SurfaceBrush: From Virtual Reality Drawings to Manifold Surfaces

ENRIQUE ROSALES, University of British Columbia, Canada and Universidad Panamericana, México
JAFET RODRIGUEZ, Universidad Panamericana, México
ALLA SHEFFER, University of British Columbia, Canada

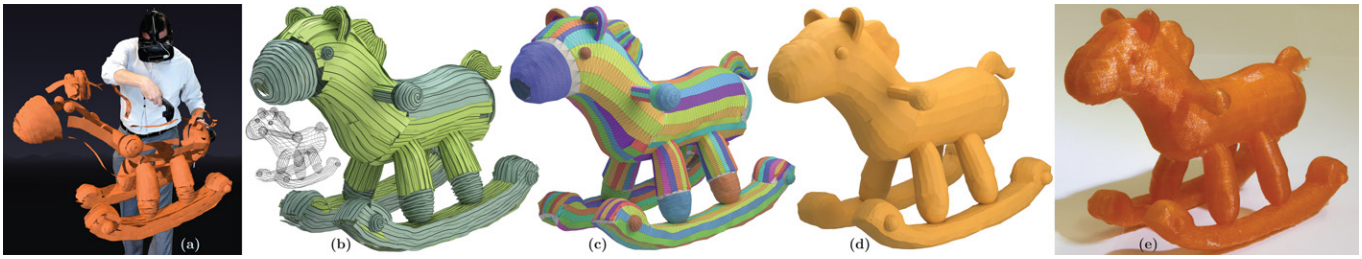


Fig. 1. (a) Drawing 3D strokes using a VR brush. (b) Completed 3D brush-stroke drawing with central stroke polylines drawn in black and ribbon color reflecting normal orientation (green for front, turquoise for back, inset shows poylines alone). (c) triangle mesh strips connecting adjacent stroke polylines (multicolor), and gray triangle strips that complete the surface connecting differently directed stroke groups. (d) Final output. (e) Fabricated model. Input drawing: © Jafet Rodriguez.

Popular Virtual Reality (VR) tools allow users to draw varying-width, ribbon-like 3D brush strokes by moving a hand-held controller in 3D space. Artists frequently use dense collections of such strokes to draw virtual 3D shapes. We propose *SurfaceBrush*, a surfacing method that converts such VR drawings into user-intended manifold free-form 3D surfaces, providing a novel approach for modeling 3D shapes. The inputs to our method consist of dense collections of artist-drawn stroke ribbons described by the positions and normals of their central polylines, and ribbon widths. These inputs are highly distinct from those handled by existing surfacing frameworks and exhibit different sparsity and error patterns, necessitating a novel surfacing approach. We surface the input stroke drawings by identifying and leveraging local coherence between nearby artist strokes. In particular, we observe that strokes intended to be adjacent on the artist imagined surface often have similar tangent directions along their respective polylines. We leverage this local stroke direction consistency by casting the computation of the user-intended manifold surface as a constrained matching problem on stroke polyline vertices and edges. We first detect and smoothly connect adjacent similarly-directed sequences of stroke edges producing one or more manifold partial surfaces. We then complete the surfacing process by identifying and connecting adjacent similarly directed edges along the borders of these partial surfaces. We confirm the usability of the SurfaceBrush interface and the validity of our drawing analysis via an observational study. We validate our stroke surfacing algorithm by demonstrating an array of manifold surfaces computed by our framework starting from a range of inputs of varying complexity, and by comparing our outputs to reconstructions computed using alternative means.

CCS Concepts: • **Computing methodologies → Mesh geometry models**; *Virtual reality*.

Authors' addresses: Enrique Rosales, University of British Columbia, Canada, albertr@cs.ubc.ca, Universidad Panamericana, Facultad de Ingeniería, Zapopan, Jalisco, 45010, México; Jafet Rodriguez, Universidad Panamericana, Facultad de Ingeniería, Zapopan, Jalisco, 45010, México, arodrig@up.edu.mx; Alla Sheffer, University of British Columbia, Canada, sheffa@cs.ubc.ca.

Additional Key Words and Phrases: Virtual Reality, 3D drawing, surface modeling, surface reconstruction

ACM Reference Format:
Enrique Rosales, Jafet Rodriguez, and Alla Sheffer. 2019. SurfaceBrush: From Virtual Reality Drawings to Manifold Surfaces. *ACM Trans. Graph.* 38, 4, Article 96 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3322970

1 INTRODUCTION

Humans frequently communicate 3D shapes via 2D sketches or drawings, inspiring the development of modeling interfaces that employ such drawings as inputs [Olsen et al. 2009]. Virtual Reality (VR) systems support real-time capture and visualization of human 3D gestures enabling users to draw surfaces directly in 3D space (Figure 1a). Using such drawings as input for 3D modeling can sidestep the main algorithmic challenge of 2D sketch-based modeling methods – the need to derive 3D information from a 2D input. Effectively leveraging the opportunity provided by VR interfaces requires modeling frameworks capable of processing the types of 3D drawings users can comfortably provide using these tools. Our observations show that artists using the VR medium frequently depict complex free-form 3D geometries using collections of dense, ruled surface *brush strokes* traced in 3D space (Figure 1b) [Sketchfab 2018]. Our *SurfaceBrush* framework algorithmically converts VR brush stroke drawings into manifold surface meshes describing the user-intended geometry (Figure 1d), enabling downstream applications such as 3D printing (Figure 1e).

Users of VR systems, such as TiltBrush [2018] or Drawing on Air [Keefe et al. 2007], trace strokes using a handheld controller. These systems then automatically translate controller locations into polyline stroke vertex positions and controller orientations into stroke normals. They subsequently render the captured input as virtual ribbons, or ruled surface strips, of user-specified width (Figure 1a). The rendered ribbons are centered around the captured stroke polyline positions and their orientation reflects the captured

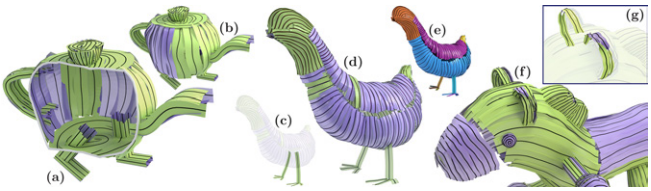


Fig. 2. Brush stroke drawings are characterized by strokes with partially overlapping ribbons, and locally similar stroke tangent directions (b,d,f). These directions change abruptly between different surface regions (e); stroke normal orientations are often inconsistent (front facing ribbons rendered in green, back facing in purple) (b,d,f), inexact and sometimes erroneous (g); stroke groups frequently intersect "inside" the model (see cutout view) (a); and drawings occasionally contain isolated strokes (c). Teapot and horse: © Jafet Rodriguez, chicken: © Elinor Palomares.

stroke normals. Our experiments show that both artists and non-experts can easily, quickly, and effectively communicate their envisioned 3D surfaces using this interface by drawing dense brush strokes that cover the surface of the intended shapes (Section 7).

Adopting this interface for surface modeling necessitates algorithmically reconstructing the user-intended 3D surfaces from the dense set of brush strokes drawn by the users. Each stroke is defined by the vertex positions and normals along its central polyline and has an associated width. This input format is distinctly different from those processed by existing surface reconstruction methodologies, and exhibits different error and sparsity patterns (Section 3). In particular, artist drawings (see e.g. Figure 2) have inconsistent stroke normal orientations and partially overlapping strokes; they frequently contain intersecting stroke groups and may exhibit isolated outlier strokes. Due to these artifacts, existing surfacing methods are inadequate for our needs (Section 2). In particular, using polyline vertices or densely sampled points on the ribbons as input to methods for reconstruction from point clouds fails to produce the desired surfaces (Figure 3).

SurfaceBrush reconstructs an intended surface from the input brush strokes by interpolating sequences of edges along the stroke polylines. It determines the edges to include and the connectivity between them by leveraging local consistency between the drawn strokes (Section 3). The key observation it utilizes is that, when depicting 3D shapes using a VR brush, users typically adopt a strategy that resembles the action sequence commonly used when applying top paint to 3D objects using a paint brush. Specifically, users often draw contiguous patches on the target surface using side-by-side strokes with similar tangent directions and change stroke directions when switching between different parts of the drawing (Figure 2). These observations argue for a surfacing strategy that prioritizes connections between side-by-side strokes with similar tangents. Following this argument, SurfaceBrush computes the output surface using a two-step process (Section 4). First, it forms *inter-stroke* mesh strips, by detecting and connecting side-by-side stroke sections, or sequences of edges (Figure 1c, multicolor, Section 5). Then, it closes the gaps between the partial surfaces consisting of a union of such strips by connecting adjacent sections along their boundaries (Figure 1c, gray, Section 6). The core challenge in employing this strip-based surfacing approach is to identify, or to match, the best stroke sections to connect in the first stage of the process and the best boundary sections to connect in the second. This challenge is augmented by our goal of producing *manifold* output surfaces while overcoming artifacts present in the data (Figure 2).

We formulate both matching problems using a discrete constrained optimization framework (Section 5). We efficiently solve them by first relaxing the manifoldness constraints, obtaining locally optimal (but not necessarily globally compatible) vertex-to-vertex matches (Sections 5.1, 5.2). We use the obtained vertex-to-vertex matches to identify corresponding stroke sections and to connect these sections using triangle strips (Sections 5.3). We eliminate non-manifold artifacts in this mesh using a correlation clustering framework that determines which triangles should remain in the mesh and which should be removed (Section 5.4). This process robustly reconstructs user-intended, manifold surfaces from complex drawings, such as the horse (Figure 1, 298 strokes, 20K vertices) in under a minute.

We validate the SurfaceBrush modeling framework by evaluating both our choice of inputs and the method we propose for processing those. We conduct a user study which confirms that experts and non-experts alike can effectively use brush strokes to visually communicate free-form surfaces in a VR environment, and validates our observation about users preference for depicting surfaces using a set of patches drawn using similarly directed strokes (Section 7). We confirm the robustness of the SurfaceBrush surfacing algorithm by demonstrating a range of reconstructed surfaces created from inputs of different complexity produced by artist and amateur users and compare the results to those produced by state of the art alternatives (Section 8). These experiments confirm that, while our outputs are consistent with the artist-intended surface geometry, the results of alternative methods are not.

Our overall contribution is a new VR drawing-based modeling framework that allows experts and amateurs alike to create complex free-form 3D models via an easy-to-use interface. The technical core of our system is a new surfacing algorithm specifically designed to reconstruct user-intended manifold surfaces from dense ribbon-format 3D brush strokes. This contribution is made possible by our detailed analysis of brush drawing characteristics.

2 PREVIOUS WORK

Our work builds upon prior research across multiple domains.

2D-Sketch-Based Modeling. 2D-sketch-based modeling methods infer depth information from collections of sparse 2D artist strokes, which are assumed to employ a specific drawing style and capture key properties of the artist-intended shape [Olsen et al. 2009]. SurfaceBrush recovers surface geometry from dense 3D strokes, an input that exhibits very different properties. Algorithms that process 2D drawings frequently leverage established drawing conventions and observations about human perception of 2D imagery [Bae et al. 2008; Li et al. 2017; Nealen et al. 2007; Schmidt et al. 2009; Shao et al. 2012; Xu et al. 2014]. Such resources are essentially non-existent for 3D drawings, since until recently there had been few opportunities for artists to use 3D strokes to depict shape. We derive the characteristics of the inputs we seek to process via examination of publicly available VR artwork databases [Poly 2018; Sketchfab 2018] and an observational study of VR 3D shape drawing (Sections 3, 7).

