimize the cost of $(i_1, j)$ and $(i_2, k)$ we break $i$ into two possible subdivisions $i_1, i_2$ based on arc-length ($l$) ratio, where $l_{i_1}/l_{i_2} = l_j/l_k$ or $l_{i_1}/l_{i_2} = l_k/l_j$, and $l_{i_1} + l_{i_2} = l_i$, and test the matches induced by these segmentations.

While theoretically more comprehensive or global segmentation refinement strategies may exist, we found our approach to work well in practice. It preserves dominant pairs and improves poor matches as intended by our subdivision heuristic.

### 3.2 Quadrangulation

Once we have an acceptable perfect stable match whose cost cannot be reduced by further segmentation, we use this segmentation and matching and its induced poly-chord graph (see Figure 6), to construct a quadrangulation.

**Extracting Quad Connectivity:** Using standard dual notations [Daniels et al. 2008] we say that two poly-chords $(i, j)$ and $(k, l)$ *intersect* in the graph theoretic sense if and only if their corresponding curve segments are interleaved on the closed curve. For instance, the purple and red segments on Fig. 10 are interleaved, resulting in intersecting poly-chords. To generate a valid quadrangulation we require that the poly-chord graph be connected . This is easily accomplished by adding curve segments connecting end-points of common segments of components of the poly-chord graph and turning each graph component into a smaller closed curve, for which our algorithm can be re-run (Figure 10). To avoid $T$-junctions we disallow the newly added segments from being further refined. To make the quad layout more compact, we merge adjacent poly-chord $(i, j)$ and $(i+1, j-1)$ when the transition between the consecutive segments is smooth.

An intersection between two poly-chords corresponds to a quadri-

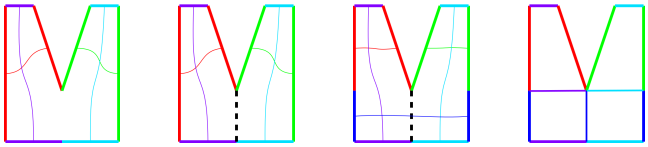


**Figure 10:** *A disconnected dual graph (left) does not allow for a valid primal quad mesh. Splitting the cycle into two by a temporary curve segment (dashed) generates valid graphs for both parts which combined together induce a valid primal quad mesh (right).*

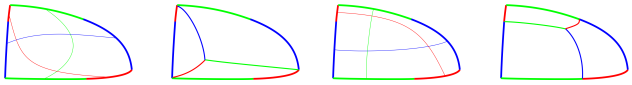
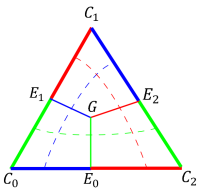


**Figure 11:** *Two intersection orders induce different quad connectivity, with the one on the right inducing a better quad shape, and consequently a smoother flow.*

lateral in the final network. Connectivity between these quads is determined by the intersection order, e.g. determining the top-down order of the intersections of the green poly-chord with the blue and red ones in Figure 11. We define the quad connectivity by incrementally embedding poly-chords into the layout of cells, or regions, bounded by input boundary segments and previously added poly-chords. Given the graph whose vertices are these cells and whose edges connect adjacent cells, we embed a poly-chord by computing the shortest path in this graph between the two vertices or cells, corresponding to the boundary curve segments connected by the poly-chord. This path minimizes the number of intersections between the new poly-chord and those already embedded. This choice minimizes the number of dual graph cycles. Such cycles correspond to interior primal quadrangulation vertices adding which, as discussed below, can reduce flow smoothness. Given two equal length choices, we prefer one that induces better shaped quadrilaterals, where quality is measured as the scaled Jacobian [Brewer et al. 2003] (Figure 11).

