Flow complex based shape reconstruction from 3D curves

BARDIA SADRI and KARAN SINGH
University of Toronto

We address the problem of shape reconstruction from a sparse unorganized collection of 3D curves, typically generated by increasingly popular 3D curve sketching applications. Experimentally, we observe that human understanding of shape from connected 3D curves is largely consistent, and informed by both topological connectivity and geometry of the curves. We thus employ the *flow complex*, a structure that captures aspects of input topology and geometry, in a novel algorithm to produce an intersection-free 3D triangulated shape that interpolates the input 3D curves. Our approach is able to triangulate highly non-planar and concave curve cycles, providing a robust 3D mesh and parametric embedding for challenging 3D curve input. Our evaluation is four-fold: we show our algorithm to match designer selected curve cycles for surfacing; we produce user acceptable shapes for a wide range of curve inputs; we show our approach to be predictable and robust to curve addition and deletion; we compare our results to prior art.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

Additional Key Words and Phrases: Sketch-based modeling, 3D Shape Reconstruction, Flow Complex

1. INTRODUCTION

Shape reconstruction is the general problem of creating an object from a sampling of a real or imagined target object, that closely resembles the target. Shape reconstruction from images, point-clouds or cross-sections is an active area of ongoing research in computer graphics, vision, and computational geometry [Dey 2011]. We consider a relatively new variant of shape reconstruction, from a collection of unorganized 3D curves (see Figures 1, 10). Curves are ubiquitous in art and design, and such 3D curves are the output of increasingly popular 3D sketching tools, such as ILoveSketch (*ILS*) [Bae et al. 2008], analytic drawing [Schmidt et al. 2009], or other 3D curve modeling interfaces [Grossman et al. 2003].

A current drawback of these interactive tools is that 3D sketches tend to visually clutter quickly and the intended object being sketched becomes difficult to understand perceptually, even for the designer. The ability to quickly and automatically infer a 3D model from the collection of curves, to resolve curve visibility for better viewing and further sketch exploration, is thus an important but missing affordance. Such a 3D model is also a basis for rapid prototyping and downstream mesh processing [Singh 2006]. Currently, the construction of a CAD surface model from 3D curve collections is a three-step, tedious and manual post-process (see Figure 1):

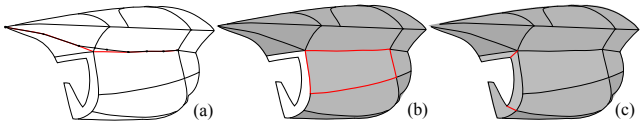


Fig. 1: Manual 3D surfacing: (a) curves (black to red) are edited to form a connected network; (b) curve loops are annotated (red) and surfaced; (c) complex loops are surfaced as simpler loops by adding curves (red).

—Curve intersections must be processed and additional curves created as necessary to create a connected 3D curve network;
—Closed loops of connected curves that define the boundary of intended surface elements must be annotated by the designer;
—Largely convex or planar loops can be surfaced using n-sided patches or variational meshing. Complex loops and intersecting surfaces must be manually simplified by adding input curves.

It is precisely this entire process that we seek to automate, enabling a tighter coupling between 3D curve design and surface modeling. We are inspired by the recent work of [Abbasinejad et al. 2011], which is an admirable first attempt at this ambitious problem, but fundamentally limited to inferring loops based on input curve connectivity and local geometry (see Figure 6).

Our goal of recovering a designer intended shape from a sparse collection of 3D curves deviates sharply from most shape reconstruction literature: point samples representing the shape are both sparse and anisotropically aligned along curves; the curves function as a wire armature or scaffold over which the surface is locally and often smoothly stretched; yet we must avoid general assumptions on the target surface being manifold or smooth.

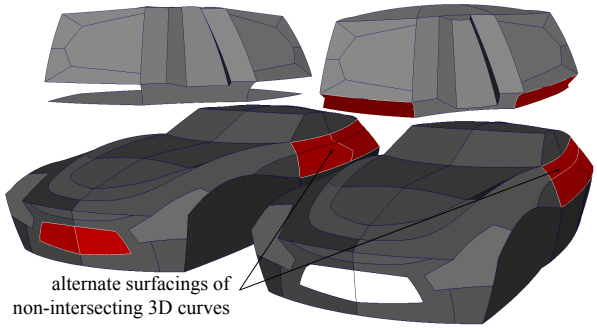


Fig. 2: ILS models surfaced by two designers with differences in red.

Bardia Sadri acknowledges the support of MITACS Elevate post-doctoral fellowship. He is currently affiliated with Side Effects Software Inc.

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

1.1 Design experiments:

To gain insight into the problem, we performed two semi-formal experiments. We first asked 3 designers to manually surface 9 *ILS* curve networks (see Figure 2, 5). We noted disagreement in only 14 of 415 patches surfaced. The contentious patches were all either perceived as semantically optional, such as an air vent, or were different surfacings of inconsistently crossing curves (see Figure 2).

We also asked 32 viewers to mark the visible surfaces they imagined for two line drawings, randomly chosen from the four in Figure 3 (lines closer to the viewer are drawn thicker). Note that the four drawings all have the same curve connectivity, and differ only in the geometric size and placement of the circular cross-sections. We noted complete agreement on the cylindrical surfaces and some variance on whether the circular surfaces were capped with a general preference ($\approx 65\%$) for the interpretations shown in Figure 3.

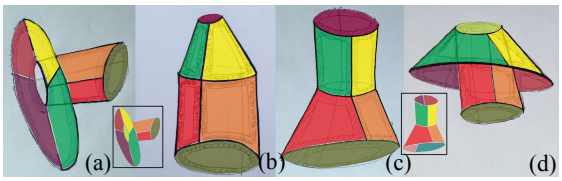


Fig. 3: Viewer-shaded visible surface patches, overlaid with color for clarity (dominant alternatives are inset). The cylindrical surfaces are consistently perceived, while the circular caps vary by geometric situation.

The strong agreement on the designer surfaces and perfect agreement on the cylindrical surfaces suggests consistency in the shape conveyed by design curve networks. All marked surfaces were intersection-free but there was no clear tendency to close cavities or keep surfaces manifold (Figure 3(a) is not closed, Figure 3(d) is not manifold). The most ambiguous shapes were Figure 3(a), (c) with their dominant alternatives shown inset, hinting at object recognition possibly affecting the choice of surfaces (hat/table, capsule/funnel). The differences in surface interpretation for curve networks with identical topology in Figure 3, suggest that approaches to surfacing 3D curves should exploit both global geometry and topology. The cycle basis approach of [Abbasinejad et al. 2011], for example produces the topology of Figure 4(a), on all four inputs of Figure 3. Our approach (Figure 4(b)-(e)) consistently surfaces the cylindrical patches, and chooses intersection-free circular loops based on an intuitive criterion described in Section 2.

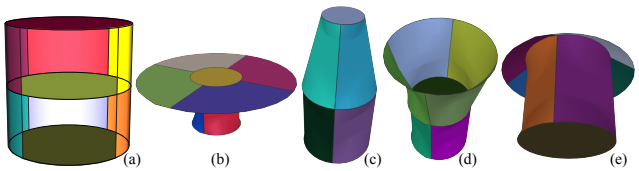


Fig. 4: The curve networks of Figure 3 surfaced using [Abbasinejad et al. 2011] missing a cylindrical surface patch (a) and our approach (b)-(e).

1.2 Design goals:

We observe a few properties from the 9 manually surfaced models that inform the goals of our surfacing algorithm.
Topology: Typically the 3D curves form a topologically connected network. Occasional disconnected curves are embedded into other surfaced curve loops as interpolated surface features or interior cycles/holes (see Figure 5). The surfaced models comprise a small set of connected manifolds, whose number can be user specified.
Geometry: Surface elements tend to interpolate proximal curve segments and result in globally intersection-free surfaces.

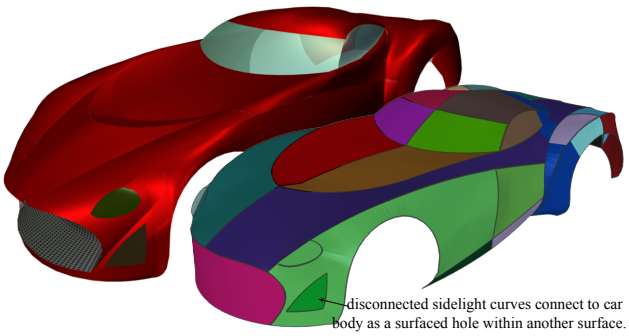


Fig. 5: Designer surfaced car (left) and our algorithmic equivalent (right).

1.3 Previous work

3D shape reconstruction from discrete point-samples has been approached using various techniques: zero-sets of signed distance functions [Hoppe et al. 1992], natural neighbour interpolation [Boissonnat and Cazals 2002], moving least square (MLS) surfaces [Alexa et al. 2001], support vector machines (SVMs) [Carr et al. 2001], and Delaunay-based techniques [Boissonnat 1984; Amenta et al. 2001] to name a few. The line of work most relevant to our work, however, employs flow-based methods [Edelsbrunner 2004; Chaine 2003; Giesen and John 2002; 2003; Dey et al. 2005]. In themselves these approaches are ill-suited to shape reconstruction from a sparse collection of 3D curves.

Most work in the area of model reconstruction from sketched curves addresses 2D sketch input [Shpitalni and Lipson 1996]. In the space of 3D sketch-based modeling, one of two approaches may be taken. The first sketches and surfaces models concurrently, by incrementally sketching curves to refine a surface mesh built by stroke inflation [Nealen et al. 2007; Olsen et al. 2009]. The second, free-form sketching of 3D curves is better suited to conceptual design, but has little or no automated support for surfacing [Bae et al. 2008], barring the recent work of [Abbasinejad et al. 2011].

The choice of which curve loops to surface in [Abbasinejad et al. 2011] is based on a complete ordering of all cycles of the graph induced by the curve network. The ordering is determined by cycle size (smaller preferred), separability (non-separating preferred) and volume of axis aligned bounding-box (smaller preferred). A cycle basis is then built by greedily selecting cycles in order resulting in a collection of connected non-closed manifolds. The approach is simple and reasonable, but being agnostic to the global geometry of the curves, can make poor choices of curve loops to surface. It will for example, pick the missing circular loop in Figure 3d, even though stretching a surface across this loop would intersect other 3D curves. Since a cycle basis must be maintained by the approach, incorrect loops chosen greedily, can impact subsequent choices (see Figure 6). Once the curve loops are chosen they are patched using a variational mapping of a triangulated circle to each curve loop independently [Mehra et al. 2009] or using n-sided patches [Várady et al. 2011]. These approaches may fail for complicated boundary curves and are best-suited to well-shaped convex

curve cycles. Recently [Bessmeltsev et al. 2012] surface complex curve loops by quadrangulating them into multiple 4-sided patches. None of these approaches however, provide a complete solution to the intersection-free surfacing of an unorganized set of 3D curves.

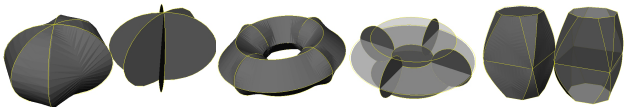


Fig. 6: Comparison of our approach (left) with [Abbasinejad et al. 2011] (right) on a sphere, torus and pot: their greedy algorithm based on a cycle ordering heuristic, incorrectly chooses the planar circular loops on the sphere and torus, and avoids choosing the top and bottom loops of the pot due to large cycle size, with undesirable consequences.