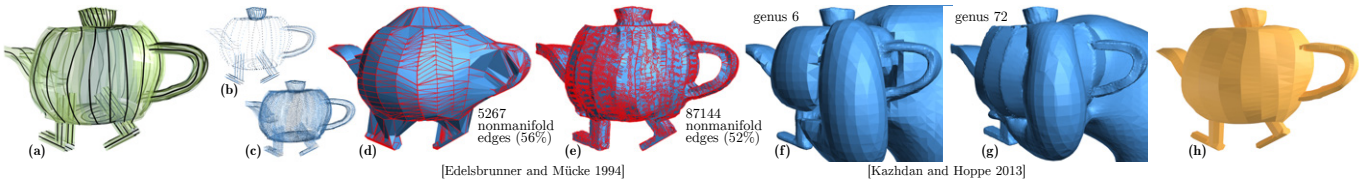


Fig. 3. Treating strokes' (a) polyline vertices as unorganized points with normals (b) and using those as input to state of the art reconstruction from point-clouds methods (d,f) produces inadequate results with multiple artifacts, such as high percentages of non-manifold edges [Edelsbrunner and Mücke 1994] (d), unnecessarily high genus, and arbitrary deviation from the input [Kazhdan and Hoppe 2013] (f). Densely sampling points along the ruled ribbons (c) and using these samples plus normals as reconstruction input produces surfaces which exhibit similar artifacts (e,g). Our output (h) accurately captures user's intent. Input drawing: © Jafet Rodriguez.

Sketch Consolidation. Our work has conceptual similarities to sketch consolidation [Fu et al. 2011; Liu et al. 2018, 2015; Noris et al. 2012; Stahovich et al. 2014; Xing et al. 2014]. However, in 2D each stroke vertex has unique nearest left/right neighbors along the stroke's orthogonal. This property no longer holds in 3D, making determination of best pairwise vertex matches a lot more challenging.

3D Curve Drawing. Researchers have proposed a range of tools for creating, rendering, and manipulating curves directly in 3D space [Amores and Lanier 2017; Diehl et al. 2004; Grossman et al. 2002; Israel et al. 2009; Jackson and Keefe 2016; Kim et al. 2018; Tano et al. 2003]. Recent systems render captured curves in real time using head-mounted displays, depicting them as ruled surface ribbons [Keefe et al. 2007, 2001; TiltBrush 2018] or as tubular shapes with cylindrical profiles [Keefe et al. 2007; PaintLab 2018]. Utilizing the content artists produce using such systems for shape modeling requires converting raw curve drawings into 3D surface models. SurfaceBrush achieves this goal using as input oriented ribbon strokes created with the widely available TiltBrush system; it can also be employed in conjunction with other VR systems which support such strokes.

VR Modeling Interfaces. Researchers have explored a range of VR modeling interfaces. VR sculpting tools [Kodon 2018; OculusMedium 2016; ShapeLab 2018] allow expert users to create sophisticated shapes. VR interfaces that support Boolean operations over a fixed set of primitives [DesignSpace 2018; Diehl et al. 2004; GoogleBlocks 2018; Tano et al. 2013] provide a promising avenue for modeling CAD geometries but are not well suited for free-form shapes. Others enable users to draw a range of swept surfaces in 3D space [GravitySketch 2018; Keefe et al. 2001; Schkolne et al. 2001; Schkolne and Schroeder 1999]. To model complex shapes using this approach, users need to mentally break them into coarse, non-overlapping, sweepable patches, and separately draw each patch: a task that requires modeling expertise and is especially challenging for organic shapes. Several VR systems facilitate editing of existing 3D surfaces [GravitySketch 2018; Kwon et al. 2005; Wesche and Seidel 2001]. Our work complements all those systems in its focus on providing experts and amateurs alike with the means to author free-form manifold geometries which they can later edit.

Several VR interfaces allow users to connect 3D curves into cycles or curve networks [Fiorentino et al. 2002; Jackson and Keefe 2016; Kwon et al. 2005; Wesche and Seidel 2001] and provide them with the option to surface those inputs using traditional cycle and network surfacing techniques, such as Coons patches or NURBs. The obtained surfaces are highly dependent on the choice of the surfacing method. Grossman [2002] and Sachs [1991] facilitate tracing of characteristic surface curves such as flow lines in a VR environment. Networks consisting of such curves can be surfaced using designated algorithms [Bessmeltsev et al. 2012; Pan et al. 2015]. Employing any of these systems users need to understand the underlying surfacing method in order to draw the curves that would form their desired output. Our approach does not require such understanding and does not constrain users to modeling particular surface families. As such we add another tool to the VR modeling palette, one specifically suited for non-expert users and generic free-form geometries.

Our choice of using dense ribbon strokes as modeling input is inspired by the method of Schkolne et al. [2001; 1999] which forms free-form surfaces by merging adjacent swept surfaces drawn by artists. Schkolne et al. generate the merged surface using a method that is designed to provide a real-time approximation of point-cloud reconstruction techniques such as Alpha-Shapes [Edelsbrunner and Mücke 1994], Figure 3. As the authors acknowledge, even on the relatively clean data they tested, the method frequently produces non-manifold geometries.

Surface Reconstruction from Curves, Point Clouds, and Triangle Soups. Research on surface reconstruction from curves targets specific input sources and leverages their distinct properties. Many methods address reconstruction from closed, planar cross-section curves, e.g [Huang et al. 2017; Sharma and Agarwal 2016; Zou et al. 2015]. Others address lofting, or surfacing of closed curve cycles [Finch and Hoppe 2011; Gao and Rockwood 2005; Nasri et al. 2009; Schaefer et al. 2004; Várady et al. 2011] and networks [Abbasinejad et al. 2012; Bessmeltsev et al. 2012; Grimm and Joshi 2012; Pan et al. 2015; Wang et al. 2016]. Our inputs do not conform to the assumptions employed by any of these methods: the strokes are not closed, are frequently non-planar, and do not form cycles or networks. They thus require a different set of priors for successful surfacing. Usumezbas et al. [2017] use curves on the surface of the output models while utilizing image data to filter out poor surfacing choices based on occlusions; we must process curves that extend inside the intended shapes, making occlusion a problematic criterion.

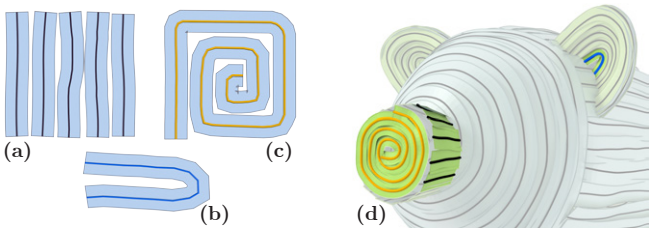


Fig. 5. Artists use different stroke patterns with locally similar tangents to cover, or paint, different surface regions: (a-c) frequent schematic patterns, (d) portion of a typical drawing where all three patterns are used. Piggy bank: © Elinor Palomares.

Methods for surface reconstruction from point clouds [Berger et al. 2017] can potentially be applied as-is to stroke polyline vertices or to a dense set of points sampled along the stroke ribbons (Figure 3 (b,c)). However sampling brush stroke drawings (Figure 2) produces point clouds with inconsistent normal orientation, multiple samples in the interior of the intended shape, and other artifacts inconsistent with the assumptions made by typical reconstruction techniques [Berger et al. 2017]. Moreover, while stroke vertex locations are typically reflective of the intended surface location, due to the inaccuracy in the stroke normals, points sampled along the ribbons are often misplaced with respect to this surface (Figure 4). These artifacts cause traditional reconstruction methods, such as [Avron et al. 2010; Bernardini et al. 1999; Edelsbrunner and Mücke 1994; Kazhdan and Hoppe 2013; Wang et al. 2016; Xiong et al. 2014] to fail dramatically (Figure 3, Section 8).

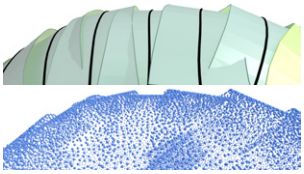


Fig. 4. Close-up of ribbons and ribbon samples on the teapot

One could potentially treat the brush strokes as triangle strips, and use methods for triangle soup surfacing and repair to attempt to recover the 3D shapes from them. However, voxel based methods, e.g. [Ju 2004; Shen et al. 2004], are only applicable to inputs one expects to be closed. Roughly one quarter of our inputs have some open surface elements (e.g. ground on the bonsai or feet on the chicken). Even on closed surfaces, these methods fail drastically in terms of the topology and geometry of the results produced (Section 8). Winding-number based approaches [Barill et al. 2018] produce similar artifacts (Section 8).

3 INPUT DRAWING CHARACTERISTICS

Analysis of publicly available VR artwork [Poly 2018; Sketchfab 2018] and observation of VR 3D shape drawings created by our study participants (Section 7) point to a number of core common characteristics of 3D brush-stroke drawings.

Dense coverage: In both datasets, the drawn stroke ribbons frequently overlap and typically densely cover the communicated shapes leaving relatively small inter-stroke gaps or holes whose size is typically smaller than the width of the surrounding strokes (e.g. Figure 1). The stroke width users employ varies across different parts of the surface, and is typically more narrow on finer features.

Local tangent consistency: Artists frequently draw contiguous surface patches using strokes with similar tangent directions (Figures 2, 5). Tangent consistency is local rather than global since artist often use very different stroke directions in different parts of the model: they choose stroke directions based on drawing convenience and often align stokes with the minimal absolute curvature directions on the intended surface.

Persistent adjacency: Artists use a range of drawing strategies when forming tangent-consistent stroke patches: they may use multiple side-by-side strokes, draw sharply-turning self-adjacent strokes, or use long self-adjacent spirals (Figure 5). The strategy may often vary across a single input. Adjacent side-by-side tangent consistent strokes typically have comparable lengths. Consequently most strokes have only a few, and often just one, immediately adjacent, similarly-directed strokes on each side.

Normal orientation: The input stroke normals are defined by the orientation of the hand-held controller. Users typically aim for the stroke ribbons to lie in the tangent plane of the intended surface, thus the stroke normals are typically roughly orthogonal to this surface (Figures 1, 2) but are rarely exact. VR systems use double-sided ribbon rendering, which obscures stroke orientations from artists. Consequently, we observe that artists do not attempt any in-out consistency, producing strokes whose orientation is essentially a function of drawing access convenience. Specifically, users typically hold the controller like a brush, with its tip pointing away from them, resulting in stroke normals that point toward the artist much of the time. Consequently, normal direction is determined by the location of the artist relative to the drawn shape and is typically *not* reflective of the surface front-back orientation (Figure 2). As this figure illustrates, normal mis-orientation is a persistent feature. Approximately one-third of the strokes in our inputs are oriented in the opposite direction to the plurality. This ratio holds across artists and input categories. Thus orientation inconsistency must be addressed by any method processing VR-brush strokes.

Intersecting and isolated strokes: When drawing different model parts, artists rely on the what-you-see-is-what-you-get principle and assume that making strokes or portions of strokes not visible from outside the object is tantamount to erasing them. Thus, when drawing different parts of the target shape, they often extend strokes into the interior of the models producing multiple intersecting stroke groups (Figures 2, 8) and do not erase occluded outlier strokes. Both existing artwork and 3D drawings created at our bequest often use sparse, isolated, strokes for communicating one-dimensional or very narrow geometries (such as the chicken feet in Figure 2). We speculate that this choice reflects the difficulty of accurately depicting outer surfaces of narrow features and leverages the fact that human observers can easily parse such abstracted or skeletal elements.