**Extracting quad geometry:** The dual graph defines the connectivity of our quadrangulation. To position the interior vertices and curves we use a two step process which leverages the quad topology to generate interior curves best reflecting the flow directions. Specifically we note that each chain of quads can be seen as a four-sided $uv$ patch interpolating two flow-line end segments. Associating the $v$ coordinate with the end segments, we expect the patch $u$-isolines to smoothly interpolate them. Our geometry computation builds on the geometry construction in [Nasri et al. 2009] which shares the same goal. We first compute the interior vertex positions that best satisfy our requirements, using a global optimization of a per-vertex formulation [Nasri et al. 2009], that sets each vertex $G$ to a weighted sum of vertex positions in neighboring quads:




$$G = \frac{\sum_{i=1}^{n}(E_i + E_{i-1} - C_i)/a_i}{\sum_{i=1}^{n} 1/a_i} \qquad (1)$$

where $E_i$ are quad network vertices that share side curves with $G$, $C_i$ are the diagonal quad corners between $E_{i+1}$ and $E_i$, and $a_i = \|E_i - C_i\|\|C_i - E_{i+1}\|$ is an estimate of the area of the corresponding quad (see inset). We then generate straight-line edges connecting these and boundary vertices as an intermediate approximation of the quadrangulation. Using this initial network each interior curve is now computed as a $u$-isoline on the quadrilateral patch containing two bounding flow-lines and the curve paths connecting

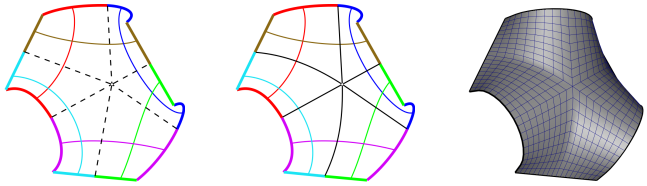


**Figure 12:** *We first position interior vertices (left) and then use the chain-long quads to position the interior curves (center). Finally, the resulting quad cycles are quad-meshed using discrete Coons patches (right).*

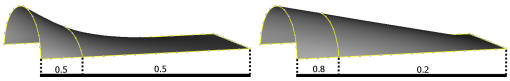



**Figure 13:** *Our distance based weighing (right) generates smoother flow line evolution than topology based one [Nasri et al. 2009].*

them, using a discrete Coons formulation [Farin 1992] (Figure 12). This formulation takes into account the distance of the new curve from the bounding flow lines, improving on the original formulation of Nasri et al [2009] (Figure 13).

**Meshing:** To fit a surface in the interior of each quad-patch we can use any number of methods. The examples shown in the paper use a quad mesh sampled on a discrete bicubic Coons surface [Salomon 2006]. This construction provides continuity across shared boundaries when the cross tangents are continuous. More sophisticated fitting tools which provide better cross-patch continuity can be used as well.

**Minimizing Flow Dislocation:** The segmentation and pairing algorithm optimizes the cost of the individual flow-line matches, does not explicitly consider the impact of the quad patch connectivity on the final flow. Specifically, at the matching stage it is hard to predict the impact of the introduction of interior patch vertices on the smoothness of the flow lines. In some cases these vertices are essential to forming a good surface such as on the top of the espresso machine (Figure 19), but in other cases removing them can improve the flow (Figure 14). Thus, given a quadrangulation, we test if removing any of the interior vertices can improve the surface quality. Recall that each such vertex corresponds to a cycle in the dual graph. We thus attempt to break cycles in the dual graph if the quadrangulation quality improves and the increase in the overall matching cost is acceptable. Specifically, for each edge $\langle (i,j),(k,l) \rangle$ of a cycle in the poly-chord graph we evaluate the consequence of swapping segment pairs to $(i,l)$ and $(k,j)$, or $(i,k)$ and $(l,j)$. A swap is valid if the following three criteria are satisfied: the quad quality, measured using the scaled Jacobian, is improved, no new cycles are

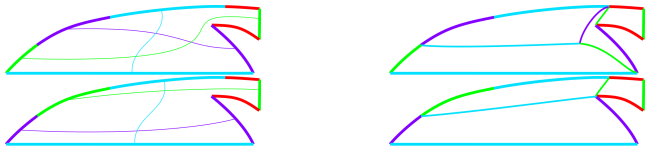


**Figure 14:** *Removing interior vertices: (Left) initial match (top) and induced quadrangulation (bottom); (Right) the final match with purple and green pairs flipped (top) has a slightly higher cost but the induced quadrangulation (bottom) has no interior vertices, leading to smoother flow-lines.*