1.4 Contribution and Overview

We present arguably, the first fully automated solution to 3D surface reconstruction from an unorganized collection of 3D design curves. Motivated by the need to capture both geometry and topology information of the input curves within a computational structure, we propose an algorithm based on the *flow complex*, a cell complex that partitions space based on a flow induced by the distance from the input 3D curves. We sample our curves in such a way to ensure that they are precisely embedded within the flow complex of this discrete sample. The final *output of our algorithm is a triangulated surface that is a sub-complex of the flow complex* that interpolates the input curves and is embedded in $\mathbb{R}^3$ (does not self-intersect). Our algorithm uses the theory of persistent homology to pin down the topological structure of the reconstructed shape, in a way that the number of connected components and voids in the 3D reconstruction can be precisely specified by the designer. We optionally condition this triangulated shape further using morphological mesh operations. Figures 10, 14 illustrate the various steps of our algorithm.

The rest of the paper is organized as follows. Section 2 reiterates the surfacing goals underlying our choice of a flow complex based algorithm, and a conceptual understanding of both the flow complex and our surfacing algorithm. Section 3 formalizes terminology pertaining to the flow complex, persistent homology, filtrations, and other constructs, used by our algorithm in Section 4. We present our results and evaluation in Section 5 and conclude in Section 6. We further provide theoretical justifications of our algorithm as an Appendix and a Glossary of mathematical terms used.

2. DESIGN RATIONALE: THE FLOW COMPLEX AND PRISONERS IN A CAGE

We aim to reconstruct a 3D mesh model of controlled topology that interpolates a set of input 3D curves, keeping in mind the design goals from Section 1.2 (awareness of both input curve topology and global geometry, and robustness to inaccurate sketch curve input). Note that the determination of the curve loops to surface and the self-intersection free surfacing of these loops are co-dependent problems that should be solved simultaneously.

Many algorithms for shape reconstruction from unorganized point-sets restrict their attention to the Delaunay complex, which simultaneously offers geometric structure and a reduced choice of surfacing elements that interpolate the input points. When reconstructing a surface from a sparse collection of curves, as opposed to a dense collection of points, there is a shift in importance from geometry towards the topology of the input curves. We thus utilise a related cell complex, the *flow complex* (albeit applied to a point sampling of curve input), that is more informative topologically.

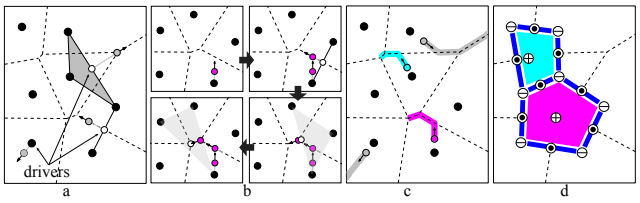


Fig. 7: Distance function characterization on 7 input points (black): (a) Voronoi boundaries (dashed lines) and drivers for grey points; (b) flow of a magenta point under multiple drivers; (c) flow trajectories for four points; (d) flow complex comprising input points *0-cells*, Gabriel edges in blue *1-cells*, magenta and cyan polygons *2-cells*. The critical points of these cells are marked as minima $\ominus$, saddles $\odot$, and maxima $\oplus$.

2.1 Flow complex Overview

We begin with two mathematically equivalent characterizations of the flow complex both of which aid in the understanding and implementation of our algorithm: *distance function* and *point inflation*. We illustrate these characterizations using Figures 7, 8 respectively, for a set of example 2D input points.

Distance function. An input point-set P defines a distance function d_P for all points in space. Given any point q, $d_P(q) = \min_{p \in P} \|q - p\|$, or the closest Euclidean distance to the point-set P. The familiar Voronoi boundaries of d_P are shown as dashed black lines in Figure 7. The function d_P in turn induces a *flow* under which points follow trajectories of *steepest ascent* of d_P. For a point q, $\mathrm{Driver}(q)$, defines this direction of steepest ascent as $(q - \mathrm{Driver}(q))$. Formally, $\mathrm{Driver}(q) = \mathrm{argmin}_{s \in S}(\|q - s\|)$, where $S = \mathrm{ConvexHull}(\mathrm{argmin}_{p \in P}(\|q - p\|))$. For points q not on Voronoi boundaries, $\mathrm{Driver}(q)$ is a uniquely closest input point $p \in P$, and the direction of steepest ascent $q - p$ is the gradient of the smooth function d_P as expected. The $\mathrm{Driver}(q)$ for points q on Voronoi boundaries is the closest point to q within the convex hull of the multiple points in P that are closest to q. Figure 7(a) shows the drivers as white points for three example grey points. The direction of steepest ascent vanishes for a finite number of points that are coincident with their drivers. These points are called *critical points* (see Figure 7(d)).

Any point thus flows along a linear trajectory, away from its driver until it hits a Voronoi boundary, from where it flows under a different driver until it hits another Voronoi boundary, finally escaping to infinity or ending in a critical point. Figure 7(b) illustrates how the magenta point flows, pushed by multiple drivers shown in white, until it coincides with its driver at a critical point. Figure 7(c) shows the the flow of the magenta point and three other points. The cyan point also ends in a critical point marked $\oplus$ in Figure 7(d) and the two grey points escape to infinity (considered a critical point).

The resulting flow complex is thus a subdivision of the plane into a number of cells, where each cell comprises a critical point and all the points whose trajectories flow into that critical point. As shows in Figure 7(d) the *0-cells* (marked $\ominus$) are the input points themselves. The *1-cells* are line segments connecting two *0-cells*, and *2-cells* are regions of the plane separated from each other by *1-cells*. The flow complex (like the Delaunay complex) provides an

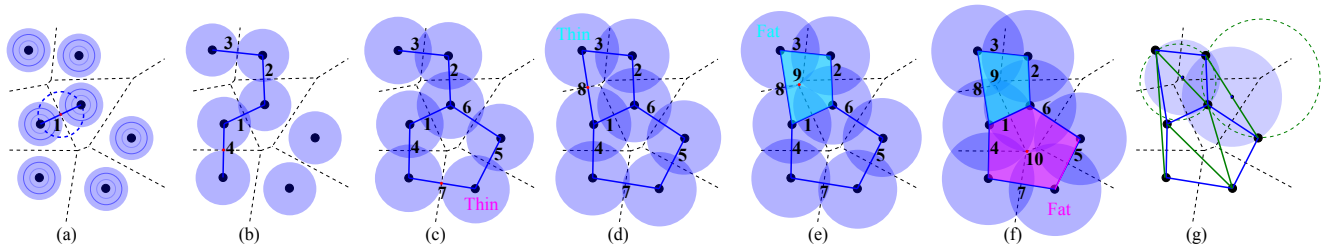


Fig. 8: Point inflation characterization on 7 input points (black): (a) formation of the first Gabriel edge; (b) more edges numbered in order of creation; (c)(d) edges may enclose a region of space forming a polygon that is eventually filled (e)(f), the radius at which a polygon is created is marked Thin and the radius when it collapses is marked Fat; (g) Gabriel edges are a subset of Delaunay edges in 2D.

intersection-free choice of surface elements based on the geometric proximity, in keeping with our design goals (see Section 1.2).

Point Inflation. An alternate conceptualization of the flow complex is to imagine inflating the input points or *0-cells* with circles of uniformly increasing radii (roughly iso-curves of the distance function) as exemplified in Figure 8a. Eventually the circles around some two proximal points will touch at a critical point shown in red (when the radius reaches half the edge length) creating a connected area represented by an edge (*1-cell*) connecting the points. Such an edge, called a *Gabriel* edge, is equivalently characterized by its diametric circle, shown in dashed blue, not containing any other input points in Figure 8(a). In 2D while all Gabriel edges are Delaunay edges, the reverse is not true (the green edges in Figure 8(g) are Delaunay but not Gabriel). One can observe that only Delaunay edges that intersect their Voronoi dual are Gabriel edges (see Figure 8(g)). As the points continue to inflate more edges are created (Figure 8(b)) and further three or more created edges may form a polygon Figure 8(c), the interior of which will eventually collapse, into a single connected area or *2-cell*, at a critical point shown in red (see Figure 8(e)(f)). The *index* of a critical point is the dimension of the cell it represents. These created edges and regions along with the input point-set comprise the geometry of the 2D flow complex, as already described using the distance function characterization. More importantly, the radius induced order in which these elements are created and collapsed or destroyed (the numbering in Figure 8) encodes the construction topology of the complex. Using techniques from the theory of persistent homology [Edelsbrunner et al. 2002a] our algorithm picks an appropriate subset of these topological events and uses their corresponding subcomplex of the flow complex as representative of the reconstructed shape in Section 4.3.

In 3D, the input curves are inflated as generalized cylinder volumes of uniformly increasing radii. Any closed curve loop in isolation will appear toroidal (see Figure 9(left)), with the hole collapsing at a large enough radius. This defines a face or *2-cell* and collections of these faces can enclose a cavity or void, which eventually collapses as well when the radius becomes large enough. While the flow complex can be theoretically defined for continuous curves, no algorithm to compute it is known, even for poly-lines. On the other hand, computing the complex for a discrete set of points is well-understood. Furthermore, by sampling the curves appropriately, we can guarantee that the poly-line representation of the curve will be embedded in the edges (*1-cells*) of the flow complex of its point samples. Figure 9(right) shows a discrete point sampling of the curves in Figure 9(left), where the curves are captured as *1-cells* or Gabriel edges. In 3D, *2-cells* are piecewise triangular patches

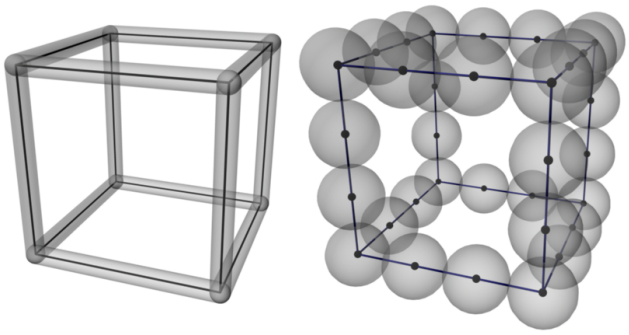


Fig. 9: Inflation in 3D around curves (left), around a discrete set of points that sample the curves, where each curve segment is a Gabriel edge (right).

whose boundary is composed of a closed chain of Gabriel edges. In general, a *2-cell* is not composed of Delaunay triangles but it is possible for a *2-cell* to coincide exactly with a Delaunay triangle. These *2-cells* separate 3D space into a number of volumes, voids, *3-cells* or metaphorical *prison* cells. Given this flow complex, our reconstruction algorithm, in essence, selects a subset of these prison cells to represent the intended voids and outputs the collection of triangles in 3D corresponding to *2-cells* that bound disjoint prison cells.

2.2 Prisoners in a cage

Most of the reconstruction literature deals with dense and isotropically sampled point data, typically received from 3D data acquisition equipment. It is relatively easy for such techniques to define geometric criteria relating to sampling accuracy, frequency, and distribution, that the acquisition device must provide, in order for these techniques to provide guarantees on the geometric and topological fidelity of the reconstructed shape. Our problem is significantly harder, since we deal with sparse human authored input. It is both difficult, to convey geometric parameters such as sampling frequency or distribution to a designer and also, unreasonable to expect the designer to draw a collection of curves that conform to those parameters. Despite this, successful surface reconstruction from design sketches can benefit from a two-way understanding between designer and algorithm, given the inherent ambiguities that can exist in the perception of 3D curve networks (see Figures 2, 3).

While we strive to capture designer intended shape from 3D curve networks automatically, it is valuable if designers can con-

ceptually understand the working of our reconstruction algorithm. This allows designers to obtain desired surfaces by creating and editing the curve network with predictable results. Our flow complex approach supports such design intuition:

Imagine the desired shape as a collection of connected closed manifolds with boundaries defined by the input curves: the enclosed voids are *prison cells*, and the input curves are *cage bars* to the prison cells. Essentially, we require that all designer desired prison cells be sufficiently well-defined by the enclosing cage bars.