Stroke Accuracy: Lastly, we note that users aim to accurately communicate the envisioned shape, thus the shape and location of most stroke polylines typically reflects the intended surface geometry along them up to low-frequency noise inevitable when drawing 3D content by hand. Note that the accuracy of any point on the ribbons away from the polyline depends on both the accuracy of

the stroke normals and the underlying surface curvature - when the curvature in the direction orthogonal to the strokes is large, even with perfect normals, ribbon sides can significantly deviate from the surface (Figure 4).

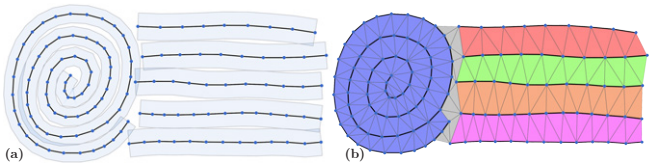


Fig. 6. Schematic surfacing illustration: (a) input strokes, (b) output mesh consisting of inter-stroke mesh strips bounded by adjacent stroke polylines (multicolor) computed first, and gap spanning strips (gray) computed later.

4 OVERVIEW

The SurfaceBrush framework is designed to operate in conjunction with existing 3D stroke drawing tools and to process as input completed ribbon stroke drawings depicting manifold, open or closed, surfaces. It converts these drawings into manifold surface meshes describing the artist-intended geometry (Figure 7).

4.1 Surfacing Goals

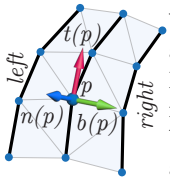
The observations about the key properties of 3D VR drawings (Section 3) allow us to formulate our algorithm's goals in terms of producing outputs consistent with user expectations.

Interpolation and Normal Consistency: While we expect some strokes or stroke sections to be outliers, we expect most stroke edges to be part of the target surface and to accurately depict its location. Thus we expect the reconstructed surface to *interpolate* the vast majority of stroke edges and expect this surface to be roughly orthogonal to the interpolated stroke vertex normals.

Union of Inter-Stroke Strips: Combining these expectations with observations about tangent consistency and persistence we argue that users expect the interpolating surface to be dominated by *surface strips* connecting, or *bounded* by, side-by-side stroke sections with similar tangent directions (Figure 6b, multicolor). Each such strip consists of a sequence of triangles where each triangle shares one edge with its predecessor. Jointly, these strips form one or more *partial surfaces* interpolating most stroke vertices and edges (Figure 6b has two such surfaces separated by the gray strip). Since we expect the adjacencies between strokes to be persistent, we expect the number of different strips bounded by each given stroke to be small (frequently just one on the left and one on the right).

Gap Closure: We expect the final surface to connect the partial surfaces closing the gaps between them. As before, we expect these *gap-spanning* surface strips (Figure 6b gray) to connect close-by boundary vertices and to be orthogonal to the partial surface normals at these vertices.

Manifoldness: To satisfy manifoldness, each stroke section must bound at most two surface strips. Since we expect the stroke normals to be orthogonal to both strips and expect the output surface to be fair, one of these strips should be on the right and the other on the left of the section with respect to the local Frenet frame defined by the stroke tangent and normal (see inset).



We expect sections along the partial surface boundaries to bound at most one gap-spanning strip located on the opposite side of the partial surface with respect to a Frenet frame defined by the boundary tangents and the partial surface normals along them.

We can thus formulate our overall surfacing goal as generating a manifold union of inter-stroke and gap-spanning strips that interpolate the vast majority of the input stroke edges and vertices (Figure 6, right). The inter-stroke strips need to connect side-by-side stroke sections, and all strips need to be persistent and connect adjacent stroke vertices with similar normals (up to orientation). By design, we do not seek to connect distinctly separate connected components (Figure 8), leaving this optional step to the user.

Notably, relaxing the manifoldness constraint makes the problem much easier. However, non-manifold meshes cannot be processed by many mesh processing algorithms and are not supported by many commonly used data-structures, making the results significantly less usable.

4.2 Algorithm

We designed our algorithm based on the requirements above. Since the geometry of the gap-spanning mesh strips can only be determined once all inter-stroke strips are in place, we compute the inter-stroke mesh strips first (Section 5) and then compute the gap-spanning ones (Section 6). This separation into stroke and gap surfacing steps allows us to take advantage of the directional similarity between strokes first, and to subsequently leverage direction similarity between (previously non-existent) partial-surface boundaries. Our first step computes dense matches between stroke vertices, then uses these matches to form initial mesh strips between the strokes (Figure 7c) and finally removes the non-manifold artifacts in the resulting mesh (Section 5). SurfaceBrush employs a similar three-step solution process during the gap processing step to match and then connect the boundaries of the partial surfaces using gap-spanning mesh strips (Section 6).

Pre-Processing. When artists use digital sketching tools, they often activate the stylus or controller trigger a few milliseconds before starting the stroke drawing motion and deactivate it a few milliseconds after concluding the motion [Liu et al. 2018]. This behavior produces short randomly oriented stroke sections next to stroke end-points. Our pre-process removes these redundant sections using an approach similar to [Liu et al. 2018]: we check if the strokes have an abrupt direction change (angle of 45° or less between consecutive tangents) within 15% of overall stroke length from either end and remove the offending end-sections.

5 INTER-STROKE SURFACE STRIPS

At the core of our framework is the need to match sections, or edge sequences, along input strokes that bound surface strips on the artist-envisioned surface. When matching stroke sections, we seek

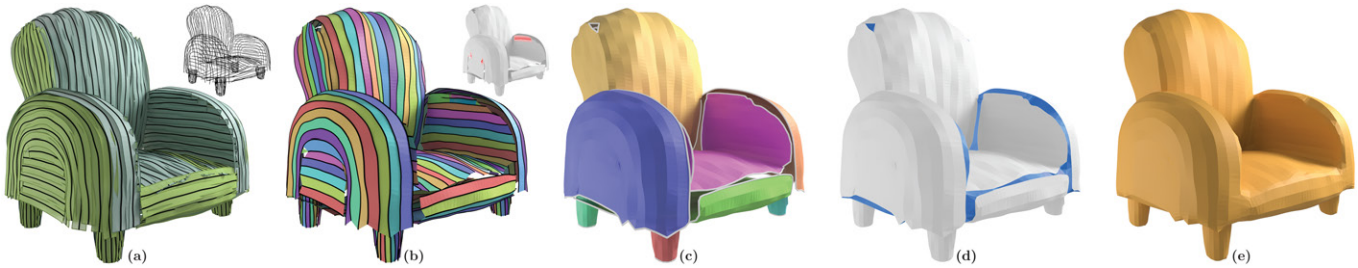


Fig. 7. SurfaceBrush stages: (a) Input stroke polylines (black) and ribbons (color reflects normal orientation). (b,c) Inter-stroke strip computation: initial mesh strips imposed by vertex-to-vertex matches (non-manifold edges highlighted in the inset) (b) and output manifold partial surfaces (each connected component rendered in a different color) (c). Gap closure (d,e): gap-spanning mesh strips (d), and final surface (e). Input drawing: © Elinor Palomares.

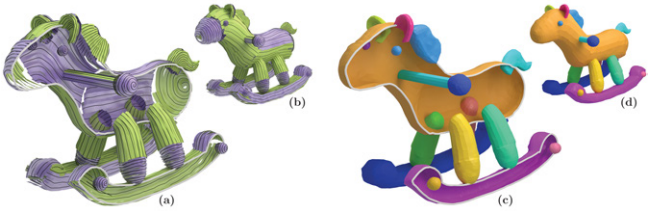


Fig. 8. (a) The horse (Figure 1 contains multiple disjoint intersecting stroke groups (see cutout view), (b) resulting surfacing output with each connected component drawn in separate color. We use a Boolean union of these parts as input to fabrication (Figure 1d). Input drawing: © Jafet Rodriguez.

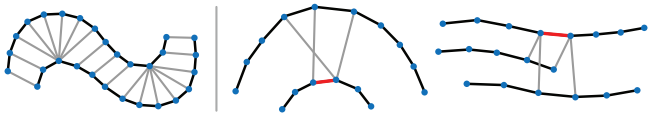


Fig. 9. (left) Desirable non-bijective matches accounting for uneven density. (right) Groups of matches that induce non-manifold meshes. Note that one cannot determine if the result will be manifold by looking at a subset of the matches (pairwise matches in 3D that cross one another in some view do not necessarily induce incompatible matches.)

matches that reflect four key properties: proximity, tangent similarity, persistence, and normal consistency. Since we seek a manifold output, we expect each stroke section to have at most one matching section on its left and one on its right. Sections in the middle of a cluster of side-by-side strokes should have matches on both sides (inset, green); sections along the boundaries of such clusters should have a matching section only on one side (inset, blue); and outlier or isolated sections should have no matches on either side (inset, red).

The partition of strokes into sections and the classification of these sections into the three types above are not known a priori and need to be deduced by our algorithm. We simultaneously segment strokes into sections and match them with respective sections on the same or other strokes, using a discrete optimization framework, that operates on the stroke's vertices. Specifically, we first obtain pairwise vertex-to-vertex matches and then use those to obtain the stroke sections and the correspondences between these sections: each pair of matching sections is defined by a maximal consecutive sequence of vertices on one stoke that match to another consecutive vertex sequence on the same or other stroke. Note that the vertex-to-vertex matches should not necessarily be bijective - given strokes with different vertex density we want to allow many to one matches to enable dense correspondences (Figure 9, left).

To account for the demands above, we need to obtain vertex-to-vertex matches that satisfy three types of criteria: (1) criteria that can be assessed at the level of individual pairwise vertex matches, (2) criteria that require assessing two matched vertex pairs at once, and (3) criteria that require assessing three or more pairs in tandem. Specifically, proximity, tangent similarity, and normal consistency can be assessed at the individual vertex-to-vertex match level. Promoting persistence implies prioritizing configurations where consecutive stroke vertices match to similarly consecutive vertices, necessitating assessing two matched vertex pairs at once. Lastly, assessing manifoldness requires analyzing, and consequently disallowing, configurations of three or more matched pairs (Figure 9, right), as smaller subsets do not necessarily provide sufficient information.

Even a simpler variant of our problem, one where the decision about incompatible matches can be done by assessing two matched pairs (rather than a larger group) is shown to be NP-complete via a reduction from 3D matching, which was shown to be NP-complete by Karp [1972]. The reduction is straightforward: the pairs in this problem correspond to sets in the 3D matching problem, and two pairs are prevented from coexisting unless the corresponding sets are disjoint. Thus, obtaining matches that satisfy our criteria using off-the-shelf methods is impractical. We nevertheless efficiently obtain a desirable solution that accounts for all three criteria types by using a multi-stage matching method that leverages the anticipated persistence of the matches we seek (Figure 10).