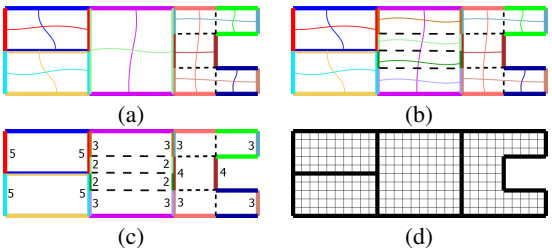



**Figure 15:** *Separately processed cycles (a) introduce T-junctions. We first resolve the T-junctions across pairs of neighbouring patches by propagation (b), generating a well defined hierarchy of matching primary segments. We then use integer programming to compute interval assignments (c) that minimizes the number of T-junctions, typically leading to a watertight mesh (d).*

introduced into the graph and the cost of the matches after the swap is no greater than the worst match cost before it. We thus perform a valid swap for the poly-chord edge of the cycle with the minimum increase in matching cost (Figure 14).

## 4 Processing Curve Networks

Up until now, we have only considered the meshing of a single curve cycle. The reason for this is that in curve networks, the majority of vertices adjacent to two or more cycles define corners that induce our initial segmentation. The remaining vertices form T-junctions that should not bias the flow-lines within cycles where the incident curves are continuous. Once the individual cycles have been quadrangulated however, we must ensure that the geometry is watertight across the common boundary of adjacent cycles. For a quad-mesh fitting this requires the sampling, or interval count, along shared boundaries to be the same on both sides. This goal is easy to achieve for a conforming quad-patch layout, such as those generated inside each input cycle, using a fixed number of intervals per boundary curve. Special care is needed though, when meshing curve networks where cycle segmentation creates T-junctions.

We optimize interval assignment using two modifications to the basic cycle quadrangulation algorithm described above. The first stage, performed after segmentation and matching process, described above, for each cycle, resolves the initial, *primary*, T-junctions between pairs of neighbouring cycles. A T-junction occurs when one curve has a segment end-point, or vertex, at a boundary point and another curve does not. Given a T-junction, we first attempt to resolve it by merging adjacent vertices based on a threshold distance, while keeping in place both sharp corners and T-junctions present in the original artist input. Throughout our experiments, we set our threshold to $5\delta$, where $\delta$ is the minimum Euclidean distance between adjacent samples of the input polylines. Intuitively, the finer the initial sampling, the more precise the algorithm is, the smaller the merging threshold we need.

For any T-junctions that we cannot resolve in this manner, we split the adjacent segment and its matching segment in the corresponding cycle. We then refine the matching accordingly. This process resolves all the primary T-junctions, but in turn introduces *secondary* T-junctions where the matching segments are split (Figure 15, (b)). These T-junctions are further reduced using another iteration of threshold based merging.

Contrary to primary T-junctions, the secondary T-junctions are guaranteed to be contained in *primary* segments that share clearly defined *primary* vertices (Figure 15, (b)), a property we take advantage of in the final interval assignment stage. At this point, the