To elaborate on this idea, imagine the prisoner as a ball placed, for example, within a cell whose input curve cage is formed by edges of a unit cube (see inset). A solid ball of radius $1/2 < r < \sqrt{2}$ centered inside the cube, can both fit in the cage and cannot escape it (at no point must the ball intersect the cage bars). Geometrically, a ball of increasing size escaping through cage bars is equivalent to inflating radii around cage bars in the flow complex definition. We show these balls in 2D for the magenta and cyan prison cells in Figure 8(c)-(f).

The designer's idea of the target shape described by a curve network, is the collection of surface patches or walls that would both interpolate the curves, and separate space into a number of intended voids or prison cells. The curves or cage bars thus need to provide definition to the cells, in order to algorithmically infer the cells and consequently the walls. A good cage, in terms of design intuition and from our algorithmic perspective, is one that isolates the intended cells well; the cage allows both free movement within and prevents escape from each intended cell for a wide range of prisoners (balls of different radii).

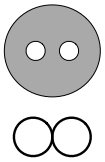
Imagine thus, two prisoners *Thin* and *Fat* represented as balls, imprisoned in an intended cell by its cage. *Thin* is the smallest prisoner that cannot escape the cage but roam freely within it and *Fat* is the largest prisoner that can fit in the cage (*Thin* and *Fat* are $1/2$ and $\sqrt{2}$ in radius for the inset cube). Our algorithm will capture those cells where *Thin* and *Fat* differ by a significant factor. Technically this difference is tantamount to choosing persistent cells (ones with significant difference between their creation and destruction radii) in the flow complex. For example, the magenta cell in Figure 8 has a larger difference between *Thin* and *Fat* radii, than the cyan cell. The magenta cell is thus more persistent than the cyan cell and between the two, considered more likely to represent a closed manifold.

In the next section we provide the technical formalism needed, to describe our algorithm that finds surfaces that interpolate the input curves and bound the most persistent prison cells. An informed reader may skip to the proposed algorithm in Section 4, using the Glossary of terms, if needed as a reference.

3. TECHNICAL FORMALISM

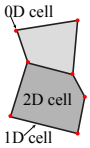
We now define key concepts and tools used by our algorithm: the flow complex, homology, filtrations, persistence and cell collapses. For further details see [Giesen and John 2003; Buchin et al. 2008].

Flow complex. The flow complex applies the classical notion of a Morse complex to a generalized gradient of the distance function in place of a smooth vector field. It is a cell complex (a CW-complex to be precise) induced by the point set P in which the cells are in one-to-one correspondence with critical points of d_P. The *cell* or *stable manifold* of a critical point c consists of all points x in space whose flow trajectories end at c. Generically, each *i-cell* is homeomorphic to an i-dimensional *open* ball and excludes its relative boundary in the complex, so that all cells together define a partition of space.



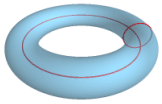
Topological equivalence. Classically, topological equivalence between two topological spaces is defined as a *homeomorphism* between them, i.e. a continuous 1-to-1 map with a continuous inverse. *Homotopy equivalence* is a weaker notion of a homeomorphism that does not preserve dimensions. For example, a disc with two holes and the figure eight are not homeomorphic but are homotopy equivalent. A still weaker version of homotopy equivalence, but one that is computationally tractable, is *homology equivalence*. Our algorithm uses a powerful technique known as *persistent homology* to detect intended prison cells, or volumes of space that are caged by the input curve network. We briefly define homology groups and their associated concepts and point readers to [Edelsbrunner et al. 2002b] for a thorough treatment of the subject. We only use a special version of homology known as *simplicial* homology under $\mathbb{Z}_2$.

Simplicial homology. Simplicial homology is defined over simplicial complexes. The notions of homology and persistence we discuss however, can be applied to the flow complex even though its cells are not necessarily simplices. We thus use the term cell instead of simplex in our discussion of homology below.



If a cell σ is a subset of the relative closure of a cell τ, then σ is called a *face* of τ (an edge that bounds a polygon in the inset is a face of the polygon in this context). We call a subset of *i-cells* of a cell complex K an *i-chain*, for example a collection of edges of the inset complex form a *1-chain*. The *sum* of two *i-chains* comprises the *i-cells* that appear in only one of the two summands (equivalent to algebraic addition in $\mathbb{Z}_2$). The boundary of an *i-cell* is the *(i − 1)-chain* comprising its $(i-1)$-faces (as expected, the boundary of a polygon in the inset complex is its bounding edges). The boundary of an *i-chain* is simply the sum of the boundaries of its constituent *i-cells*. The boundary for the inset 2-*chain* of polygons is as expected the 1-*chain* of edges around both polygons since the common edge cancels out. Conversely, an *i-boundary* is the boundary of some $(i+1)$-*chain*. Finally, an *i-cycle* is an *i-chain* that has an empty boundary. The edges of a polygon in the inset thus form a 1-*cycle*, since summing the 0-*boundaries* (incident points of each edge), counts each point twice, i.e. an empty boundary.

The above terminology in place, we define two *i-cycles* in a complex as *homologous* if their sum is an *i-boundary*. When two *i-cycles* are homologous, one can be continuously deformed into the other without leaving the complex; thus they represent, in topological terms, the same cycle. This homologous relationship partitions all *i-cycles* into equivalence *homology classes* (see Figure 8).



The i-th *Betti* number $\beta_i(K)$, is the number of homology classes that form the i-homology group. Intuitively, $\beta_i(K)$ counts the number of topologically distinct *i-cycles* in K. In particular, for a 3D complex β_0 counts the number of connected components, β_1 counts the number of holes and handles, and β_2 counts the number of enclosed voids or prison cells. For a closed orientable surface of genus g, $\beta_0 = 1$, $\beta_1 = 2g$, and $\beta_2 = 1$.

Homology groups are topological invariants and from our perspective a simple yet powerful designer directive to describe the

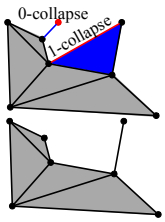
overall topology of the intended 3D shape represented as a collection of 3D curves.

Filtrations and persistence. A *filtration* of a cell complex is a total ordering of its cells, in which each cell appears later than any of its faces, for example the numbering of flow complex elements in Figure 8. One can prove that in a cell complex filtration, the homology groups change each time an i-*cell* is added to the partially developed complex [Edelsbrunner et al. 2002b]: introducing an i-*cell*, either *creates* a new class of i-*cycles*, incrementing β_i, or *destroys* a previously non-trivial class of $(i-1)$-*cycles* by turning them into $i-1$-*boundaries*, decrementing β_{i-1}. In Figure 8 edges labeled $7, 8$ when added, create new 1-*cycles*, while edges $1-6$ when added, destroy 0-*cycles* by turning the two incident points into a 0-*boundary* and decrementing the number of components. Every cell can thus be labeled a *creator* or *destroyer* in a filtration. Adding a 2-*cell* in 3D similarly, either creates a new 2-*cycle* by enclosing a void, or destroys the 1-*cycle* that had been formed by its boundary edges. Conversely, each i-D homology class is created by some creator i-*cell* and later destroyed (if at all) by the arrival of a destroyer $(i+1)$-*cell*. If the creator cell is denoted by $\sigma^{(i)}$ and its corresponding destroyer by $\tau^{(i+1)}$, then the pair $(\sigma^{(i)}, \tau^{(i+1)})$ is called a *persistent pair*. In Figure 8 for example, elements $(7, 10)$ and $(8, 9)$ are persistent pairs. Persistent pairs can be determined algorithmically and the interval between their creation and destruction provides a comparative measure of *persistence* of topological features in the evolution of the cell complex with respect to the given filtration. In our context, persistence is better quantified as a function of the radii at which the elements are added to the evolving complex than simply using their rank in the filtration.

Persistence captures our design intuition of well caged and roomy prison cells. In 2D, the radius at which a 1-*cycle* is created, marked *Thin* in Figure 8, is the smallest prisoner that cannot escape the 1-*cycle*. The radius at which this 1-cycle is destroyed, marked *Fat*, is the largest prisoner that can fit within the cell, without intersecting any of its input points. If our intended 2D shape is to have only 1-*cycle*, we will thus choose the more persistent magenta cell over the cyan cell. Generally homology classes with small persistence can be regarded as *topological noise* while those with large persistence mark substantive topological features that the designer intends to convey using the set of input 3D curves.

Filtration by distance value. Our algorithm uses a filtration of the flow complex in which all *0-cells* and *1-cells*, appear before any of the *2-cells* and *3-cells*, which themselves appear in increasing order of the value of the distance function at their corresponding critical points. Under this filtration we mark 2-*cells* as destroyer or creator. We further use the logarithm of the distance function in computing the persistence of creator 2-*cells* (or the 3-*cells* paired with them). The persistence of a creator 2-*cell* σ paired with a 3-*cell* τ is thus $\log d_P(\tau) - \log d_P(\sigma) = \log \frac{d_P(\tau)}{d_P(\sigma)}$. This choice is justified in the Appendix.

Collapses. A generic tool used by our algorithm is the *collapse* operation. Given a sub-complex K of the flow complex, a *free cell* in K is any i-*cell* σ that is incident to exactly one $(i+1)$-*cell* τ in K. Collapsing the free cell σ removes from K both σ and τ. The resulting subcomplex, denoted by $K \downarrow \sigma$, is homotopy equivalent to K (they have the same Betti numbers).



Inset, as an illustration of a 1-*collapse*, removing the free 1-*cell* (red edge) also removes its incident 2-*cell* (blue triangle). An example 0-*collapse* of the free red point is also shown. In our algorithm we use 1- and 2-*collapses* to remove elements of the flow complex that are undesirable for our reconstructed surface (see Figure 10(b), (c)).

We say that a sub-complex K' is a *1-collapse* of a subcomplex K, or equivalently K *1-collapses* to K', if K' is obtained from K through an arbitrary sequence of 1-*collapses*. We also define a restricted 1-*collapse*, called a 1_{res}-*collapse*, where free 1-*cells* or edges that correspond to segments of the discretized input curves are not collapsed. A 1_{res}-*collapse* ensures that all curve segment edges retain at least one incident surface (2-*cell*) after the collapse operation, preserving a surface that interpolates the input curves.

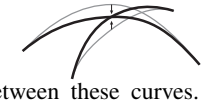
Armed with concepts and tools our approach uses, we now detail our algorithm that captures the intuition described in Section 2.2.

4. ALGORITHM

The input to our algorithm is a curve collection $\Gamma = \gamma_1 \cup \ldots \gamma_n$ where each γ_i is a 3D curve. We assume that Γ samples, and is contained in a target shape Σ for which each connected component of $\Sigma - \Gamma$, called a *target patch*, is homeomorphic to an open disk from which zero or more closed disks are removed (a patch can have holes). Each connected component of $\mathbb{R}^3 - \Sigma$ is called a target void or prison cell, whose number is the second Betti number β_2^* of the target shape Σ. Generally the number of disjoint voids or volumes β_2^*, of the intended shape is inherent to the design and best defined as a user parameter. Alternatively, as we shall see, our algorithm allows β_2^* to be automatically determined based on a user-defined persistence threshold, or interactively varied by the designer while browsing the resulting output shape.