We first note that absent the manifoldness requirement, the matches we seek for can be computed independently for each stroke. Specifically, for a single stroke we can cast the optimization of the remaining criteria as a maximization of a score function (Section 5.1) that accounts for both the quality of individual matches and for persistence, or pairwise compatibility between the matches at consecutive stroke vertices. The matches that maximize this combined function can be efficiently computed using a classical dynamic programming framework [Viterbi 1967] (Section 5.1). This method, however, is designed for finding matches for all vertices and consequently cannot account for cluster borders or outliers. Thus, to avoid undesirable

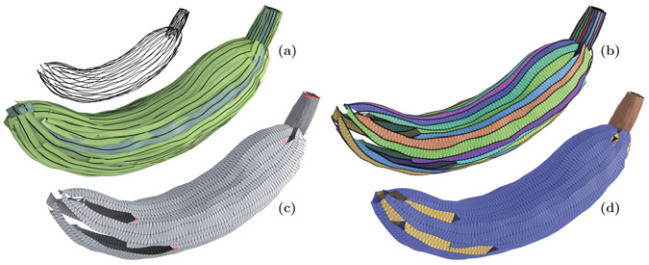


Fig. 10. Inter-stroke mesh strip formation: (a) input strokes and ribbons (inset shows stroke polylines) (b) initial mesh strips reflecting restricted matches (c) non-manifold edges and vertices in the initial mesh highlighted in red, (d) output partial meshes after consolidation and extension. Input drawing: © Jafet Rodriguez.

matches, we restrict the set of per-vertex *matching candidates* during this computation. Our first matching pass (Section 5.2) uses very conservative matching candidate sets, generating correct matches for a large subset of vertices but intentionally leaving some vertices unmatched. We use the computed matches to define inter-stroke mesh strips (Section 5.3, Figure 10b).

We eliminate non-manifold configurations in the resulting mesh (Figure 10c) while minimally reducing the matching score function by formulating these goals as a classical correlation clustering problem [Bansal et al. 2004] and solve it using an approximation method (Section 5.4). While the problem solved in this step remains NP-hard, thanks to our restrictions on the possible matches assessed and our enforcement of persistence between the matches, the number of non-manifold artifacts in the resulting mesh is very small. Thus, they can be efficiently and effectively resolved by applying the clustering to only small subsets of the mesh triangles enabling speedy solution.

The restrictions on the matching candidate sets imposed in our first matching pass (Section 5.2) may result in unmatched stroke vertices for which suitable matches do exist (the unfilled spaces between strokes in Figure 10b). We generate mesh strips connecting such previously unmatched vertices by applying the matching, meshing and manifold consolidation steps again to stroke sections along the boundaries of the current partial surface, using an updated more lax matching candidate set (Section 5.5). The output of this step is a manifold partial surface mesh interpolating stroke groups with similar directions (Figure 10d).

5.1 Match Computation

When looking for matches, we distinguish between left and right sides of each stroke using the direction of the stroke binormal $b(p) = t(p) \times n(p)$ in the local Frenet frame at each stroke vertex p defined by the stroke's tangent $t(p)$ and normal $n(p)$, to distinguish between these sides. Given a stroke $S = p_0, \ldots, p_n$ and a candidate set of matching vertices for each stroke vertex p, we evaluate the potential left (or right) matches (p_i, q_i) using a combination of vertex-to-vertex scores $S_l^v(p_i, q_i)$ for left side matches and $S_r^v(p_i, q_i)$ for right-side matches, and a persistence score $S^e(p_i, p_{i+1}, q_i, q_{i+1})$ that assesses the compatibility between the potential matches of

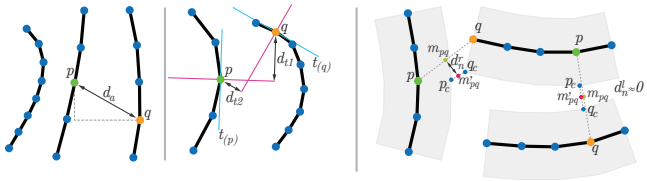


Fig. 11. Components of the vertex matching score $S_r^v(p, q)$: d_a, d_t, d_n^r

consecutive stroke vertices. Both scores are designed to be symmetric so as to prioritize matches which are bijective whenever possible, in order to reduce the occurrence of non-manifold artifacts. We define the combined score of matching the vertices of S to the vertices $Q = q_0, \ldots, q_n$ as their left side matches as

$$M_l(S, Q) = \prod_i S_l^v(p_i, q_i) S^e(p_i, p_{i+1}, q_i, q_{i+1}) \tag{1}$$

We define $M_r(S, Q)$ in a symmetric manner and look for left and right matches that maximize these scores. We use a product rather than a sum to discourage outlier matches. Maximizing the per-stroke scores can be seen as a variant of the classical Markov chain optimization. Given a set of *matching candidates* $C(p_i)$ for each vertex $p_i \in S$, we can compute the matches within these sets that independently maximize $M_l(S)$ or $M_r(S)$ in polynomial time using the classical Viterbi algorithm [Viterbi 1967]. To obtain a valid solution, we exclude from the per-stroke scores vertices with empty matching candidate sets $C(p_i)$ or the edges emanating from them. The strategy we employ to compute the matching candidate sets during different stages in our surfacing process is elaborated on in relevant sections below. We define our overall matching goal as maximizing the matching scores across all strokes S in our drawing,

$$M = \sum_S M_r(S, Q) + M_l(S, Q). \tag{2}$$

Absent any constraints, this goal can be achieved by maximizing the scores for each stroke individually (as there is no requirement for the matches to be symmetric).

Vertex-to-Vertex Matching Score. Given a pair of vertices p and q, we define the score of using q as the left or right match of p as a function of three distance terms, designed to be on the same scale (Figure 11). The first is the absolute distance between them

$$d_a = \|p - q\|.$$

The second term measures the degree to which the vertices are side-by-side with respect to their respective tangents, $t(p)$ at p and $t(q)$ at q, as

$$d_t = \frac{|(p - q) \cdot t(p)| + |(p - q) \cdot t(q)|}{2}.$$

We set $t(o) = (o_n - o_p)/\|o_n - o_p\|$ where o_n and o_p are the next and previous vertices on the stroke of the vertex o.

Lastly, we use the following construction to measure the degree to which the vector $\overrightarrow{pq}$ is orthogonal to the stroke normals at p and q and to assess whether the matches are consistent with respect to the strokes' Frenet frames, namely whether the left (or right) match of each vertex is on its right side with respect to its Frenet frame

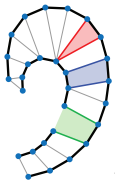
(see Figure 11, right). When assessing a left match, we compute an offset vertex p_c located at distance $w(p)$ on the left side of p along the frame's binormal $b(p)$, where $w(p)$ is the user-specified stroke width at p. We compute both left and right offset vertices q_r, q_l for q using a similar strategy and using offset magnitude $w(q)$. We set q_c to the offset vertex of q closest to p_c and compute their midpoint $m'_{pq} = (p_c + q_c)/2$. When q lies to the left of p and $\overrightarrow{pq}$ is orthogonal to the stroke normals at p and q, this midpoint m'_{pq} coincides with the midpoint $m_{pq} = (p + q)/2$. If either one of the criteria does not hold, the two midpoints will be far apart (see Figure 11,right). Following this observation, we define

$$d_n^l = \|m_{pq} - m'_{pq}\|.$$

We define the overall score for assigning q as the left match of p:

$$S_l^v(p, q) = e^{\frac{-(d_a+d_t+d_n^l)^2}{2\sigma^2}}. \quad (3)$$

We define the right-side assignment score $S_r^v(p, q)$ using d_n^r computed symmetrically to d_n^l. Our empirical observations indicate that users rarely leave unintentional gaps between side-by-side stroke ribbons that are wider than half of these strokes' widths. Thus, we expect the values of each of the three distance metrics d_a, d_t and d_n for most desirable matches to be less than $d_{max} = 1.5(w(p)+w(q))/2$. Consequently, we expect pairwise matches where the sum of the three distances exceeds $3d_{max}$ to be undesirable. Using the three sigma rule we encode this preference by setting $\sigma = d_{max}$.



Persistence Score. Persistence requires the majority of consecutive vertices along a given stroke to match to similarly consecutive vertices (blue in inset). Exceptions include discretization mismatches (red in inset) and transitions between stroke sections that bound different strips (green in inset). We account for persistence without unduly penalizing such exceptions and assess the acceptability of these exceptional cases when they occur by formulating this score using geometric rather than topological properties. In addition to promoting persistence, the score we use further reinforces our preference for matching side-by-side, parallel, stroke sections.

Given a pair of consecutive vertices p_i, p_{i+1} that match to a pair of vertices q_i and q_{i+1}, respectively, we measure persistence using a combination of three distances

$$d_p = \|(p_{i+1} - p_i) - (q_{i+1} - q_i)\| + \|(p_{i+1} - q_i) - (q_{i+1} - p_i)\| + \|(p_{i+1} - q_{i+1}) - (p_i - q_i)\|$$

The first term promotes matches that have the same spatial relationship between the edge p_i, p_{i+1} and the line q_i, q_{i+1}. The second and third jointly promote co-planarity and parallelism between them. These terms zero out when the edges are both parallel and coplanar and jointly reflect how far they are from satisfying these conditions. We convert this distance sum into a score in the $[0, 1]$ range as follows.

$$S^e(p_{i,i+1}, q_i|q_{i+1}) = e^{\frac{-d_p^2}{2\sigma^2}}. \quad (4)$$

We use the same value of σ as for the vertex-to-vertex matching score, following the same argument.

5.2 Restricted Matching

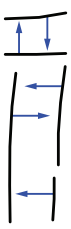
Directly computing the best matches for each stroke while including all vertices on all strokes in the candidate sets of each vertex is computationally expensive. Moreover, our per-stroke score optimization is defined so as to find left (right) matches for each vertex with a non-empty left (right) matching candidate set. Yet, user drawings may depict open surfaces whose boundary vertices should have matches only on one side, and isolated strokes which should have no matches on either side. Thus, to avoid outlier matches, we need to discard potential outliers during candidate matching set computation. Our restricted matching pass obtains conservative, reliable matches by leveraging our expectation of match persistence. Persistence indicates that most strokes are likely to have just a few, and frequently only one matching stroke on the left or right. Following this observation, instead of looking for per-vertex matches globally, we first locate for each stroke a single most likely, or most *dominant*, *neighboring* stroke on its right and a single one on its left. We then compute the best left and right per-vertex matches along each stroke using a restricted set of matching candidates, which only includes vertices on these dominant neighboring strokes, if they exist, and on the currently processed stroke itself.

Locating Dominant Neighboring Strokes. To locate one dominant left and one dominant right neighbor for each stroke S, we first compute matches for vertices along this stroke that maximize $M_l(S)$ and matches that maximize $M_r(S)$, by considering possible matching candidates on **all** strokes. We then use the frequency of matches from the stroke S to other strokes to define the dominant left and right neighbors for this stroke.

During this first matching pass , we define the left (right) *matching candidate set* for each vertex p to include vertices q across **all** input strokes that satisfy the following *baseline* matching conditions:

(1) $\|q - p\| \leq d_{max}$
(2) the angle between $q - p$ and the binormal $b(p)$ is at most $60°$ (for the right candidate set we assess the angle between $q - p$ and $-b(p)$)
(3) $q \neq p$ and q is not an immediate neighbor of p along its stroke.

The baseline matching conditions are designed to reduce the likelihood of outlier matches and to speed up computation by reducing the solution space.