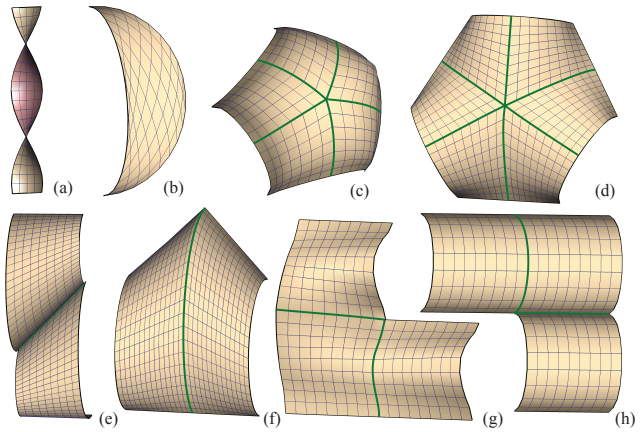



**Figure 16:** *Quadrangulation and meshing of closed curves.*

network is converted to quad-patch topology using the method of Section 3. In the final step, when generating the per-patch meshes, we need to assign a consistent interval count to each segment. For a given primary segment, we require that the number of intervals on both sides of the segment are equal. We further require that each secondary segment (one bounded by primary or secondary vertices) and its matching segment have the same number of intervals. Finally, we wish to minimize the total interval count while enforcing a minimum number of intervals per edge based on its length.

If we formulate all of these requirements as a wishlist, as shown by Mitchell [1997], there may exist configurations where no valid assignment by Mitchell [1997], there may exist configurations where no valid assignment exists. We therefore relax our watertightness requirement, which allows us to reformulate this problem in terms of a minimization. Consider a pair of adjacent primary segments $L$ and $R$. By virtue of the first step, we know that $L$ and $R$ share common endpoints; however, they may each contain a differing number of secondary segments. If $l$ is a secondary segment on $L$ and $r$ is a secondary segment on $R$, let $n_{l,R}$ and $n_{r,R}$ represent the number of intervals that the secondary segments $l$ and $r$ are divided into, respectively. We can then express our minimization condition as the following function:

$$\min\ f(x) = w \sum_{(L,R)} \Big(\sum_{l \in L} n_{l,L} - \sum_{r \in R} n_{r,R}\Big)^2 + \sum_{L,l} (n_{l,L})$$

The first term in this equation seeks to minimize the number of mismatched interval counts along a given pair of adjacent primary segments. The second term seeks to minimize the total number of intervals for the entire mesh. We use $w = 1000$ to minimize the number of mismatches as much as possible. This minimization is subject to a number of constraints. We require that opposite segments of each quad patch have the same number of intervals. Additionally, we require that the number of intervals on a given secondary segment does not fall below a specified minimum. This minimum is determined by dividing the secondary segment length by a user-specified desired (local or global) interval length. Together, the minimization function and constraints form a quadratic, mixed-integer programming problem, which we solve using Tomlab /CPLEX. This approach lead to valid assignments for all the inputs we tested. The assigned intervals are used to optimize the positions of the secondary T-junctions and generate the final meshes.

## 5 Results

**Closed Curves:** We generated a number of synthetic test inputs to evaluate the behavior of our method on a variety of closed curves