Our algorithm first uses a feature size ϵ to create a discrete point-set P representative of the input curves Γ in Section 4.1. The flow complex $\mathcal{F}_P$ of this point-set P is then computed using [Giesen and John 2003] in Section 4.2. We then pick a sub-complex of $\mathcal{F}_P$ that matches the intended shape topology using the β_2^* most persistent prison cells and ensure that all *2-cells* are bounded by input curves, in Section 4.3. This reconstruction, while topologically correct often has geometric inaccuracies, such as cut-off corners and crevices; regions filled by small voids of low persistence that are not chosen by our topological reconstruction. We solve these problems by *thickening* the reconstruction based on a threshold t, to incorporate more voids in Section 4.4 and finally prune the thickened reconstruction in Section 4.5 to better conform the output shape to the input curves. Our algorithm thus has precisely three user controllable parameters ϵ, β_2^* and t with automatically computed defaults and the final output of our algorithm is a sub-complex of $\mathcal{F}_P$ that interpolates the input curves Γ (see Figure 10).

4.1 Discretizing the input



Input curves from sketches often have imprecise intersections. We correct these, since faithfully interpolating such curves will create small, undesirable geometric bridges between these curves. Thus, if the closest points on two curves are within a small distance as shown inset, we snap them to their mid-point and smoothly deform the curves in the neighbourhood of the intersection.

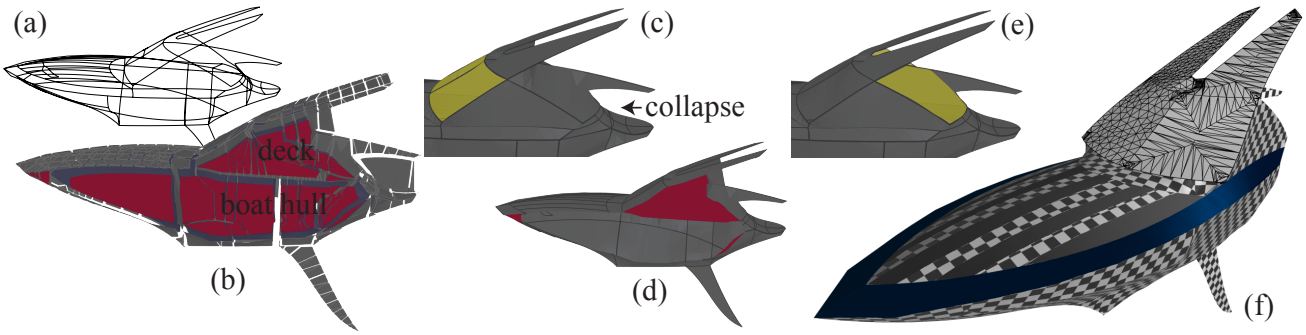


Fig. 10: Algorithm stages: (a) input curves; (b) exploded view of the flow complex of the sampled curves, a cross-section revealing *3-cells* in red; (c) the topological reconstruction correctly captures the boat hull void and collapses any surface patches not bounded by input curves, but surfaces the back of the boat deck leaving the windshield open (yellow highlight); (d) the topology preserving thickening of (c), adds surfaces, such as the windshield, and *3-cells* shown in red; (e) the added *3-cells* are removed along with potentially different surfaces, such as the back of the deck (yellow highlight) for a better geometric interpolation of the input than (c); (f) a rendered result with conformally parameterized surfaces, the windshield shows both the flow complex triangulation (right) and the result of morphological remeshing (left).

Now, given that flow complex algorithms are currently only known for point-sets, we sample each input curve as a poly-line, turning Γ into a polygonal network whose vertices are the discretized input point-set P. There are two considerations in ensuring that the flow complex of P is an adequate representation of the flow complex of Γ (see Figure 9). One, the point sampling along curves needs to be dense enough that the points are a good proxy for curves as cage bars, traded-off against algorithmic efficiency. Two, we require that each poly-line curve be embedded within the edges of the flow complex, since our output surface is a subset of the flow complex, in other words, each segment of an input poly-line curve, must be Gabriel (see Section 2.1).

Our initial sampling of the curves is regular and arc-length based. A reasonable sampling distance for a curve is based on the minimum Hausdorff distance to all other curves in Γ (in practice we use half this distance). This aims to ensure that a *Thin* prisoner caged by the continuous curves cannot simply escape between two point samples along a curve. Our sampling distance must also be no more than a given feature size ϵ, to ensure a geometrically acceptable sampling of Γ. The parameter ϵ can be user defined or based on a error tolerance of deviation between the input curve and its poly-line equivalent.

We then "Gabrielize" this initial poly-line network of curves by iteratively subdividing non-Gabriel segments until each resulting sub-segment becomes Gabriel. A segment pq, where p and q are adjacent points along a poly-line curve, is Gabriel if no third point r encroaches upon it, i.e. lies in the interior of the ball of diameter pq. Observe that for a vertex r encroaching upon a segment pq, if s is the projection of r onto pq, then r does not encroach upon either of ps or sq. Subdividing segment pq by inserting point s thus makes the segments ps and sq Gabriel, at least with respect to point r. In principle such an iterative subdivision of any remaining non-Gabriel segments can be used to produce a Gabriel poly-line curve network in a finite number of steps.

We note however, that successive projection of encroaching points to affected segments is numerically unstable, particularly where two curves approach each other at a small angle (see Figure 11(a)). Numerically inaccurate projections do not eliminate encroaching and can lead to indefinite subdivision near the joining vertex v, yet addressing this using an exact arithmetic approach is unnecessarily expensive. One practical solution is to eliminate

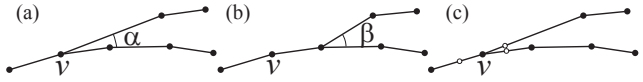


Fig. 11: Avoiding numerical issues in Gabrielizing two curve segments incident to a vertex v at a small angle α (a), by dragging one of the incident curves to a neighbour of v to enlarge the angle to β (b), or subdividing the segments at a small equal distance from v (c).

curve segments that meet at very small angles by *dragging* one of curves along the other curve until the resulting angle reaches a suitable threshold (Figre 11 (a), (b)). Another solution is to subdivide curve segments incident to the joining vertex v by intersecting them with a small ball centered around v (see Figure 11 (c)). We can then numerically search for a radius small enough that the new segments incident to v are Gabriel (by construction these segments will not be affected by subsequent subdivisions).

The set of points P of the Gabriel poly-line curve network computed as above, is now used to compute its flow complex, a subset of which will constitute our final surface output.

4.2 Computing the Flow Complex

The Flow Complex is computed exactly as described in [Giesen and John 2003]. Briefly, the flow complex $\mathcal{F}_P$ of a point-set P is constructed using its Voronoi and Delaunay complex as:
0-cells: The input point-set P.
1-cells: Gabriel edges, i.e. Delaunay edges that intersect their dual Voronoi facet.
2-cells: Triangulated surface patches bounded by Gabriel edges, computed by tracing flow back from each 2-*cell*'s critical point.
3-cells: Volumes enclosed by 2-*cells*.

Figure 12 illustrates 2-*cell* construction by reverse flow tracing. The point of intersection of any Delaunay triangle e^* that intersects its dual Voronoi edge e is the critical point of a 2-*cell*. The three Voronoi facets incident to Voronoi edge e are dual to edges of the Delaunay triangle e^*, and we trace the flow backward recursively along each of the three faces. Consider one such Voronoi face f and its dual Delaunay edge f^*. Let e' be the second Voronoi edge incident to f that intersects the plane of triangle e^*. Either e and e' straddle Delaunay edge f^* i.e. f intersects f^*, or e and e' lie within the the Delaunay triangle e^*. In the first case f^* is a Gabriel

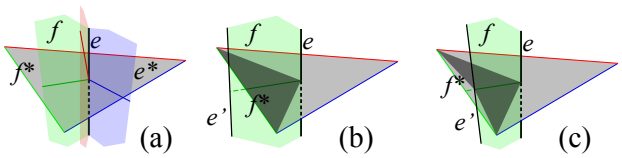


Fig. 12: *2-cell* construction by reverse tracing flow into its critical point: (a) the intersection of a Voronoi edge e and its Delaunay dual e^*; the flow is traced back along the green Voronoi face f in (b) and (c); (b) either f intersects its Delaunay dual f^* and f^* is a bounding Gabriel edge and the darkened triangle part of the *2-cell*; (c) or f does not intersect f^*, and the darkened chevron is added to the *2-cell* and the flow recursively traced along other Voronoi faces incident to edge *e'*.

edge that marks part of the boundary of the 2-*cell* and the triangle formed by this edge and the critical point is added to its 2-*cell*. In the second case the chevron of two triangles, each defined by one incident vertex of f^*, the critical point and the intersection point of e' with triangle e^*, are added to the 2-*cell*. In this case the reverse flow recursively continues along the other Voronoi facets incident to Voronoi edge e'. Note that the reverse flow at each step is toward the driver for the current Voronoi face, which is the mid-point of its dual Delaunay edge. In its simplest form a 2-*cell* can be a single Delaunay triangle with 3 Gabriel edges.

Given the elements of the flow complex, the distance function for a 2-*cell* (or 3-*cell*) is simply the minimum distance from its critical point to an input-point on its boundary. Conceptually, as 2-*cells* are added in order of their distance function, 2-*cells* whose arrival encloses a void are marked as creators and all others as destroyers. Algorithmically, we use the connectivity in the dual graph determined by the 2- and 3-*cells*, to define persistent pairs of creators and destroyers [Buchin et al. 2008]. We also order the 2-*cells* to be in increasing order of their distance function but where all destroyer cells appear before all creator 2-*cells*.

Input: Flow complex $\mathcal{F}_P$ and the persistent paring of its maxima, the desired second Betti number β_2^* of the output
Output: Topological reconstruction as a subcomplex K_{top} of $\mathcal{F}_P$

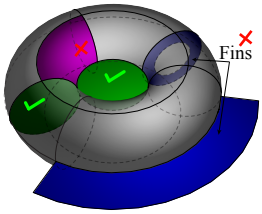
TopologicalReconstruction(β_2^*)

1 $D \leftarrow$ { destroyer *2-cells* of $\mathcal{F}_P$}
2 $C \leftarrow \{\sigma_1^+, \ldots, \sigma_{\beta_2^*}^+ : \sigma_i^+$ is the i-th most persistent creator *2-cell*$\}$
3 $K_0 \leftarrow$ subcomplex of $\mathcal{F}_P$ induced by cells in $D \cup C$
4 $K_1 \leftarrow 1_{res}$-*collapse* of K_0
5 $K_2 \leftarrow$ 1-*collapse* of K_1
6 **for** each *2-cell* σ in K_2 in the decreasing order of d_P:
7 **if** σ is incident to the same void of K_2 on both sides:
8 $K_1 \leftarrow 1_{res}$-*collapse* of $K_1 - \sigma$
9 **goto** line 5
10 **return** $K_{top} = K_1$ as the topological reconstruction

Fig. 13: Topological reconstruction

4.3 Topological Reconstruction

Our algorithm now finds a sub-complex of $\mathcal{F}_P$ that best captures the intended topology the target shape (see Figure 13, 14), as described by the prisoners in a cage metaphor in Section 2.2. The Betti numbers (see Section 3) capture various aspects of shape topology. The number of connected components can be readily determined from input curve connectivity or easily provided by the user. As each component can be reconstructed independently, we assume here that the input forms a single connected component $\beta_0^* = 1$. Now, given the intended number of closed regions or voids β_2^*, let D be the set of destroyer 2-*cells* and let C be the β_2^* creator 2-*cells* with the highest persistence among all creator 2-*cells*. The subcomplex K_0 of $\mathcal{F}_P$ induced by cells in C and D evidently has β_2^* voids ($\beta_2(K_0) = \beta_2^*$). We then ensure that all 2-*cells* in our reconstruction are completely bounded by edges from the input collection of curves, by computing K_1 as a 1_{res}-*collapse* of K_0 (see Section 3). Figure 10(b), (c) shows an example unwanted 2-*cell* removed as a 1_{res}-*collapse*. K_1 has the same homology as K_0, since a collape operation is homotopy invarint.