We define the left (right) *matching frequency* F_{ST} from stroke S to stroke T as the percentage of vertices $p_i \in S$ that match vertices $q_j \in T$ as their left (right) matches. Note that this value is not symmetric: given for instance two side-by-side strokes where one is shorter than the other, the frequency for mapping the shorter to the longer will be higher than the other way around. We define a stroke T to be the *dominant left (right) neighbor of stroke* S if the following three conditions hold: the left (right) matching frequency from S to T is higher than from S to any other stroke; this frequency F_{ST} is at least 30%; and at least one pair of consecutive vertices on S matches a pair of consecutive vertices on T (the latter constraint discards T-junction matches where a stroke matches an end-vertex of another). As the inset shows (arrows point to the computed

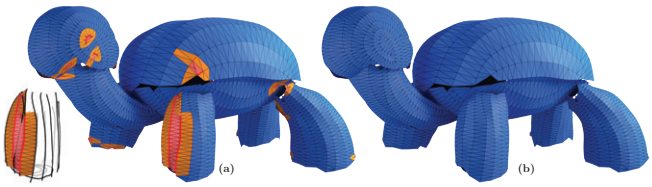


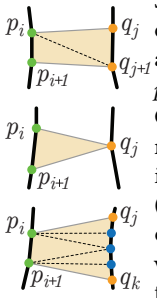
Fig. 12. (a) Annotated consolidation input: incompatible triangles (red, offending edges highlighted) undecided triangles (orange), unaffected "output" triangles (blue). (b) Consolidated manifold mesh. Inset shows the strokes that trigger the non-manifold artifacts on the leg.

dominant neighbors), we intentionally do not enforce symmetry in this process - allowing multiple strokes to share the same stroke as their dominant left or right neighbor and have strokes with no neighbors on one or both sides.

Restricted Matching Candidate Set. We define the left (right) restricted matching set of each vertex along a given stoke to include vertices on the same stroke and its left (right) dominant neighbor, if one exists, that satisfy the baseline matching conditions (1) to (3). We restrict this set further in the vicinity of stroke end-vertices, forcing condition (2) to hold at both p and q. We use these restricted matching candidate sets to compute the left and right per-vertex matches that optimize $M_l(S)$ and $M_r(S)$. Limiting the matching candidate sets drastically reduces the likelihood of outlier matches and produces locally optimal results along each individual stroke (Figure 10b).

5.3 Mesh Strip Generation

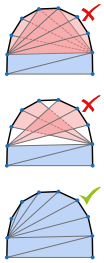
Our meshing step receives as input a set of vertex-to-vertex matches p_i, q_j between vertices on the same or different strips. It uses match pairs containing consecutive vertices on each stroke to determine the local meshing strategy (see inset). Given two consecutive match pairs p_i, q_j and p_{i+1}, q_{j+1} (or similarly p_{i+1}, q_{j-1}) it triangulates the quad $p_i, p_{i+1}, q_{i+1}, q_i$ (or similarly $p_i, p_{i+1}, q_{i-1}, q_i$) using the diagonal that produces a more planar, better shaped triangulation.



Since we expect the mesh to be fair, it discards the quad if the dihedral angle between the resulting triangles is under 45°. Given two consecutive pairs p_i, q_j and p_{i+1}, q_j, it forms the triangle p_i, q_j, p_{i+1}. Given consecutive vertices p_i, p_{i+1} that match two non-consecutive vertices q_j, q_k on the same stroke, it triangulates the polygon formed by the edges $(p_i, q_j), (q_k, p_{i+1}), (p_{i+1}, p_i)$ and the section q_j, q_k only if we have no matches between any pair of vertices within this section. This condition is used to avoid introducing non-manifold configurations. It triangulates the polygon using edges that connect interior vertices along the section q_j, q_k to p_i or p_j, selecting a manifold, consistently oriented triangulation that maximizes the matching score along the section.

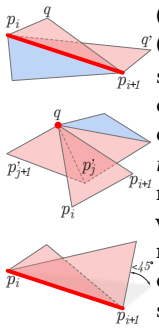
5.4 Manifold Consolidation

We expect each stroke section in the final output mesh to bound at most one mesh strip on its left and right. Violating this expectation produces partial surfaces with non-manifold edges or vertices (Figure 12a). While our matching strategy is designed to minimize the likelihood of such non-manifold artifacts, it does not fully prevent them. Thus the partial mesh defined by the union of mesh strips computed as described above may contain non-manifold edges and vertices. Our manifold consolidation step removes a subset of the triangles surrounding such non-manifold entities to produce a manifold output mesh. In selecting the subset to remove it seeks to maintain as many triangles as possible in place, while optimizing the matching quality along mesh edges connecting matched vertices.



Since strips often overlap along only a small portion of their boundaries, leaving one strip in place while deleting others would introduce unnecessary holes into the mesh. At the same time, deleting individual triangles next to non-manifold edges and vertices can result in an inconsistent mesh, which does not satisfy our persistence prior and contain undesirable holes and tunnels, as illustrated in the inset, middle. We obtain a manifold and fair solution that respects our priors by employing a correlation clustering framework [Bansal et al. 2004] (see inset,bottom). Our persistence term, combined with the use of the restriction of the matching set, strongly discourages the type of matches that lead to non-manifold artifacts. Consequently, the non-manifold artifacts we face are typically very localized, allowing us to employ correlation clustering locally, one problematic mesh region at a time.

We first identify pairs of adjacent triangles which we consider as incompatible, namely ones that cannot jointly belong to the output mesh, using the following criteria (see inset). (1) A pair of triangles that share the edge p_i, p_{i+1}, are classified as incompatible if their non-shared vertices q and q' are on the same side of this edge.



(2) A pair of triangles $t_1 = (q, p_i, p_{i+1})$ and $t_2 = (q, p'_j, p'_{j+1})$ sharing a common vertex q are classified as incompatible if they are on the same side of the stroke containing q and if the projection of one of the edges of t_1 on the plane of t_2 intersects t_2 or vice versa (note that a valid mesh can contain multiple triangles that are on the same stroke side with respect to a common vertex as long as they do not "overlap"). (3) Lastly, while sharp creases in our output mesh are possible, we view them as undesirable, and classify triangles that share a common edge as incompatible if the dihedral angle between them is less than 45°.

We resolve all of these artifacts by discarding a subset of the incompatible elements together with a subset of the triangles in their immediate vicinity producing a manifold mesh. In making the decision which elements to keep and which to remove we seek to maximize the output matching score M (Equation 2). Since directly optimizing this score would make the problem intractable, we approximate it in our graph arc weight assignment as described below.

We compute the graphs we apply the clustering to as follows. We classify triangles as *undecided* if they belong to a set of incompatible triangles or if they are immediately adjacent to an edge or vertex shared by a pair of incompatible triangles, and classify them as

output otherwise. We form a separate graph for each connected component of undecided triangles (which share edges or vertices). Each graph has a node for each undecided triangle, and a single *output* node that represents all output triangles. We connect these nodes with arcs and assign arc weights as follows.

- We construct an arc for each pair of mutually incompatible triangles t_1, t_2, and assign it a high negative weight (-30).
- We construct arcs for all pairs of undecided triangles that share common edges and are not mutually incompatible and assign them a weight of 1.
- We construct an arc between each undecided triangle and the output node and define its weight as follows. We recall that each triangle t in our mesh connects a stroke edge p_i, p_{i+1} to a vertex q on the same or other stroke, which is to the left or right of this edge. We compute the sum of the matching scores $M(t) = M_a(p_i, q) + M_a(p_{i+1}, q)$ where a is l or r based on the side of the stroke that the vertex is on. We define the arc weight as $M(t) + C$ where C is the number of edges shared by the triangle t and output triangles. We include the edge count in the cost to implicitly minimize the size of the holes formed by the consolidation step.

We use these assignments to formulate clustering as a constrained maximization problem. We maximize $\sum_{ij} w_{ij} Y_{ij}$, where w_{ij} are the weights defined on the graph arcs, and $Y_{ij} = 1$ if the end nodes of an arc are in the same cluster and $Y_{ij} = 0$ otherwise. The sum increases whenever the end-nodes of an arc with a positive weight are assigned to the same cluster or when end-nodes of an arc with a negative weight are kept apart. We compute an adequate approximate solution to this problem using the lifted multicut approximation method [Keuper et al. 2015]. Following the computation, we retain the subset of undecided triangles that belong to the same cluster as the output node (Figure 12b). This subset is guaranteed to be manifold, as the correlation clustering method ensures that any pair of conflicting triangles are placed into different clusters. We apply this clustering process to every group of triangles that are marked as undecided; the union of the triangles previously marked as output and the collection of subsets of undecided triangles kept after each clustering operation then form our output manifold mesh.

5.5 Partial Mesh Extension

The partial mesh generated via the three step process described above was computed by only considering matches from each stroke to itself and its dominant neighboring strokes. This restriction produces mesh strips that satisfy all our criteria, but may leave side-by-side stroke sections unmatched in cases where a stroke has multiple immediately adjacent strokes on its left or right (Figure 10b). We connect such left-out stroke sections with mesh strips using a similar process to the one above. We first apply our matching algorithm (Section 5.1) to sections of the input strokes that lie on the boundaries of the current partial meshes. During the match computation, we restrict the candidate set of each vertex p to include vertices on the boundaries of the connected mesh component that p is on that satisfy the baseline matching conditions described in Section 5.2, and use the same restriction on tangent similarity near end-vertices. The restriction to the same connected component is designed to limit the matches to lie on roughly similarly directed strokes. We then apply our consolidation process to the mesh computed from these matches. Following this consolidation, we have a manifold mesh (Figure 10d), which connects similarly directed strokes using mesh strips. We compute consistent normal orientations for each connected component of this mesh using simple breadth-first-traversal and close obvious small holes (ones with four or less sides) inside each such component (typically located at transitions between different strips along the same stroke).

6 CLOSING THE GAPS

The final stage of our algorithm closes gaps between close-by components of the partial mesh as well as any remaining narrow holes within them. It achieves this goal by using a similar mesh strip formation process to the one used to form inter-stroke strips, with some minor differences outlined below. This step is quite similar to the mesh extension process (Section 5.5); while separating the two improves input fidelity, for simplicity of implementation the process outlined in Section 5.5 can be skipped with only minor impact on fidelity.

Boundary Smoothing. The boundaries of the partial surfaces are often very jaggy and contain occasional overlaps between opposite boundaries. We resolve both artifacts by locally smoothing the boundary vertices, using the following simple update $p_i = p_i/2 + (p_{i-1} + p_{i+1})/4$. We only apply this update if it does not change significantly the normals of the adjacent triangles (which we evaluate by thresholding the angle between the pre-smoothed and smoothed normals to be at most 45°). This step leads to more reliable matches and better shaped gap-spanning mesh strips.

Matching. We compute the matching scores as described in Section 5.1, defining the Frenet frame at each boundary vertex using the tangent to the boundary and the normal to the partial surface. We set the maximal distance d_{max} for the boundaries of each connected partial surface component using the average of the distances between matched vertex pairs across this component. We define the candidate set of each vertex to include other vertices on the boundaries of both this partial surface and others that satisfy the baseline matching conditions (1) and (3) in Section 5.2. We relax condition (2) to require the angle between $q - p$ and the boundary binormal $b(p)$ to be at most 80° (we orient the binormal to point away from the bounded surface). We then proceed to compute the best matches for each boundary loop as described in Section 5.2.

Meshing and Consolidation. We form gap-spanning mesh strips by applying the algorithm in Section 5.3 as-is to the newly computed matches and remove non-manifold artifacts as described in Section 5.4. During consolidation, we leave all triangles on the previously computed partial surface in place by labeling them as *output*.