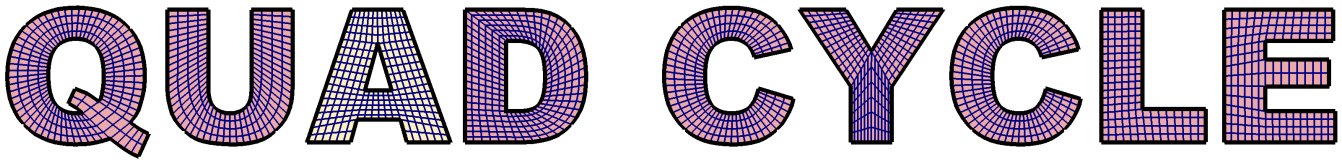



**Figure 17:** *Quad meshes of complex closed curves including interior cycles.*

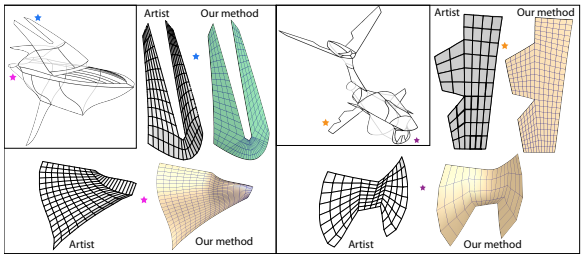



**Figure 18:** *Artist generated meshes (left) and ours (right) exhibit very similar flow-line patterns.*

| | input cycles | output quad cycles | mesh size | interior vertices |
|---|---|---|---|---|
| Sphere Bag | 3 | 9 | 987 | 0 |
| Boat | 30 | 82 | 4464 | 9 |
| Spaceship | 41 | 94 | 5008 | 6 |
| Car | 26 | 70 | 5020 | 13 |
| Espresso | 54 | 75 | 6904 | 5 |
| Speaker | 13 | 42 | 8548 | 1 |
| Plane | 140 | 192 | 10705 | 10 |
| Submarine | 39 | 103 | 16600 | 31 |

**Table 1:** *Algorithm statistics for different curve networks.*

with different side configurations demonstrated in Figures 16 and 17. These included a variety of convex regions (Figure 16 (a-f)) with different degrees of planarity and different number of boundary discontinuities. For some of the inputs the expected surface shape, is best captured by introducing an extraordinary interior vertex (c,d). For other regions with $n > 4$ sides such as (e,f) a regular connectivity better captures the intended shape. Our method makes the appropriate choice based on analyzing the relationships between the input curve segments,and the degree of parallelism between them. This is in contrast to purely connectivity methods, e.g. [Nasri et al. 2009], where the choice is strictly based on the number of segments. Figure 16 (b) shows an a-typical two sided region, nevertheless reasonably fittted by our method, while (g,h) show non-convex regions, where the optimal pairing is found automatically through refinement of initial segments. The letters in Figure 17 show the robustness of the method in the presence of complex non-convex curves as well as processing of faces with interior loops. To handle such models, we first locate a pair of matching segments on different loops with minimal matching cost and introduce the shortest straight segment connecting those. The method then proceeds as usual on the resulting single cycle.

**Curve Networks:** We tested our method on a variety of input curve networks (Figure 1, 6, 19) generated by different modeling systems [Bae et al. 2008; Schmidt et al. 2009; Rose 2007]. As demonstrated by the figures these networks contain a variety of complex, non-convex cycles. Our method successfully captures the designer intent conveyed by the networks generating predictable and smoothly flowing quad-meshes interpolating the input curves. While the airplane (Figure 19) was created using a classical CAD modeling system, many of the other inputs (car, espresso maker, submarine, starcraft) (Figure 19) were generated using sketching tools, which easily introduce noise and inaccuracies that hamper traditional surfacing. Our method is robust to such artifacts.

We compare our outputs on the boat and starcraft to those generated by an artist (Figures 3 18). The flow-line structure of our meshes is largely identical to artist generated one, with only minor differences, such as flow on the side of the boat cabin, where both our and artist interpretations are feasible (our outputs contain a few extra cycles not present in the artist models).

**Quantitative Evaluation:** On an Intel i7 CPU 870 2.93GH machine our method takes on average two seconds to quadrangulate a single curve cycle (most of the time spent on matching), making it amenable for interactive surfacing in a sketch based modeler like ILoveSketch [Bae et al. 2008]. The most time consuming regions are the front of the car (166s) and the top of the speaker (66s) (Figure 19). Intervals assignment is practically instantaneous, taking 0.1s for a an average network and taking 2s to process the largest model (plane). The quad statistics for the models we tested are summarized in Table 1 and include numbers of input cycles, number of output quad cycles, mesh size(s), and the number of added extraordinary vertices, All the generated meshes are watertight.

**Limitations:** Our approach has three broad limitations which can be addressed by future research.

*Global context:* The biggest limitation of our method is lack of global context. Our flow-line analysis for each input cycle in a network is independent. In practice however, most adjacent cycles meet at sharp corners, typically resulting in a similar segmentation and flow across shared curve segments. The context of adjacent cycles could be useful in enforcing flow line continuity across cycles and predicting the flow within an individually ambiguous cycle.

*Failure cases:* While our algorithm works well on design curve inputs from a variety of sources, it may not provide meaningful results for arbitrarily shaped curve cycles with no perceptible flow-lines. The absence of corners on a completely smooth curve cycle will not provide us a meaningful initial segmentation to refine. In such cases we can impose an initial segmentation based on curvature maxima and arc-length of the input curve.