Including all destroyer 2-*cells* in the construction K_0 implies a shape without topological holes or handles since no 1-*cycle* in K_0 (and therefore K_1) can be non-trivial. Intended shapes such as the torus in Figure 14 can indeed have holes ($\beta_1^* > 0$) and we must remove one or more destroyer 2-*cells* from K_1 to achieve this. As illustrated inset, we must be careful not to remove 2-*cells* that separate distinct voids since we have already settled on the desired number of voids β_2^*. We must also be careful not to remove 2-*cells* incident to the same void on both sides that represent *fins* (2-*cells* removed by a 1-*collapse* but protected by a 1_{res}-*collapse*). Thus we repeatedly delete cells (in decreasing order of distance function) that are incident to the same void on both sides provided that they *would survive* a *1-collapse*. Removing these destroyer 2-*cells* opens up holes and handles in the output shape while interpolating the input curves and preserving the number of voids. The resulting sub-complex K_{top} is returned as the topological reconstruction.

4.4 Thickening the Reconstruction

While the topological reconstruction K_{top} is successful in largely capturing the intended prison cells or voids, nooks and crannys that should be part of the larger void tend to get cut-off, since these tight corners form their own small, low-persistence voids in the flow complex. Figure 15 illustrates this problem in 2D. The intended closed shape for the input points in Figure 15(a) is Figure 15(d). Figure 15(b) in contrast is the result of K_{top}, with all the destroyer edges in black and the red edge that creates the region of largest persistence. While K_{top} has a single closed region as desired in Figure 15(d), a corner is cut and its locality captured as a fin. Further, obtaining Figure 15(d) from Figure 15(b) cannot be achieved by manipulating destroyer cells alone, and requires the addition of the blue creator edge in Figure 15(c). Simply adding this edge to K_{top} however, changes its homology incrementing β_1. We thus "thicken" K_{top} to K_{thick} by adding both the blue edge and its persistent pair (the blue region), which leaves the shape homology unchanged. Subsequently, we will use geometric criteria to remove the blue region along with the destroyer black edge in the interior of the shape to produce Figure 15(d), using a *1-collapse*.

In 3D, thickening the reconstruction entails adding a number of creator 2-*cells* (and their paired 3-*cells*) other than the β_2^* creator 2-*cells* chosen by K_{top}, in a manner that leaves homology of the thickened sub-complex K_{thick}unchanged. We control thickening using a distance threshold t that bounds the distance function of any cell added as a result of the thickening. K_{thick} is a superset of K_{top}, algorithmically constructed by first adding all creator 2-*cells* and all 3-*cells* with distance function $d_P < t$, that are not already

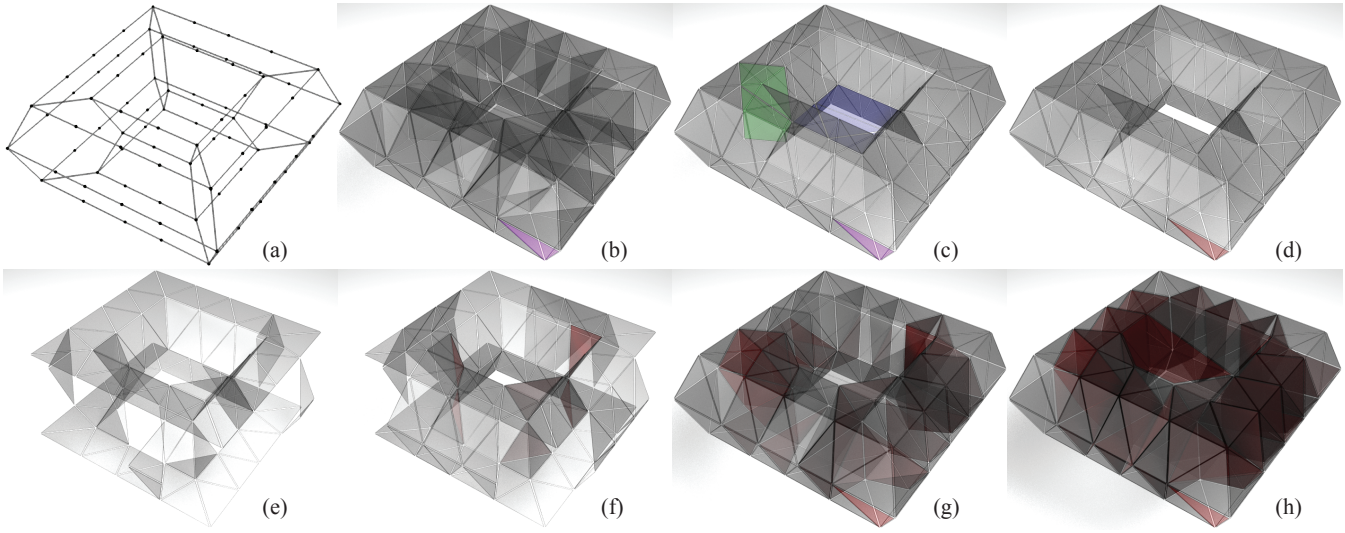


Fig. 14: Topological reconstruction: (a) torus curves sampling; (b) all destroyer 2-*cells* with the highest persistence creator 2-*cell* in magenta; (c) destroyers from (b) that trivialize 1D homology (block the hole and handle) shown in blue and green; (d) the topological reconstruction K_{top}. The bottom row (e)-(h) shows stages of $\mathcal{F}_P$ as elements are added in order of d_P. (e) a few destroyers in grey; (f) more destroyers and a few low persistence creators in red; (g) more elements including the creator with highest persistence; (h) complete flow complex $\mathcal{F}_P$.

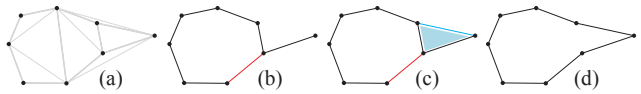


Fig. 15: Thickening in 2D: The input points in (a) result in the topological reconstruction (b) with black destroyer edges and a single red creator edge. We thicken the reconstruction (b) by adding the blue creator edge and its paired blue region (c) and then prune it based on geometric criteria to obtain the desired reconstruction (d), always preserving the topology of (b).

present in K_{top}. We then iteratively remove some of these added cells to ensure a homology invariant thickening: added 2-*cells* missing their paired 3-*cells* are removed; added 3-*cells* missing any of their boundary 2-*cells* are removed. K_{thick} thus provably has the same homology groups as K_{top} by construction.

A small threshold t generally thickens by adding 3-*cells* that are both small in size and of low persistence, since both the creator 2-*cell* and its paired 3-*cell* have $d_P < t$. Thickened reconstructions resemble the accumulation of dirt in models, initially jamming small 3-*cells* into tight corners and adding larger 3-*cells* with increasing threshold (see Figure 16). From our experiments, a good default threshold for t is the distance function associated with the creator 2-*cell* with the largest persistence. This threshold captures the size of the largest 2-*cell* that bounds the most persistent prison cell and it is reasonable to expect that candidate creator 2-*cells* that might improve the geometric quality of our reconstruction have a similar or smaller size. Once the flow complex and distance function filtrations have been computed it is also possible for a user to control the amount of thickening in real-time by interactively manipulating t.

4.5 Pruning the Thickened Reconstruction

A thickened reconstruction may on its own be sufficient or even preferable for certain applications such as hidden surface removal. Nevertheless, for many applications a strict surface output is

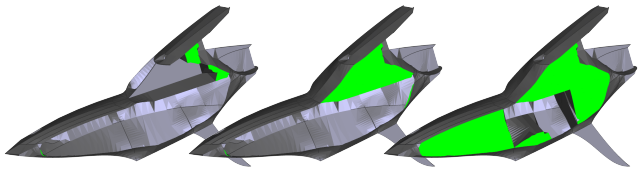


Fig. 16: Thickened reconstructions by increasing distance threshold (default middle) viewed as cross-sections with added 3-cells shown in green.

Input: Flow complex $\mathcal{F}_P$ and the topological reconstruction K_{top}
Thickening threshold t
Output: Thickened reconstruction as a subcomplex K_{thick} of $\mathcal{F}_P$

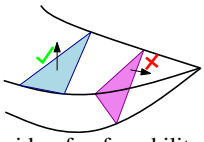
THICKENRECONSTRUCTION(K_{top}, t)
1 $K \leftarrow K_{top} \cup \{\sigma \in \mathcal{F}_P : \sigma$ is a *3-cell* or creator *2-cell* with $d_P(\sigma) < t\}$
2 **repeat**
3 $\quad K \leftarrow K - \{\sigma^{(2)} \in K - K_{top} :$
$\quad\quad (\sigma, \tau^{(3)})$ is a persistent pair and $\tau \notin K\}$
4 $\quad K \leftarrow K - \{\tau^{(3)} \in K : \tau$ is incident to a *2-cell*$\sigma^{(2)} \notin K\}$
5 **until** nothing else can be deleted from K
6 **return** $K_{thick} = K$

Fig. 17: Thickening the initial reconstruction

sought, devoid of any 3-*cells*. We achieve this through 2-*collapses* (removing a free 2-*cell* and its incident 3-*cell*), leaving the homology unchanged. Determining which 2-*cell* σ to collapse employs a feasability function $f(\sigma)$ that is based on the geometry of the cell, the input curves or other application specific criteria. Cells are then removed greedily from a priority queue in order of increasing feasability so that the final reconstructed surface is largely comprised of the most feasible 2-*cells*. Each collapse may turn some new 2-*cells* free (by removing one of their two incident 3-*cells*) or block the future collapse of some previously free 2-*cells* (by removing their only incident 3-*cell*). We thus ensure that the head of

the queue about to be collapsed is still a free 2-*cell*, and insert any newly freed cells into the queue after each collapse.

The collapse of a 2-*cell* often frees one or more 1-*cells*. Once all 3-*cells* have been pruned by 2-*collapses*, these free edges and their associated 2-*cells* can be removed by a 1_{res}-*collapse*. Conceptually however, the cascade of 2-*cells* associated with collapsing free 1-*cells* initiated by a collapsed 2-*cell*, are best processed together, as a larger surface patch whose boundary is strictly comprised of the input curves. Grouping the 2-*cells* into these surface patches and then treating each group of 2-*cells* collectively provides a larger context over which to compute the feasability function.



We settled on a simple yet effective pruning feasibility function, based on the observation that a representative surface normal of a feasible surface patch tends to be orthogonal to the tangent direction of the input curves that bound it. We implement this idea for feasability of a 2-*cell* σ, as a measure of the angle between the normal n of the Delaunay triangle that contains the critical point of σ, and the input curve tangents t_i at the triangle vertices. Specifically, $f(\sigma) = 3 - \sum_{i=1}^{3} |n.t_i|$. For a vertex with more than one incident input curve, we pick the curve that minimizes $|n.t_i|$. We also experimented with other feasability heuristics, such as favouring surface patches with low total mean curvature (designers typically utilize more curves in high curvature regions of shape, resulting in relatively flat surface patches) that were less successful in practice.