Orientation. Our partial surfaces are oriented during construction and we orient each gap-spanning strip after consolidation. However, when connecting these surfaces and strips together, we may introduce gap-spanning strips that cannot be consistently oriented when merged with the connected components they bound (a Moebius

strip effect). We detect and resolve such configurations by first comparing the orientations of each partial surface and a strip it shared a border with. For an orientable surface, we expect the orientations of pairs of border triangles (one from the strip and one from the partial to either be identical for all pairs of triangles or be inverted for all of them. If this is not the case, we count the number of aligned and inverted pairs. We keep the strip triangles which conform with the majority choice (aligned or not) and discard the others. Finally, we use a greedy breadth-first traversal to establish a common orientation for the combined mesh. This process produces consistently oriented surfaces for all orientable input geometries tested.

Optional Post-Processing. Our core surfacing framework robustly closes narrow gaps and holes between input strokes. Since our system is not limited to closed surfaces only, the determination whether to surface larger and hence inherently ambiguous holes is left to the user. We allow users to selectively close such large holes using the hole-filling mechanism implemented in CGAL [2018]. Finally, users can smooth the resulting mesh using standard Laplacian smoothing to eliminate local geometric noise, and use Boolean operations to join intersecting closed mesh components together.

As noted earlier, artists often use sparse strokes to communicate narrow geometries (such as the chicken's feet in Figure 2). We represent such isolated strokes, ones which are not part of the output triangulation, using their original triangulated ribbons.

7 VR DRAWING STUDY

To observe how experts and amateurs communicate shapes when presented with a Virtual Reality flat stroke drawing interfaces, we asked five participants to draw simple shapes (cubes and half-spheres) using this interface. Our set of participants included one formally trained artist, two 3D modelers, and two programmers. Each participant was provided with a quick tutorial on the use of the TiltBrush drawing interface; to avoid biasing the examples shown during the tutorial did not include any dense stroke drawings, but focused on basic TiltBrush manipulation. Participants were allocated a total of one hour to practice using TiltBrush and to then "Try to draw a clean description of the surface" of a cube and a half-sphere. They were told to stop once they were happy with the results. Three participants employed the dense side-by-side stroke drawing style from the get-go and proceeded to draw both examples using this style. The remaining two created sparse curve drawings as their initial attempt, but were not satisfied with those and after some experimentation converged to the dense side-by-side stroke style as well. Figure 13 shows six of the drawings created by the participants. Additional drawings are included in the supplementary material.

In addition to this targeted experiment, we asked a modeler and a non-expert to create VR drawings of shapes of their choice using TiltBrush after showing them a few typical inputs created by one of the authors. The drawings they produced are shown in Figures 1, 3, and 14 (bonsai, pumpkin, mushroom, hart, tree, piggy bank, teddy, dolphin). The drawings created exhibit the characteristics we describe in Section 3. Drawing these models took the participants 30 minutes on average. Creating such irregular, free-form shapes using existing modeling technologies would require significantly more time and a degree of familiarity with these tools that our target users

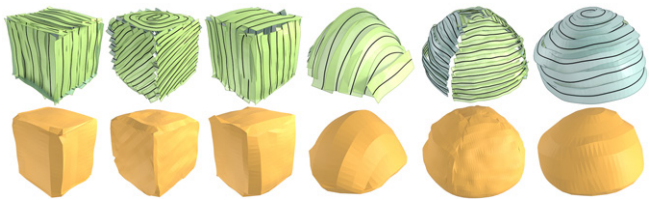


Fig. 13. Typical drawings created by study participants when instructed to create cubes and half-spheres (domes) and our reconstruction results.

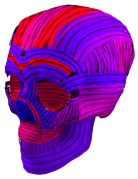
may not possess. The professional modeler who created our bunny input (Figure 14) in 25 minutes estimated that it would take him two and a half hours to create the same model in 3D Studio Max. These experiments confirm that users see the dense side-by-side drawing style as a convenient and effective way to communicate shapes, and validate our argument for employing dense stroke drawing as a modeling tool suitable for artists and amateurs.

8 RESULTS AND VALIDATION

We tested our algorithm on twenty nine inputs. These include inputs created by amateur first-time users (Figure 13), an amateur user who had some experience with the system (e.g. turtle, bonsai, chicken, heart), and two modelers (e.g. bunny, wooden horse, teapot, skull). Two inputs (ship, plane) were downloaded from online repositories. The inputs range in complexity from simple shapes (spheres and cubes in Figure 13) to complex models such as the bunny, horse, and skull. In all cases our outputs accurately reflect the user-drawn shapes.

Many images in online VR drawing repositories, see e.g. Figure 16 are created to provide a compelling visual rather than a detailed model description; they contain multi-color strokes and use large numbers of isolated strokes to convey narrow ruled surfaces. To process this data, we augment our surfacing method to use color as a negative matching cue, disallowing matches between differently colored strokes. Our method reconstructs all surface elements, e.g. tower, sail, deck, in these inputs while preserving the isolated features intact, allowing us to rerender the sail on the boat with surface texture.

Artists often depict surface creases (Figure 15a) by drawing ribbons whose sides delineate the desired crease shape. Our default algorithm is designed to connect stroke spines and when used as is rounds such creases leaving a beveled edge (Figure 15b). We provide users with an option to retain creases, if they choose to do so. To this end, when the the normals of two matched side-by-side strokes form angle of 90° or less, instead of connecting the stroke spines, we retain one half of the ribbon along one of the strokes, and connect the side of that ribbon to the spine of the other. We use the stroke with a larger number of vertices along the marched section of interest to perform this task (Figure 15c).



Time Information. VR drawing software records the time each stroke was drawn at. While artists draw some immediately adjacent strokes sequentially, as the inset shows, drawing order is not a reliable indicator of adjacency (color represents drawing time

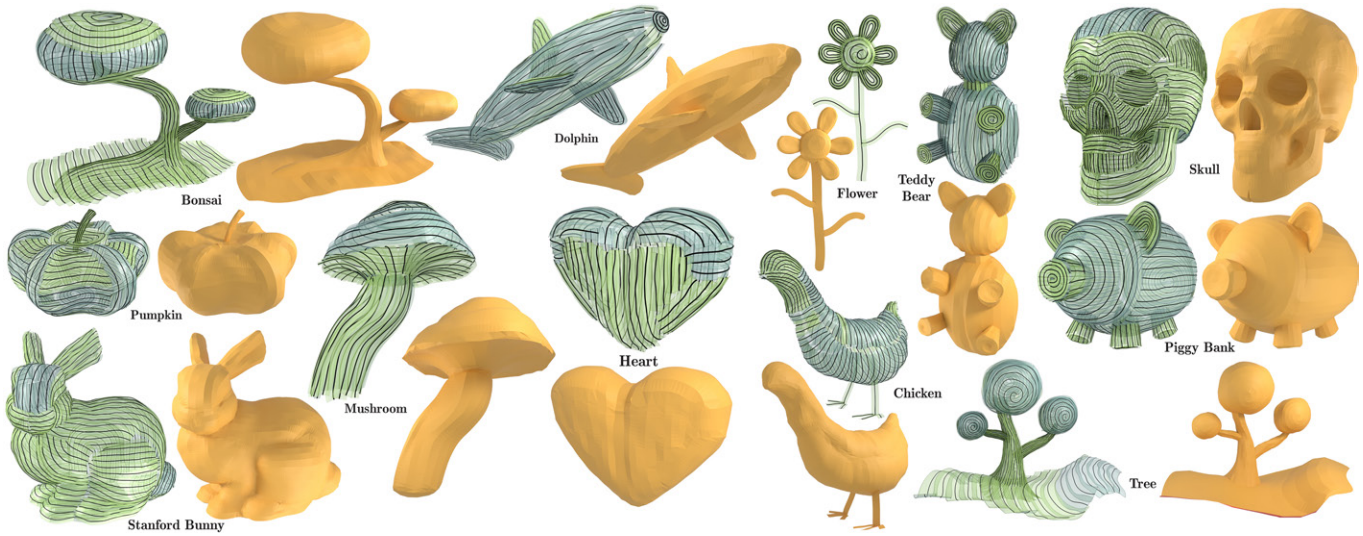


Fig. 14. A range of inputs and SurfaceBrush results. Flower and skull: © Enrique Rosales, bonsai, pumpkin, mushroom, dolphin, heart, chicken, teddy bear, piggy bank, tree: © Elinor Palomares.

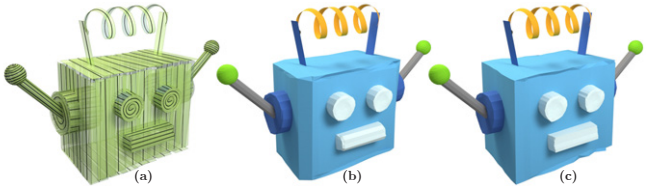


Fig. 15. Optional sharp feature preservation: (a) input, (b) default output with sharp features rounded, (c) with an optional feature preservation step. Input drawing: © Enrique Rosales.

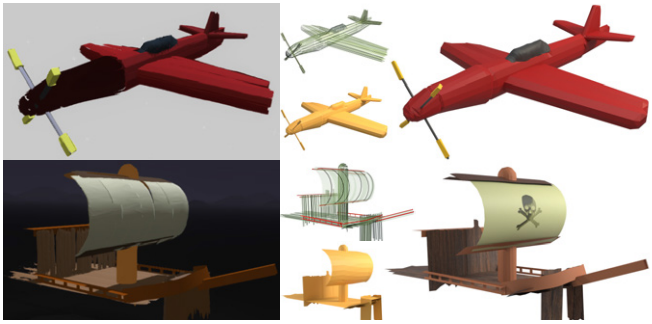


Fig. 16. Given two VR drawing from an online repository ((top) © *Olga Zinchenko* derived from *I Have 2 Cents* (poly.google.com), (bottom) © Skeazy J (poly.google.com)) we successfully reconstruct all the surfaces in the input while preserving isolated strokes (inset, red) intact. We use the obtained surface (bottom, right) to rerender the input with texture (a functionality not supported by a ribbon-based representation).

from earliest, blue, to latest, red). Thus limiting or biasing matches toward immediately preceding or succeeding strokes using drawing order, could produce undesirable artifacts on typical user inputs. Our framework is by design not dependent on stroke drawing order, and thus can robustly handle such typical data.

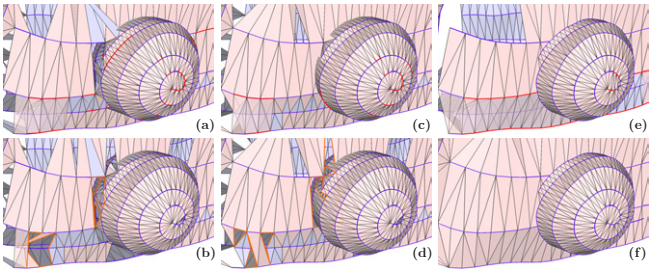


Fig. 17. Relying purely on vertex-to-vertex scores when computing matches between stroke vertices produces pre-consolidation partial meshes (a) with numerous noisy matches and non-manifold edges resulting in post-consolidation surface with many artifacts (b). Using unrestricted matches (c, before consolidation, d, after) produces less artifacts but leaves many undesirable holes, surfacing which would result in sharp dihedral angles and undesirable connections between separate mesh components. Our results before consolidation (e) and after partial surface computation (f). Our output is fair and keeps the separate components apart. Non-manifold edges in red, shading reflects front/back orientation.