*Algorithmic complexity:* While our central idea of flow-line segmentation and matching is conceptually clear, various aspects of our implementation could be streamlined. For example, while most of the parameters used by the method were derived based on clear algorithmic goals, a few such as *drop* in Section 3.1) are based on trial-and-error, and could be learned from designer quadrangulated examples.

## 6 Conclusions

We presented the first, to our knowledge, method for quadrangulating general designer specified closed 3D curves and curve net-

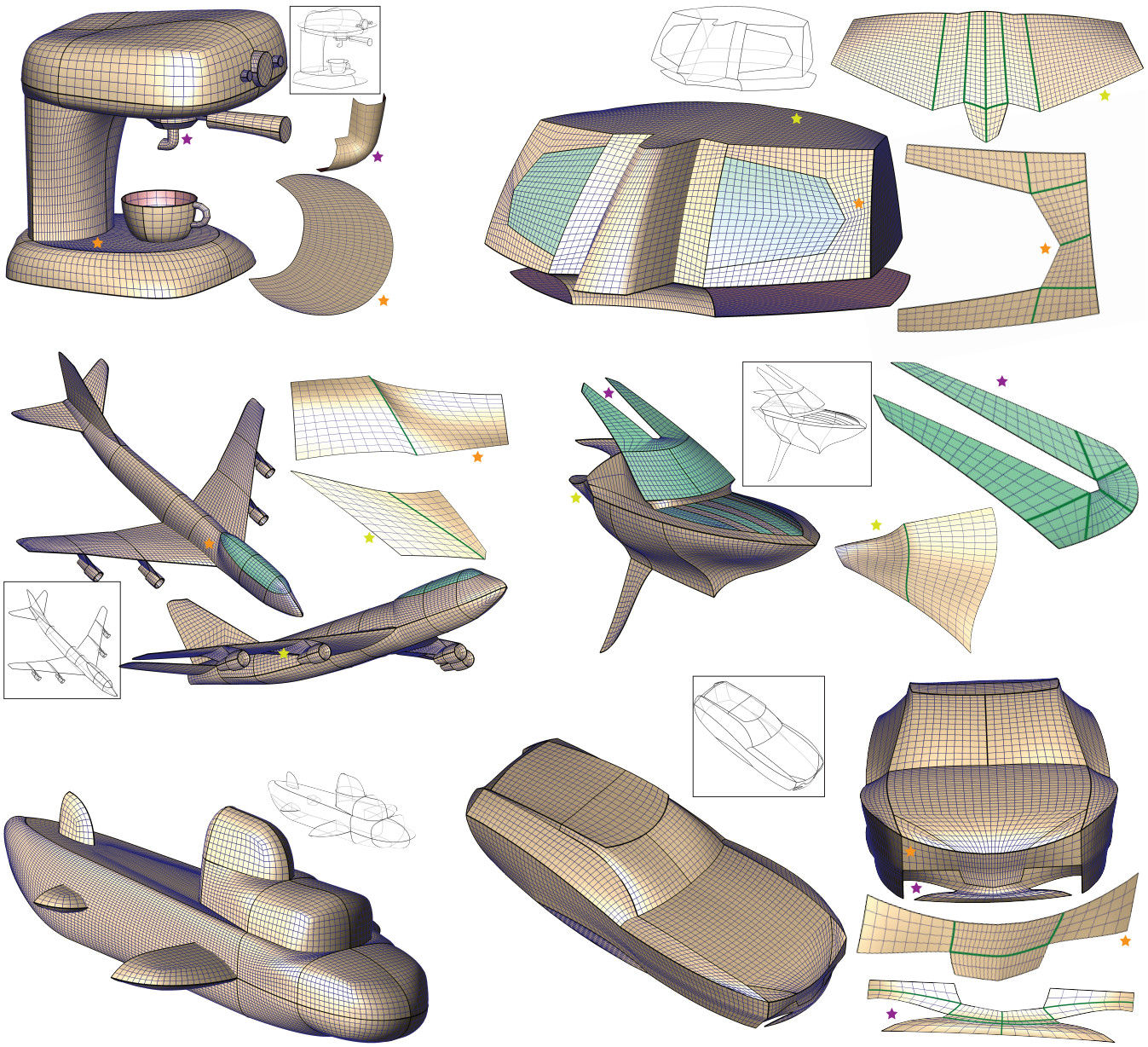


**Figure 19:** *Quadrangulation and meshing of curve networks. The stars indicate the network locations of the highlighted complex regions.*

works. Our results show the approach to robustly process complex curve networks, generating interpolating quad meshes consistent with designer intent. Our key insight is an interleaved segmentation and matching algorithm, that pairs dominant flow-lines and uses poor matches to guide segmentation refinement, computing a polychord graph that captures user-intended bridging directions across a closed curve. We advocate the use of stable matching as the principled way to formulate our quadrangulation goals and anticipate it to be well-suited to other problems relating to shape matching or coherence, where both dominant components and their correspondence is sought.

Our work points to a number of future directions. Rather than restrict our input to a constrained geometric definition of a design curve network, we attempted to quadrangulate any 3D curve network as a designer would, using the principle of flow-line segmentation and matching. A formal perceptual study of the precise difference between ambiguous and design curves (Figure 2) is thus an ambitious but worthy goal. While our segmentation refinement strategy works well in general, approaches with theoretical guarantees of match quality are also worth exploring. Our method focuses on quad-only meshing, however in