5. RESULTS AND EVALUATIONS

Implementation. We implemented the algorithm using CGAL Lazy-Exact kernel. Our naive implementation of the clean-up and discretization of input curves (Section 4.1) takes about the same amount of time, as the subsequent computation of the Flow complex (Section 4.2). The complexity of finding all critical points of the distance function of a point-set in $\mathbb{R}^3$ is linear in terms of the Delaunay complex of that point-set, i.e. $O(n^2)$ for a set of n points. The algorithm of [Giesen and John 2003] for computing the flow complex runs in linear time in terms of the total number of triangles comprising the *2-cells* of the flow complex. A loose bound on the algorithm is $O(n^4)$, for n input points, but to our knowledge, no point-set is known to have a flow complex with more than $O(n^2)$ triangles. The flow complex algorithm on a MacBook Pro laptop with a 2.3GHz intel Core i5 processor takes between 20 seconds and 1 minute for curve network examples in Figure 23 approximated by around 4000 points. Once the flow complex and filtrations are computed, the rest of the algorithm (Section 4.3-4.5) takes a few seconds to compute, and vary parameters like β_2^* or the thickening threshold t.

Testing. The algorithm was tested on a variety of both synthesized and real-sketch input (see Figure 23). We first tested the model on 6 analytic primitives represented as curve networks, such as the torus and trebol shown in Figure 23(top). In each case it reconstructed the surfaces as expected. Compared to our 9 manually surfaced ILoveSketch models, it further produced results consistent with designer created surfaces. We also tested the algorithm's balance between geometry and topology in surfacing the topologically identical tubular structures shown in Figures 3, and 23 (rows 2 and 3). The cylindrical surfaces were consistently surfaced. The top circular cap depends on the configuration of the geometric cage and are either closed (row 2 left, row 3 right) or open in (row 2 right, row 3 left), in keeping with the intuition of a cage that is roomy but prevents escape. We also ran our algorithm on the ILoveSketch curve benchmark, as well as examples generated by the analytical sketching tool of [Schmidt et al. 2009]. We acknowledge imperfect results in about 10% of the models but broadly, the results are quite expressive of the intended objects (see Figure 23(middle, bottom)).

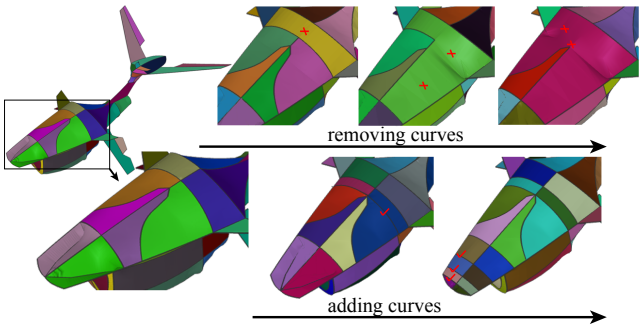


Fig. 18: Impact of adding or removing curves from the input.

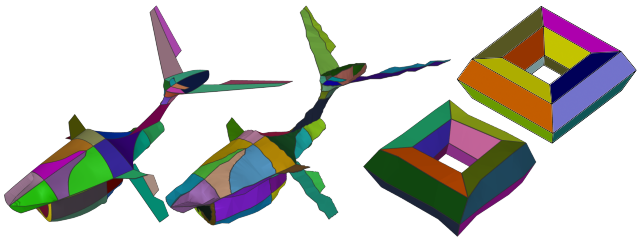


Fig. 19: Impact of adding noise to the input curves.

Robustness. We also tested our algorithm's resilience to the addition or removal of curves as well the addition of noise to the input curve geometry. Figure 18 shows how our surfacing algorithm is impacted by succesively adding (red ticks) and removing (red crosses) input curves to the top of the spacecraft (zoomed in). Despite the successive removal of 5 significant curves (shown as three stages in the top row), our algorithm correctly surfaces the top of the spacecraft to enclose the fuselage void. Eventually however, a lack of cage bars will cause other lower persistence prison cells to be chosen and the overall surface structure to change. Similarly, input curves added to reaffirm a successfully reconstructed model (two stages in the bottom row), reinforce the prison cells with additional cage bars. This both preserves the topological structure and improves the geometric quality of the reconstruction (interpolating the added curves near the nose-tip reduces its otherwise pinched appearance). It is also important that our approach be robust to geometric noise that can result from sketching or scanning inaccuracies. While large amounts of noise can alter the global structure of the Delaunay and flow complex and our surfaced output as a result, Figure 19 shows our approach to be robust to uniform noise (of magnitude upto 10% of the model's bounding box diagonal) applied to the input points. Note that the input curve network may have to be resampled after the addition of noise to ensure that all curve segments are Gabriel (see Section 4.1).

Comparison. A variety of prior art addresses different aspects of our problem: processing a collection of input curves, determining which cycles to surface for an intended shape topology and creating an intersection-free surface that interpolates the input curves. One advantage of our approach is that we do not require a pre-

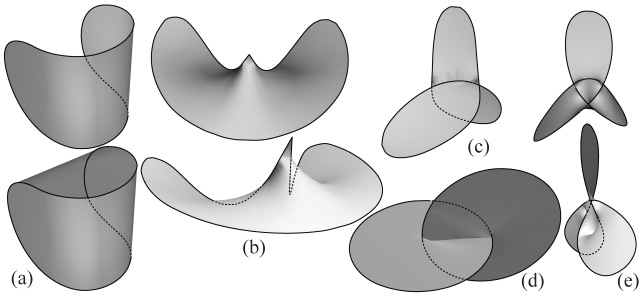


Fig. 20: Complex closed curves patched and morphologically remeshed: (a) rocker ($\beta_2^* = 0, 1$); (b) bat-wing (two views); (c) folded curve (not knotted); (d) linked circles; (e) trefoil knot (two views).

cisely connected network of input curves, but can handle an arbitrary collection including unconnected curves representing surface holes (see Figure 5), feature lines, or even additional points (see Figure 21) to be interpolated. In contrast approaches strongly driven by the topological connectivity of curves [Abbasinejad et al. 2011][Bessmeltsev et al. 2012] are sensitive to how well the input curves are processed into a single curve network (see Section 4.1) and have no way of incorporating floating curves and points into the resulting surface.

The determination of which curve cycles to surface as addressed by [Abbasinejad et al. 2011] is based on a heuristic algorithm that can fail with global impact on the curve cycles chosen (see Figure 4, 6). We view [Abbasinejad et al. 2011] as complementary to our approach: once our topological reconstruction over the reduced space of flow complex elements has defined the voids (their approach assumes a collection of open manifolds), their heuristics can help define our feasbility metric for pruning the thickened reconstruction in Section 4.5.

Work focused on surfacing designer curve cycles already identified for surfacing, such as [Bessmeltsev et al. 2012], works well for many designer curve cycles but unlike our approach, is not intersection-free and fails for more complex or arbitray curve cycles like Figure 20, 21.

Mesh conditioning and parameterization. The mesh resulting from the flow complex as shown in Figure 10(f) tends to have many sliver triangles. However, this output is amenable to conditioning using edge flips and collapses [Botsch et al. 2010] as can also been seen in Figures 10(f), 20. The output of our algorithm also guarantees intersection-free surfacing of challenging 3D curve input (see Figures 20, 21), which is invaluable in the subsequent paramterization of the surface boundaries. We show the result of a conformal paramterization on our surfaced output in Figure 10(f) that is inherently superior to those produced with a 3D curve alone, or by mapping it to circle [Mehra et al. 2009]. Our approach thus has potential applications in hole filling and other mesh repair tasks.

Limitations. We clearly cannot surface all closed curves with a single intersection-free manifold (see Figure 20). For any closed curve, we produce a small collection of intersection-free manifolds that together interpolate the curve boundary but may share non-manifold edges, even for unknotted curves such as Figure 20(c).

There are also downsides to using the flow complex. Conditioning the curve network for use in our algorithm can be difficult in the presence of nooks and crannies. It can be relatively expensive to compute the flow complex using exact constructions for a fully reliable cell-structure [Cazals et al. 2008], though we have not found this necessary in practiice. A more visible shortcoming is that the flow complex may be geometrically sparser than the Delaunay complex, exemplified in 2D by Figure 22. The blue Gabriel edges in Figure 22(left) are a subset of the grey Delaunay edges. Our algorithm can at best reconstruct the shape in the middle using only Gabriel edges, as opposed to the more suitable construction on the right using Delaunay edges. Pinched corners like Figure 22(middle), where the desired surface element is not even part of the flow complex cannot be addressed by thickening. A hybrid approach that geometrically improves the flow complex based reconstruction using Delaunay elements, is subject to future work.

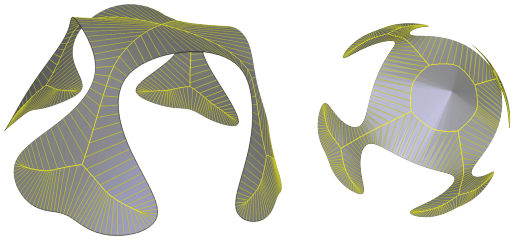


Fig. 21: Surfacing a challenging closed curve (left), with an additional point (right). Yellow lines depict the intersection of the Voronoi complex of P and the surface, visualizing flow from the boundary into *index-2* critical points.

Finally, while motivated by perceptual observations, our approach is purely based on geometry and toplogy. Even though our surfaces match viewer perception well, we make no claim to model the surface loops humans perceive. Instead, we provide guidelines aiding users to provide 3D curve networks that our algorithm will successfully reconstruct. We encountered around 10% failures on sketch-based 3D curve input that did not conform to our guidelines, but note that this was easily remedied by adding curves to the sketch or "bars to the cages".

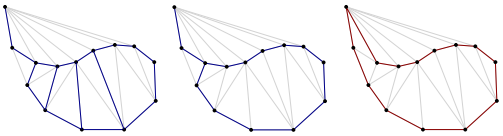


Fig. 22: Gabriel edges in blue are a subset of Delaunay edges in light grey for the given 2D point-set (left). The best flow-complex based shape reconstruction must employ only Gabriel edges (middle). Allowing non-Gabriel Delaunay edges can result in a better shape reconstruction (right).

6. CONCLUSION

We have introduced a a fully automated flow complex based algorithm for the reconstruction of shape from an input collection of 3D curves. The algorithm allows the user to control the number of voids in the output and attempts to capture the remaining topology to the best extent possible. It generates non-self-intersecting meshes that can be used downstream to generate higher quality output. We are also able to surface non-manifold knotted shapes from boundary curves (see Figure 20(d)-(e)). Other aspects of our algorithm, such as the yellow medial-axis like curve on surface shown in Figure 21, are also likely to prove useful for surface parameterization or as handles for shape deformation. We hope that various aspects of the techniques presented in this paper will inspire future work in curve based shape modeling.