Comparison to Algorithmic Alternatives. Figure 17 compares our method against two potential algorithmic alternatives. We show the impact of accounting for persistence during match assessment by introducing the persistence scores S_e (Section 5.1) by comparing our results (Figure 17f) to results produced using only vertex scores S^v (Figure 17ab). As the figure shows, using vertex scores alone results in poor match persistence and subsequent visible surfacing artifacts. Similar artifacts appear if we do not restrict the first matching step (Section 5.2) to only dominant neighbor strokes (Figure 17cd). In all cases, the results obtained using our complete pipeline are significantly more reflective of the user input.

Comparisons to Prior Art. As discussed in Section 2 the inputs we process do not conform to the input specifications of the existing

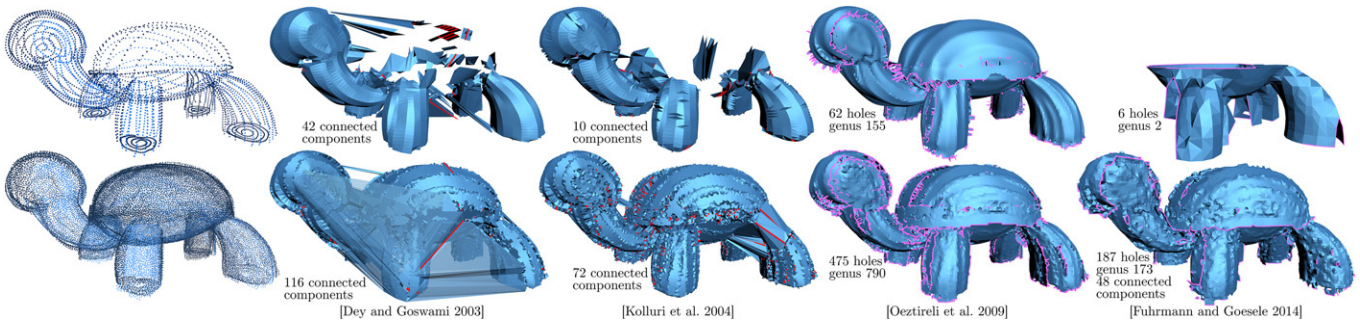


Fig. 18. Comparison against representative point cloud reconstruction techniques using as input stroke vertices (top) and dense ribbon samples (bottom): Left to right : input, [Dey and Goswami 2003], [Kolluri et al. 2004],[Öztireli et al. 2009], [Fuhrmann and Goesele 2014]. Non-manifold edges highlighted in red, boundaries in purple. Our result shown in Figure 21.

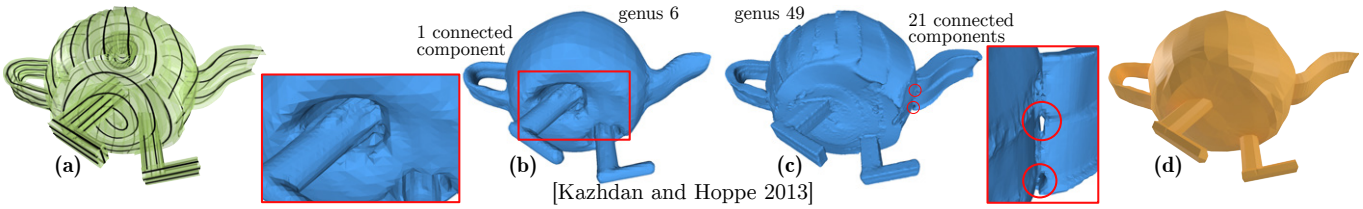


Fig. 19. Even with manually corrected normal orientation (a), state of the art frameworks that utilize normals, here [Kazhdan and Hoppe 2013], produce outputs with excessive genus and other artifacts using stroke vertices (b) or dense ribbon samples (c). (d) Our result. Input drawing: © Jafet Rodriguez.

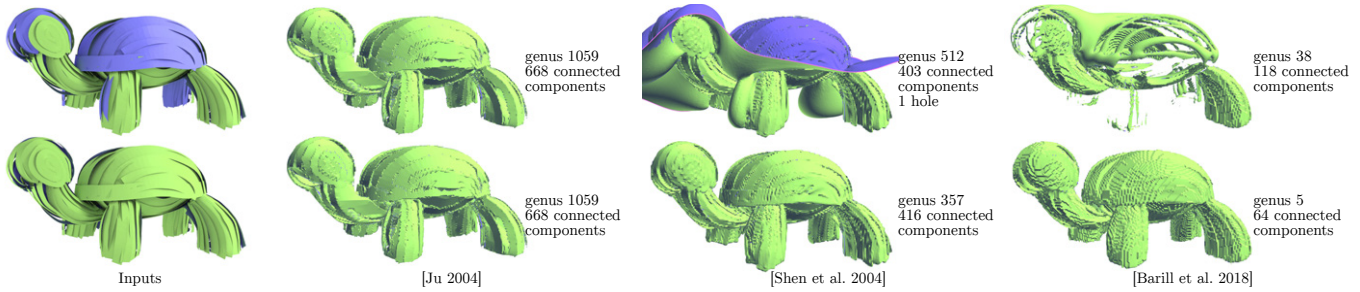


Fig. 20. Attempting to reconstruct the user created models by applying topology repair methods to input ribbons fails on both original ribbons (top) and ribbons wit manually corrected normal orientation (bottom). Left to right: input, result of [Shen et al. 2004], result of [Ju 2004], result of [Barill et al. 2018]. Our result is shown in Figure 21. Input drawing: © Elinor Palomares.

curve loop, cycle or network surfacing methods. Our input drawings can be easily converted into oriented point-clouds by using stroke polyline vertices or point samples on the ruled ribbons around them. Figures 3 and 18 show comparisons of our outputs to those produced from such point-clouds using a range of state of the art techniques [Dey and Goswami 2003; Edelsbrunner and Mücke 1994; Fuhrmann and Goesele 2014; Kazhdan and Hoppe 2013; Kolluri et al. 2004; Öztireli et al. 2009]. Additional comparisons to reconstructions produced using these methods, and the ball-pivoting method of [Bernardini et al. 1999] are included in the supplementary material. Ball-pivoting outputs exhibit similar artifacts to those demonstrated for the other reconstruction techniques.

Existing methods that incorporate per-point normals as part of the data, e.g. [Fuhrmann and Goesele 2014; Kazhdan and Hoppe 2013; Öztireli et al. 2009], typically rely on those to have consistent in-out orientation. Our data does not satisfy this assumption, leading such methods to catastrophically fail, producing meshes with excessive genus and other artifacts. Even when the strokes are manually oriented for global consistency, the self-intersections between different stroke groups commonly present in our data cause major artifacts in the reconstructions computed with these methods (Figure 19). Delaunay type methods [Dey and Goswami 2003; Edelsbrunner and Mücke 1994; Kolluri et al. 2004] are similarly ill-suited for the uneven input spacing and the low-frequency errors present in artist data, and produce outputs with large numbers of non-manifold edges and vertices, multiple redundant connected components, and with mesh triangles connecting unrelated surface

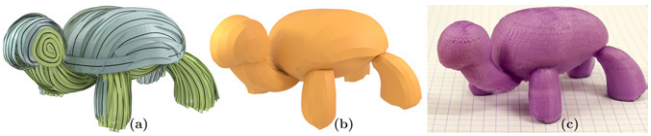


Fig. 21. From VR drawing to 3D printed model: (a) input, (b) reconstructed surface, (c) 3D printed model. Input drawing: © Elinor Palomares.

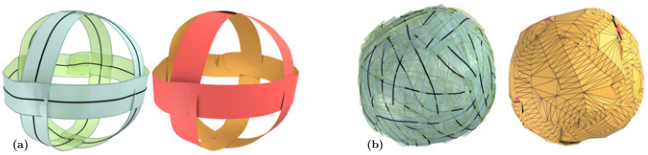


Fig. 22. Our framework is not designed for surfacing very sparse (a) or randomly oriented (b) strokes. In the former scenario it leave the input strokes essentially as is. In the latter where persistence and tangent consistency do not apply it produces results comparable to those obtained via reconstruction from point clouds.

parts. Our targeted framework overcomes all of these artifacts and produces the user-expected output on the tested inputs.

Topology Repair methods [Barill et al. 2018; Ju 2004; Shen et al. 2004] are designed for closed surfaces. Even when the input drawings depict closed shapes, applying these methods to our input triangle ribbons produces inadequate results (Figure 3, top). Result quality only marginally improves when ribbon orientation is manually corrected (Figure 20, bottom).

Parameters and Runtimes. Our method has no user-tuned parameters, and all the results shown in the paper were obtained under identical conditions. Our algorithm takes under 5 seconds to surface the teapot (95 strokes, 3K vertices) models of similar complexity, and takes 52 seconds to surface our biggest model, the horse (298 strokes, 20K vertices) on a 4 core Intel Core i7-6700HQ with 2.60 GHz RAM and 32 GB DDR4. Out of this time, 60% is spent in the matching code and 25% doing consolidation.

3D-Printable Models. As indicated earlier, and illustrated in Figures 2 and 8 users often use disjoint sets of strokes to draw different model parts; these are kept as disjoint components by our method. When a user specifies that the intended output is expected to be a connected closed intersection-free manifold surface, after closing all holes (Section 6) we use a Boolean union operation to combine all overlapping components into one. The resulting models can then be 3D printed as shown in Figures 1, 21.

9 CONCLUSIONS

We present SurfaceBrush, a novel framework for freeform surface modeling using virtual reality brush strokes as input. This modeling interface is supported by a specialized surfacing algorithm that converts raw artist strokes into a manifold, user-intended surface. Our studies show that both experts and amateurs can successfully use our framework to create compelling 3D shapes.

Limitations and Future Work. Our surfacing method is based on observations of practices artists typically employ when using VR brushes to draw 3D shapes. Thus it, predictably, breaks down when artists drastically deviate from the fence-painting metaphor and use either very sparse (Figure 22a) or arbitrarily directed (Figure 22b) strokes. However, as our experiments show, even first-time users typically quickly converge to producing the type of inputs we expect when asked to depict geometric shapes, and thus are unlikely to experience this limitation. At the same time these restrictions may cause difficulties with processing of legacy inputs created to visually and artistically convey rather than model 3D content. Such inputs may use strokes to create artistic effects, e.g. mimicking van Gogh's, impressionist or Pointillist drawing styles, and may use layers of differently directed strokes to depict fur, hair, or texture. Our method is not designed to recover shapes from such highly stylized data.

The focus of our reconstruction method is on fidelity to user input. Exploring regularization and beautification of input and output created from non-expert drawings is an interesting future research topic that could potentially lead to more robust methods that reconstruct user intended rather than directly depicted shapes.

Our reconstruction process is currently offline, thus users can only see the resulting model after completing the drawing. It would be interesting to explore a variation of our method that provides users with real-time feedback as they draw. Such a method can potentially save user time and provide helpful real-time suggestions. As already noted, artist strokes are often aligned with principal (typically minimal absolute) curvature directions - it would be worth exploring how this extra information can be used in geometry optimization and other surfacing tasks down the line.