some cases designer intent is better served by allowing a small number of triangular elements (e.g. Figure 16 (b)), motivating a technique for mixed but predominantly-quad meshes. We would also like to apply our technique as-is to the finite-element meshing of closed planar domains, balancing flow-line alignment against mesh quality.

## References

ABBASINEJAD, F., JOSHI, P., AND AMENTA, N. 2011. Surface patches from unorganized space curves. *Comput. Graph. Forum 30*, 5, 1379–1387.

BAE, S., BALAKRISHNAN, R., AND SINGH, K. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proc. Symposium on User interface software and technology*, ACM, 151–160.

BOMMES, D., VOSSEMER, T., AND KOBBELT, L. 2010. Quadrangular parameterization for reverse engineering. In *Proceedings of the 7th international conference on Mathematical Methods for Curves and Surfaces*, Springer-Verlag, Berlin, Heidelberg, MMCS'08, 55–69.

BOMMES, D., LEMPFER, T., AND KOBBELT, L. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum 30*, 2, 375–384.

BORDEGONI, M., AND RIZZI, C. 2011. *Innovation in Product Design: From CAD to Virtual Prototyping*. Springer.

BREWER, M., DIACHIN, L. F., KNUPP, P., LEURENT, T., AND MELANDER, D. 2003. The mesquite mesh quality improvement toolkit. In *Proceedings, 12th International Meshing Roundtable*, 239–250.

COONS, S. 1964. *Surfaces for computer aided design*. Technical Report, MIT.

DANIELS, J., SILVA, C. T., SHEPHERD, J., AND COHEN, E. 2008. Quadrilateral mesh simplification. *ACM Transactions on Graphics 27*, 5, 1.

DANIELS, J., SILVA, C. T., AND COHEN, E. 2009. Semi-regular quadrilateral-only remeshing from simplified base domains. In *Proc. Symposium on Geometry Processing*, 1427–1435.

DAS, K., DIAZ-GUTIERREZ, P., AND GOPI, M. 2005. Sketching free-form surfaces using network of curves. *Sketch-based interfaces and modeling SBIM*.

DE GOES, F., GOLDENSTEIN, S., DESBRUN, M., AND VELHO, L. 2011. Exoskeleton: Curve network abstraction for 3d shapes. *Computer and Graphics 35*, 1, 112–121.