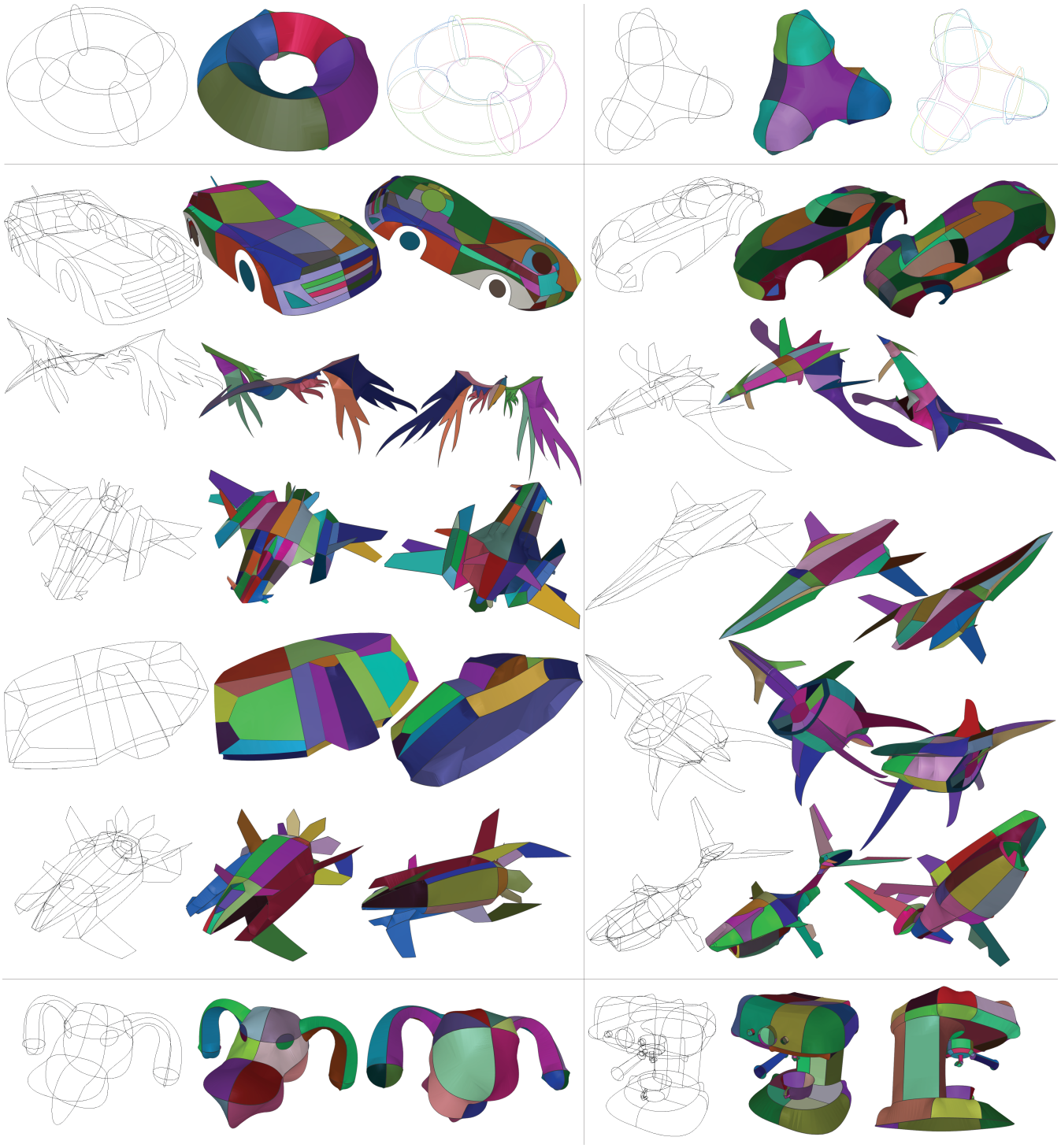


Fig. 23: Reconstruction results: (top row) simple concept curve networks with reconstruction followed by outlines of the patched cycles displayed in different colours; (middle row) examples from the ILoveSketch [Bae et al. 2008] curve network collection, with two different view of each model shown; (bottom row) examples from the analytic drawing tool of [Schmidt et al. 2009].

APPENDIX

Theoretical Justification of Algorithm

We elaborate here on the theoretical bases behind our algorithm. We describe conditions over the input curves which, if satisfied, provably ensure that our algorithm succeeds in reconstructing certain aspects of the target shape. Unlike traditional reconstruction algorithms, which explicitly tie sampling density to various measures of feature size in the target, our characterization of a suitable input is intended to be intuitive and understandable to a designer. We thus continue to phrase the wording of our conditions in terms of the *cage and prisoners* metaphor, introduced in Section 2.

Given the input curve network Γ and the target shape Σ, we call each connected component of $\Sigma - \Gamma$, i.e. each piece obtained by cutting the target shape along the curves of the network, a *target patch*. The boundary of a target patch is a curve cycle in Γ, potentially with inner cycles representing holes (see Figure 5). As all surface detail must be represented by input curves, it is reasonable to assume that a target patch has simple topology, homeomorphic to an open disk, possibly with holes [Bessmeltsev et al. 2012]. Although our algorithm uses a point-set P representation of the curves Γ, we base the following discussion and the proposed sampling conditions on Γ itself and not on P. This is subsequently reconciled using a result on the stability of persistence. We thus employ distance function d_Γ for any point in space to be the distance to the nearest point in Γ. Further, for any $r \geq 0$, let Γ^r denote the set of points within distance r or less from Γ, i.e. $\Gamma^r := \bigcup_{x \in \Gamma} \overline{B}(x, r)$, where $\overline{B}(x, r)$ is the closed ball of radius r centred at x (see Figure 9(left)).

Empty Balls as Prisoners. An *empty ball* is an open ball that does not intersect the curve network Γ. Consider an empty ball $B = B(x, r)$ of centre x and radius r, centred in a target void U. B is empty if $x \notin \Gamma^r$ by definition. In Section 2 terminology, we say that prisoner B can *escape* prison cell U, if we can move B along some path until its centre lies outside U without the ball ever intersecting the prison cage Γ. Equivalently, the connected component of Γ^r that includes x, also includes a point that is not in U. An empty ball B that cannot escape a target void U is *caged in* U. Let $R(U)$ be the radius of the largest empty ball centred in U (prisoner *Fat*), and $r(U)$ the radius of the largest empty ball centred in U that can escape it (prisoner *Thin*). U is *caging* if $R(u) > r(u)$. Note that while U may cage a prisoner of size smaller than $r(U)$, it cannot cage all prisoners of that size. We call an empty ball $B(x, r)$ *safely caged* in U if $x \in U$ and $r > r(U)$. The set of centres of safely-caged balls in a target void U is called its *safe region*.

Our algorithm identifies target voids that must not only be caging but where the range of sizes of safely caged prisoners is large. Formally we say a target shape is (ξ, λ)-*caging*, for constants ξ, λ with $1 < \xi < \sqrt{\lambda}$, if for every bounded target void U:

(1) $R(U) > \lambda r(U)$.
(2) For any two empty balls $B_1 = (x_1, r)$ and $B_2 = (x_2, r)$ with $x_1, x_2 \in U$ and $\xi r(U) < r < R(U)/\xi$, B_1 can be moved along a path to B_2 without intersecting Γ.
(3) For any empty ball $B(x, r)$ with $x \in U$ and $r \leq r(U)$, $B(x, r/\xi)$ can be moved along a path inside some safely caged ball (not necessarily in U) without intersecting Γ.

The first condition requires that prisoner *Fat* be at least a factor λ larger than prisoner *Thin* in U. The second states that the prison cell should be connected for prisoners more than a factor ξ in-between the extreme sizes accommodated. The third condition says that any prisoner too small to be safely caged, can shift its centre to the safe region of some target void, if shrunk by a factor of $1/\xi$.

Note that the definitions of caging curves is scale-invariant. As a sampling condition thus, being (λ, ξ)-caging is an *adaptive* measure and not a uniform one. We can now state a key claim relating to our algorithmic result, when the input curves Γ are (ξ, λ)-caging for the target shape.

LEMMA 1. *If input curves Γ are (ξ, λ)-caging for target shape Σ, then the logarithmic distance function induced by Γ has precisely one local maximum of persistence greater than* $\log \lambda$ *inside each target void. All other maxima have persistences below* $\log \xi$.

The proof sketch of the above statement provided below uses the notion of Poincaré-Lefschetz duality, see e.g. [Cohen-Steiner et al. 2009], which relates topological changes that take place in Γ^r as r passes a critical value to those that happen to its complement, i.e. $\mathbb{R}^3 - \Gamma^r$. When r is varied from zero to ∞, each time r passes the value of a local maximum of d_Γ, a connected component of $\mathbb{R}^3 - \Gamma^r$ disappears (is destroyed). It can be shown that as r passes the value of the creator 2-*cell* that pairs up with this maximum, a connected component of the $\mathbb{R}^3 - \Gamma^r$ is split into two distinct new components. In the dual setting, r is *reduced* from ∞ down to zero. A new connected component is created in $\mathbb{R}^3 - \Gamma^r$, as r passes the function value of the maximum in question, and two connected components merge into one (and therefore one is destroyed) as r passes the function value of the primal creator 2-cell. This turns the primal maximum into a dual minimum and the primal creator 2-*cell* into a dual destroyer 1-*cell*.

PROOF. (*sketch* of Lemma 1) For a bounded void U, the definitions of $r(U)$ and $R(U)$ imply there must be at least one maximum m in U with $d_\Gamma(m) \geq R(U)$. This is because there is a ball of radius of $R(U)$ caged in U and one can follow the flow trajectory from the centre of this ball to a maximum that has to also be in U since d_Γ along the trajectory only grows. Let m be the local maximum in U with the largest distance function value. It can be verified using the duality discussed above that m is paired with the 2-*cell* at which a connected component of $\mathbb{R}^3 - \Gamma^r$ joins the unbounded connected component as r decreases. Satisfying condition (1) in the definition of (ξ, λ)-caging, is equivalent to saying that at $r = r(U)$, the component A of $\mathbb{R}^3 - \Gamma^r$ that includes m is still disjoint from the unbounded one. This ensures that the persistence of m is at least $\log R(U) - \log r(u) \geq \log \lambda$. Condition (2) ensures that any connected component subsequently separated from A ends up either with a maximum of the distance function at $r < \xi r(U)$ and thus has persistence at most $\log \xi < \log \lambda$ or is split at $r > R(U)/\xi$, which again limits the persistence of the maximum in the component to be no more than $\log \xi$. Finally, condition (3) ensures that any maximum contained in U at distance smaller than $r(U)$ to Γ also ends up with persistence below $\log \xi$. Thus m is going to be the only maximum in U that ends up with a persistence greater than or equal to $\log \lambda$. □

The above Lemma explains the rationale behind our picking the β_2^* maxima with the highest persistences and creating corresponding voids for them in our reconstruction by including their creator 2-*cells*.

Substituting Γ with P. Although P densely samples Γ, the sequence of topological changes that occur in P^r as r increases

can be vastly different from that of Γ^r. An important result of [Cohen-Steiner et al. 2007] however, proves that events (critical points) with persistence values greater than a threshold determined by the largest disparity between $\log d_P$ and $\log d_\Gamma$ are essentially in one-to-one correspondence. More precisely, for two real-valued functions f and g over a space X, critical points of f can be put in correspondence with those of g in such a way that if x is a critical point of f and y its matching critical point in g, then both the difference between $f(x)$ and $g(y)$ and the difference between the persistence values of x (under f) and y (under g) are no more than $\|f-g\|_\infty = \sup_x |f(x)-g(x)|$.

In our case, $f(x) = \log d_P(x)$ and $g(x) = \log d_\Gamma(x)$. It can be easily observed that if X is taken as the whole of $\mathbb{R}^3$, then $\|f-g\|_\infty$ will be infinite and of no use for us. However, if we exclude from X points within a suitably small distance from Γ, and ensure that P samples Γ sufficiently finely, then a bound can be placed on $\|f-g\|_\infty$ as described by the following lemma.

LEMMA 2. *Let X be $\mathbb{R}^3 - \Gamma^\varepsilon$ for some $\varepsilon > 0$. Let P sample Γ in such a way that each point of Γ has a point of P within distance δ. Then if Γ is (λ,ξ)-caging for the target shape Σ, each target void of Σ will include precisely one maximum of f of persistence larger than $\log\sqrt{\lambda\xi}$ inside X provided that $(1+\delta/\varepsilon) < \sqrt{\lambda/\xi}$.*

PROOF. Under the assumed sampling density, for each point $x \in X$ one has: $d_\Gamma(x) \le d_P(x) \le d_\Gamma(x) + \delta$, from which, together with the fact that $d_\Gamma(x) > \varepsilon$ we get

$$d_P(x) \le \left(1 + \frac{\delta}{\varepsilon}\right) d_\Gamma(x),$$

or equivalently $f(x) - g(x) \le \log\left(1+\frac{\delta}{\varepsilon}\right)$. Since $f(x) \ge g(x)$ for any x, we conclude that $\|f-g\|_\infty \le \log\left(1+\frac{\delta}{\varepsilon}\right)$.