ACKNOWLEDGMENTS

We are deeply grateful to Nicholas Vining and Nico Schertler for help with paper editing and proofing, to Luciano Silver Burla for help with video creation, to Chrystiano Araújo for help with code integration, to Elinor Palomares for her artistic inputs, and Chenxi Liu and Edoardo Dominici for help with running comparison experiments. The authors were supported by NSERC, CONACYT and Google.

REFERENCES

Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta. 2012. Surface Patches from Unorganized Space Curves. In *Proc. Symposium on Computational Geometry*. 417–418.

Judith Amores and Jaron Lanier. 2017. HoloARt: Painting with Holograms in Mixed Reality. In *Proc. Human Factors in Computing Systems*. 421–424.

Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. 2010. ℓ_1Sparse Reconstruction of Sharp Point Set Surfaces. *ACM Trans. Graph.* (2010), 135:1–135:12.

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models. In *Proc. Symposium on User Interface Software and Technology*. 151–160.

Nikhil Bansal, Avrim Blum, and Shuchi Chawla. 2004. Correlation clustering. *Machine Learning* 56, 1-3 (2004), 89–113.

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Trans. Graph.* (2018), 43:1–43:12.

Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. 2017. A Survey of Surface Reconstruction from Point Clouds. *Comput. Graph. Forum* 36, 1 (2017), 301–329.

F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. 1999. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5, 4 (1999), 349–359.

Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. 2012. Design-Driven Quadrangulation of Closed 3D Curves. *ACM Trans.Grap.* 31, 5 (2012).

CGAL. 2018. CGAL The Computational Geometry Algorithms Library. https://www.cgal.org/

DesignSpace. 2018. DesignSpace VR. http://www.designspacevr.org/
Tamal K. Dey and Samrat Goswami. 2003. Tight Cocone: A Water-tight Surface Reconstructor. *J. Comput. Inf. Sci. Eng.* 3, 4 (2003), 302–307.
Holger Diehl, Franz Müller, and Udo Lindemann. 2004. From raw 3D-Sketches to exact CAD product models Concept for an assistant-system. In *Sketch Based Interfaces and Modeling*.
Herbert Edelsbrunner and Ernst P. Mücke. 1994. Three-dimensional Alpha Shapes. *ACM Trans. Graph.* 13, 1 (1994), 43–72.
Mark Finch and Hugues Hoppe. 2011. Freeform Vector Graphics with Controlled Thin-Plate Splines. *ACM Trans. Graph.* 30, 6 (2011).
Michele Fiorentino, Raffaele de Amicis, Giuseppe Monno, and Andre Stork. 2002. Spacedesign: A Mixed Reality Workspace for Aesthetic Industrial Design. In *Proc. Symposium on Mixed and Augmented Reality*.
Hongbo Fu, Shizhe Zhou, Ligang Liu, and Niloy J. Mitra. 2011. Animated Construction of Line Drawings. *ACM Trans. Graph.* (2011), 133:1–133:10.
Simon Fuhrmann and Michael Goesele. 2014. Floating Scale Surface Reconstruction. *ACM Transactions on Graphics* 33 (2014), 1–11.
Kun Gao and Alyn Rockwood. 2005. Multi-sided attribute based modeling. *Mathematics of Surfaces XI* (2005), 219–232.
GoogleBlocks. 2018. Google Blocks. https://vr.google.com/blocks/
GravitySketch. 2018. Gravity Sketch. https://www.gravitysketch.com/
Cindy Grimm and Pushkar Joshi. 2012. Just DrawIt: A 3D Sketching System. In *Proc. Symposium on Sketch-Based Interfaces and Modeling*. 121–130.
Tovi Grossman, Ravin Balakrishnan, Gordon Kurtenbach, George Fitzmaurice, Azam Khan, and Bill Buxton. 2002. Creating Principal 3D Curves with Digital Tape Drawing. In *Proc. Human Factors in Computing Systems*. 121–128.
Zhiyang Huang, Ming Zou, Nathan Carr, and Tao Ju. 2017. Topology-controlled Reconstruction of Multi-labelled Domains from Cross-sections. *ACM Trans. Graph.* 36, 4 (2017), 76:1–76:12.
J.H. Israel, E. Wiese, M. Mateescu, C. Zöllner, and R. Stark. 2009. Investigating three-dimensional sketching for early conceptual design—Results from expert discussions and user studies. *Computers and Graphics* (2009), 462 – 473.
B. Jackson and D. F. Keefe. 2016. Lift-Off: Using Reference Imagery and Freehand Sketching to Create 3D Models in VR. *IEEE Trans. on Visualization and Computer Graphics* 22, 4 (2016), 1442–1451.
Tao Ju. 2004. Robust Repair of Polygonal Models. *ACM Trans. Graph.* (2004), 888–895.
R. Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, 85–103.
Michael Kazhdan and Hugues Hoppe. 2013. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* 32, 3 (2013), 29:1–29:13.
D. Keefe, R. Zeleznik, and D. Laidlaw. 2007. Drawing on Air: Input Techniques for Controlled 3D Line Illustration. *IEEE Transactions on Visualization and Computer Graphics* 13, 5 (2007), 1067–1081.
Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola, Jr. 2001. CavePainting: A Fully Immersive 3D Artistic Medium and Interactive Experience. In *Proc. Symposium on Interactive 3D Graphics*. 85–93.
Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoué, Thomas Brox, and Bjorn Andres. 2015. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proc. ICCV*. 1751–1759.
Yongkwan Kim, Sang-Gyun An, Joon Hyub Lee, and Seok-Hyung Bae. 2018. Agile 3D Sketching with Air Scaffolding. In *Proc. Human Factors in Computing Systems*. 238:1–238:12.
Kodon. 2018. TenkLabs Kodon. https://www.tenklabs.com/kodon
Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. 2004. Spectral Surface Reconstruction from Noisy Point Clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04)*. 11–21.
Jung-hoon Kwon, Han-wool Choi, Jeong-in Lee, and Young-Ho Chai. 2005. Free-Hand Stroke Based NURBS Surface for Sketching and Deforming 3D Contents. In *Proc. Pacific-Rim Conference on Advances in Multimedia Information Processing*. 315–326.
Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces Through 2D Sketching. *ACM Trans. Graph.* (2017), 125:1–125:14.
Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: Consolidating Raw Sketches into Artist-Intended Curve Drawings. *ACM Trans. Graph.* 37, 4 (2018).
Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2015. Closure-aware Sketch Simplification. *ACM Trans. Graph.* (2015), 168:1–168:10.
A. Nasri, M. Sabin, and Z. Yasseen. 2009. Filling N -Sided Regions by Quad Meshes for Subdivision Surfaces. *Computer Graphics Forum* 28, 6 (2009), 1644–1658.
Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. *ACM Trans. Graph.* 26, 3 (2007).
G. Noris, D. Sýkora, A. Shamir, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. Sumner. 2012. Smart Scribbles for Sketch Segmentation. *Comput. Graph. Forum* (2012), 2516–2527.
OculusMedium. 2016. OculusMedium. https://oculus.com/medium/
L. Olsen, F.F. Samavati, M.C. Sousa, and J. Jorge. 2009. Sketch-Based Modeling: A Survey. *Computers & Graphics* 33 (2009). Issue 1.
A. C. Öztireli, G. Guennebaud, and M. Gross. 2009. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. *Computer Graphics Forum* (2009).
PaintLab. 2018. PaintLab VR. http://paintlabvr.com/
Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang. 2015. Flow Aligned Surfacing of Curve Networks. *ACM Trans. Graph.* 34, 4 (2015), 127:1–127:10.
Poly. 2018. Tilt Brush sketches. https://poly.google.com/tiltbrush
E. Sachs, A. Roberts, and D. Stoops. 1991. 3-Draw: a tool for designing 3D shapes. *IEEE Computer Graphics and Applications* 11, 6 (1991), 18–26.
S. Schaefer, J. Warren, and D. Zorin. 2004. Lofting curve networks using subdivision surfaces. *Proc. Symposium on Geometry Processing* (2004), 103.
Steven Schkolne, Michael Pruett, and Peter Schröder. 2001. Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. In *Proc. Human Factors in Computing Systems*. 261–268.
S. Schkolne and P. Schroeder. 1999. *Surface Drawing*. Caltech Department of Computer Science Technical Report CS-TR-99-03.
Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic Drawing of 3D Scaffolds. *ACM Trans. Graph.* 28, 5 (2009), 149:1–149:10.
Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: Shading Concept Sketches Using Cross-Section Curves. *ACM Trans. Graph.* 31, 4 (2012).
ShapeLab. 2018. ShapeLab. https://store.steampowered.com/app/571890/ShapeLab/
Ojaswa Sharma and Nidhi Agarwal. 2016. 3D Surface Reconstruction from Unorganized Sparse Cross Sections. In *Proc. Graphics Interface*. 33–40.
Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. 2004. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Trans. Graph.* (2004), 896–904.
Sketchfab. 2018. Tilt Brush. https://sketchfab.com/tags/tiltbrush
Thomas F. Stahovich, Eric J. Peterson, and Hanlung Lin. 2014. An efficient, classification-based approach for grouping pen strokes into objects. *Computers & Graphics* (2014), 14 – 30.
Shun'ichi Tano, T. Kodera, Takashi Nakashima, I. Kawano, K. Nakanishi, G. Hamagishi, M. Inoue, A. Watanabe, T. Okamoto, K. Kawagoe, K. Kaneko, T. Hotta, and M. Tatsuoka. 2003. Godzilla: Seamless 2D and 3D Sketch Environment for Reflective and Creative Design Work. In *INTERACT*.
Shun'ichi Tano, Shinya Yamamoto, Junko Ichino, Tomonori Hashiyama, and Mitsuru Iwata. 2013. Truly Useful 3D Drawing System for Professional Designer by "Life-Sized and Operable" Feature and New Interaction. In *Human-Computer Interaction – INTERACT 2013*. 37–55.
TiltBrush. 2018. Google TiltBrush. https://tiltbrush.com/
Anil Usumezbas, Ricardo Fabbri, and Benjamin B. Kimia. 2017. The Surfacing of Multiview 3D Drawings via Lofting and Occlusion Reasoning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), 4560–4569.
T. Várady, Alyn Rockwood, and P. Salvi. 2011. Transfinite surface interpolation over irregular n-sided domains. *Computer-Aided Design* iv (2011).
A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13, 2 (1967), 260–269.
Ruimin Wang, Ligang Liu, Zhouwang Yang, Kang Wang, Wen Shan, Jiansong Deng, and Falai Chen. 2016. Construction of Manifolds via Compatible Sparse Representations. *ACM Trans. Graph.* (2016), 14:1–14:10.
Gerold Wesche and Hans-Peter Seidel. 2001. FreeDrawer: A Free-form Sketching System on the Responsive Workbench. In *Proc. Virtual Reality Software and Technology*. 167–174.
Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. 2014. Autocomplete Painting Repetitions. *ACM Trans. Graph.* (2014), 172:1–172:11.
Shiyao Xiong, Juyong Zhang, Jianmin Zheng, Jianfei Cai, and Ligang Liu. 2014. Robust Surface Reconstruction via Dictionary Learning. *ACM Trans. Graph.* (2014), 201:1–201:12.
Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *ACM Trans. Graph.* 33, 4 (2014), 131:1–131:13.
Ming Zou, Michelle Holloway, Nathan Carr, and Tao Ju. 2015. Topology-constrained Surface Reconstruction from Cross-sections. *ACM Trans. Graph.* 34, 4 (2015), 128:1–128:10.