FARIN, G., AND HANSFORD, D. 1999. Discrete Coons patches. *Computer Aided Geometric Design 16*, 691–700.

FARIN, G. 1992. *Curves and surfaces for computer aided geometric design: a practical guide*. Academic Press.

FINCH, M., AND HOPPE, H. 2011. Freeform Vector Graphics with Controlled Thin-Plate Splines. *ACM Trans. on Graphics (SIGGRAPH Asia) 30*, 6.

GAHAN, A. 2010. *3D Automotive Modeling: An Insider's Guide to 3D Car Modeling and Design for Games and Film*. Elsevier Science.

GAO, K., AND ROCKWOOD, A. 2005. Multi-sided attribute based modeling. *Mathematics of Surfaces XI*, 219–232.

IRVING, R. W. 1985. An efficient algorithm for the "stable roommates" problem. *J. Algorithms 6*, 4, 577–595.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3, 375–384.

LEVY, B., AND LIU, Y. 2010. Lp centroidal voronoi tesselation and its applications. *ACM Trans. Graph.*.

LEVY, B. 2003. Dual domain extrapolation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3, 364–369.

MALRAISON, P. 2000. *N-SIDED Surfaces: a Survey*. Defense Technical Information Center.

MARINOV, M., AND KOBBELT, L. 2006. A Robust Two-Step Procedure for Quad-Dominant Remeshing. *Computer Graphics Forum 25*, 3 (Sept.), 537–546.

MCCRAE, J., AND SINGH, K. 2009. Sketching piecewise clothoid curves. *Comput. Graph. 33* (August), 452–461.

MCCRAE, J., SINGH, K., AND MITRA, N. J. 2011. Slices: a shape-proxy based on planar sections. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, 168:1–168:12.

MEHRA, R., ZHOU, Q., LONG, J., SHEFFER, A., GOOCH, A., AND MITRA, N. J. 2009. Abstraction of man-made shapes. *TOG (Proc. SIGGRAPH Asia) 28*, 5, 1–10.

MITCHELL, S. A. 1997. High fidelity interval assignment. In *Proceedings, 6th International Meshing Roundtable*, 33–44.

NASRI, A., SABIN, M., AND YASSEEN, Z. 2009. Filling N -Sided Regions by Quad Meshes for Subdivision Surfaces. *Computer Graphics Forum 28*, 6 (Sept.), 1644–1658.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26* (July).

ORBAY, G., AND KARA, L. B. 2011. Sketch-based modeling of smooth surfaces using adaptive curve networks. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, 71–78.

OWEN, S. 1998. A survey of unstructured mesh generation technology. In *Proc. International Meshing Roundtable*.

ROSE, K. 2007. Modeling developable surfaces from arbitrary boundary curves. *Processing*, August.

RUIZ-GIRONÉS, E., AND SARRATE, J. 2010. Generation of structured meshes in multiply connected surfaces using submapping. *Adv. Eng. Softw. 41* (February), 379–387.

SALOMON, D. 2006. *Curves and Surfaces for Computer Graphics*. Springer-Verlag.

SCHAEFER, S., WARREN, J., AND ZORIN, D. 2004. Lofting curve networks using subdivision surfaces. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP '04*, 103.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Trans. on Graph. (Proc. SIGGRAPH Asia) 28*, 5.

SINGH, K., PEDERSEN, H., AND KRISHNAMURTHY, V. 2004. Feature based retargeting of parameterized geometry. In *Proceedings of the Geometric Modeling and Processing 2004*, IEEE Computer Society, Washington, DC, USA, GMP '04, 163–.

STEVENS, K. A. 1981. The visual interpretation of surface contours. *Artificial Intelligence 17*.

TAN, J. J. M. 1991. A necessary and sufficient condition for the existence of a complete stable matching. *J. Algorithms 12*, 1 (Jan.), 154–178.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '06, 201–210.

VÁRADY, T., ROCKWOOD, A., AND SALVI, P. 2011. Transfinite surface interpolation over irregular n-sided domains. *Computer-Aided Design*, iv.