Thus if $1+\delta/\varepsilon < \sqrt{\lambda/\xi}$ we get $\|f-g\|_\infty < \log\sqrt{\lambda/\xi}$. By the above-mentioned stability result, when Γ is (ξ,λ)-caging, if $\|f-g\|_\infty \le \frac{1}{2}(\log\lambda - \log\xi) = \log\sqrt{\lambda/\xi}$ over X, then the persistence of a critical point of f differs by no more than $\log\sqrt{\lambda/\xi}$ from the persistence of its matching critical point of g. Consequently, the separation between persistence values of the largest persistence maximum of each void (greater than $\log\lambda$) and those of other maxima in that void (no greater than $\log\xi$) demonstrated by Lemma 1 would imply a separation for the critical points of P by which each target void of Σ would have precisely one maximum of persistence greater than $\log\lambda - \log\sqrt{\lambda/\xi} = \log\sqrt{\lambda\xi}$. □

In essence, the parameter ε used in the statement of the above lemma sets a hard lower-bound on the smallest size of a prisoner for all target voids: The definition of a (λ,ξ)-caging curve network Γ is scale-invariant but if Γ is to be approximated by a discrete sample P that is at a Hausdorff distance of δ from Γ, then one can not afford to cage target voids whose largest caged prisoner has size smaller than ε as determined by the above Lemma.

From the argument in the proof of Lemma 1, we can further see that for any target void U, any point $x \in U$ with $d_\Gamma(x) > r(U)$, i.e. the centre of an empty ball of radius $r(U)$ safely caged in U, will have to be contained in the reconstructed void $\hat{U}$ that corresponds to U. This is because these points are contained in the same connected component of $\mathbb{R}^3 - \Gamma^{r(U)}$ as the highest persistence maximum in U and thus all flow into critical points in this set, which is not further subdivided by the algorithm since all creator *2-cells* that can subdivide it have persistence lower than the required threshold to be considered. Taking the sampling of Γ by P into account, we can further provide the following additional guarantee statement, which essentially states that except for points very close the cage, reconstructed voids agree with target voids:

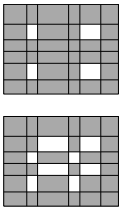
Let Γ be (λ,ξ)-caging for the target shape Σ and let P and δ be as defined in Lemma 2, then for any target void U, the subset of U consisting of points x with $d_P(x) > r(U) + \delta$, are contained in the reconstruction $\hat{U}$ produced by TOPOLOGICALRECONSTRUCTION.

Handling undersampling. When Γ is under-sampled, it is possible to get tiny prison cells with higher persistence than the desired target voids, resulting in wrong set of target voids in the reconstruction. Algorithmic efficiency notwithstanding, it is thus important to sample the input curve densely and increase the sampling density if the reconstruction seems to fail. Additionally, when selecting the β_2^* creator 2-cells with the highest persistence, one might ignore those with distance function values smaller than a threshold determined by the sampling density of the curves. Interactive user selection from a set of candidate voids is also a viable option, given the typically small number of target voids.

First Betti Number. Unlike β_0^* and β_2^* that we prefer to be designer specified, we attempt to infer β_1^* automatically from the flow complex, given the other Betti numbers. We use the insight that voids in target 3D models are typically disjoint in that they are not bounded in part by a common surface. This removes any internal walls that might block handles from forming. Similarly, while fins that interpolate input curves are common in design objects, a desired surface patch that blocks a topological tunnel or hole is rare. We thus attempt in Section 4.3 to maximize β_1^* by unblocking as many holes and handles as possible, while preserving β_2^*,

Let us consider first the case where a target void U has no input curves in its interior. Let m be the maximum with highest persistence in U and let c be its creator *2-cell* in the reconstruction; thus $d_P(c) \le r(U)$. The introduction of c to the filtration splits a previous void into two, one of which contains m. Let us call this void at this stage $\tilde{U}$. Topologically, $\tilde{U}$ is identical to the connected component of $\mathbb{R}^3 - \Gamma^{r(U)}$ containing m. As the filtration progresses, this component may shrink with the arrival of cells contained in $\tilde{U}$:

An arriving creator *2-cell* in U break it into distinct voids. Since we discard any such creator due to low persistence from our reconstruction, we can assume $\tilde{U}$ stays connected for the rest of the filtration. An arriving destroyer *2-cell* in U blocks a tunnel but keeps the void connected. Thus at the end of line 3 of the TOPOLOGICALRECONSTRUCTION algorithm, K_0 comprises the void $\tilde{U}$ with some of its tunnels blocked by destroyer *2-cells*. Deleting *all* these destroyers, reopens all the tunnels of $\tilde{U}$.



Now consider a target shape with input curves interior to a target void. As described above we expect such design curves to represent internal fins rather than walls that block tunnels. Imagine such interior curves attached to the inside of a void along the inset rectangular boundary. We show at least two different ways to create holes while keeping all curve segments incident to at least one *2-cell* (a fin). We protect such destroyer *2-cells* that will surface internal fins, by only considering those that survive a *1-collapse* for removal (remember that a *1-collapse* removes all fins). Among the destroyer *2-cells* that can be removed we repeatedly remove the one with the largest d_P unblocking the tunnel that closed last in the filtration.

Glossary of Terms

We provide here a quick reference to terms used by our algorithm in Section 4. The description below is informal and specific to this paper, with pointers to more accurate and complete definitions.

—*i-cell*: An i-D cell of a cell complex. For a 3D flow complex an *i-cell* is a point, a Gabriel edge, a triangulated surface patch, and a connected volume bounded by *2-cells*, for $i = 0, 1, 2, 3$ respectively (see Section 2.1). See also Section 3 (simplicial homology) for definitions of *i-boundary*, *i-chain* and *i-cycle*.

—1-*collapse*: A "free" edge is incident to a single 2-*cell* in a cell complex, and the removal of both constitutes a 1-*collapse* operation. A 1-*collapse* of a cell complex is the successive removal of all free edges and their incident 2-*cells* from the complex untill no free edges remain. A 2-*collapse* similarly removes "free" 2-*cells* along with their adjacent 3-*cells* (see Section 3(collapses)).

—1_{res}-*collapse*: A 1-*collapse* where edges representing input curve segments are not collapsed (see Section 3(collapses)).

—*Betti numbers*: Topological descriptors of a cell complex. In 3D, β_0, β_1, and β_2 count the number of connected components, holes and tunnels, and cavities of a shape, respectively.(see Section 3(simplicial homology)).

—*Creators and Destroyers*: Every 2-*cell* when added to the flow complex either destroys its 1-*cycle* of bounding edges or creates a new 2-*cycle* by enclosing a void. Every *i-cell* is similarly a destroyer or a creator (see Section 3(filtrations and persistence)).

—*Filtration*: A total ordering of cells in a complex where cells appear after all their faces, such as ordering flow complex elements by distance function (see Section 3(filtrations and persistence)).

—*Gabriel edge*: An edge in a cell complex whose diametric ball contains no points (see Section 2.1(point inflation)).

—*Persistence*: Interval between the creation and destruction of cell complex elements (see Section 3(filtrations and persistence)).

REFERENCES

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

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *IEEE Visualization*.

AMENTA, N., CHOI, S., AND KOLLURI, R. K. 2001. The power crust, unions of balls, and the medial axis transform. *Comput. Geom. Theory Appl. 19,* 2-3, 127–153.

BAE, S.-H., BALAKRISHNAN, R., AND SINGH, K. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. UIST '08. ACM, New York, NY, USA, 151–160.

BESSMELTSEV, M., WANG, C., SHEFFER, A., AND SINGH, K. 2012. Design-driven quadrangulation of closed 3d curves. *SIGGRAPH Asia*.

BOISSONNAT, J.-D. 1984. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph. 3,* 4, 266–286.

BOISSONNAT, J.-D. AND CAZALS, F. 2002. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Comput. Geom. Theory Appl. 22,* 1-3, 185–203.

BOTSCH, M., KOBBELT, L., PAULY, M., ALLIEZ, P., AND UNO LEVY, B. 2010. *Polygon Mesh Processing*. AK Peters.

BUCHIN, K., DEY, T. K., GIESEN, J., AND AND, M. J. 2008. Recursive geometry of the flow complex and topology of the flow complex filtration. *Comput. Geom. Theory Appl. 40*, 115–157.

CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proc. SIGGRAPH 2001*. 67–76.

CAZALS, F., PARAMESWARAN, A., AND PION, S. 2008. Robust construction of the three-dimensional flow complex. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*. Number 5903. 182–191.

CHAINE, R. 2003. A geometric-based convection approach of 3D reconstruction. In *Proc. 1st Sympos. Geom. Proc.* 218–229.

COHEN-STEINER, D., EDELSBRUNNER, H., AND HARER, J. 2007. Stability of persistence diagrams. *Discrete & Computational Geometry 37,* 1, 103–120.

COHEN-STEINER, D., EDELSBRUNNER, H., AND HARER, J. 2009. Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math. 9*, 79–103.

DEY, T. 2011. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press.

DEY, T. K., GIESEN, J., RAMOS, E. A., AND SADRI, B. 2005. Critical points of the distance to an epsilon-sampling of a surface and flow-complex-based surface reconstruction. In *Proc. 21st Annu. ACM Sympos. Comput. Geom.* 218–227.

EDELSBRUNNER, H. 2004. Surface reconstruction by wrapping finite point sets in space. *Discrete & Computational Geometry 32*, 231–244.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002a. Topological persistence and simplification. *Discrete & Computational Geometry 28,* 4, 511–533. (Invited).

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002b. Topological persistence and simplification. *Discrete & Computational Geometry 28,* 4, 511–533. (Invited).

GIESEN, J. AND JOHN, M. 2002. Surface reconstruction based on a dynamical system. *Computer Graphics Forume 21*, 363–371.

GIESEN, J. AND JOHN, M. 2003. The flow complex: A data structure for geometric modeling. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*. 285–294.

GROSSMAN, T., BALAKRISHNAN, R., AND SINGH, K. 2003. An interface for creating and manipulating curves using a high degree-of-freedom curve input device. In *Proceedings of SIGCHI*. CHI '03. ACM, 185–192.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J. A., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. In *Proc. SIGGRAPH '92*. 71–78.

MEHRA, R., ZHOU, Q., LONG, J., SHEFFER, A., GOOCH, A., AND MITRA, N. J. 2009. Abstraction of man-made shapes. *ACM Transactions on Graphics 28,* 5, #137, 1–10.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. In *ACM SIGGRAPH 2007 papers*. SIGGRAPH '07. ACM, New York, NY, USA.

OLSEN, L., SAMAVATI, F., SOUSA, M., AND JORGE, J. 2009. Sketch-based modeling: A survey. *Computers & Graphics 33*, 85–103.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Transactions on Graphics 28,* 5, (to appear). Proceedings of SIGGRAPH ASIA 2009.

SHPITALNI, M. AND LIPSON, H. 1996. Identification of faces in a 2d line drawing projection of a wireframe object. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 1000–1012.

SINGH, K. 2006. Industrial motivation for interactive shape modeling: a case study in conceptual automotive design. In *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. ACM, New York, NY, USA, 3–9.

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