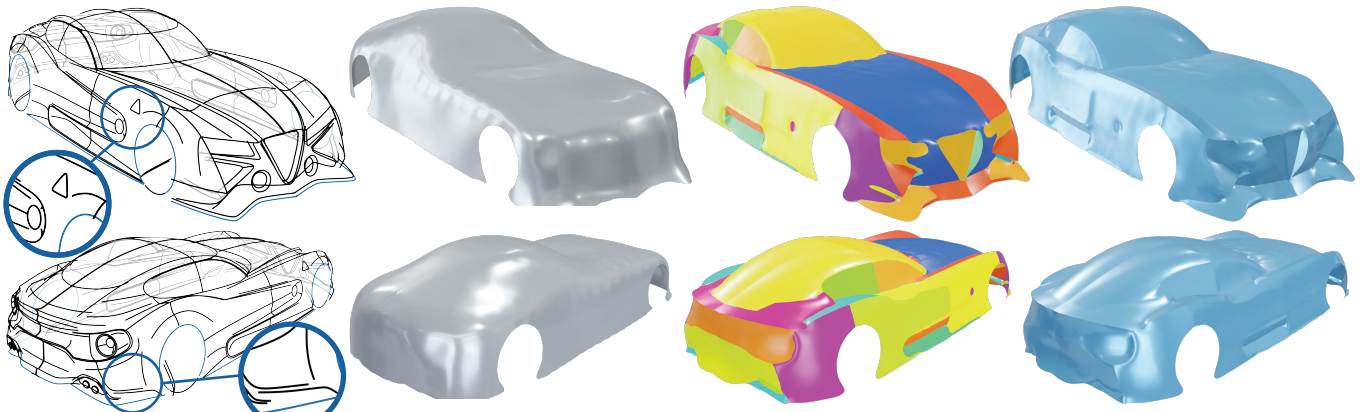
Piecewise-Smooth Surface Fitting onto Unstructured 3D Sketches

EMILIE YU, Inria, Université Côte d'Azur, France
RAHUL ARORA, University of Toronto, Canada
J. ANDREAS BÆRENTZEN, Technical University of Denmark, Denmark
KARAN SINGH, University of Toronto, Canada
ADRIEN BOUSSEAU, Inria, Université Côte d'Azur, France



(a) Unstructured 3D sketch (b) Smooth surfacing [Huang et al. 2019] (c) Joint segmentation & fitting (d) Piecewise-smooth surfacing

Fig. 1. 3D sketches created in Virtual Reality (VR) or with sketch-based modeling systems often depict piecewise-smooth surfaces (a), but lack proper inter-stroke connectivity to detect the individual surface patches, as illustrated in the insets where pen strokes do not intersect precisely, and where detail strokes lie on the imaginary surface without being connected to other strokes. State-of-the-art surfacing algorithms only produce smooth surfaces from such sparse and unstructured 3D data (b). Our algorithm segments such an initial smooth surface into regions aligned with the pen strokes to produce a piecewise-smooth surface that better captures the intended shape. ©James Robbins.

We propose a method to transform unstructured 3D sketches into piecewise smooth surfaces that preserve sketched geometric features. Immersive 3D drawing and sketch-based 3D modeling applications increasingly produce imperfect and unstructured collections of 3D strokes as design output. These 3D sketches are readily perceived as piecewise smooth surfaces by viewers, but are poorly handled by existing 3D surface techniques tailored to well-connected curve networks or sparse point sets. Our algorithm is aligned with human tendency to imagine the strokes as a small set of simple smooth surfaces joined along stroke boundaries. Starting with an initial proxy surface, we iteratively segment the surface into smooth patches joined sharply along some strokes, and optimize these patches to fit surrounding strokes. Our evaluation is fourfold: we demonstrate the impact of various algorithmic parameters, we evaluate our method on synthetic sketches with known ground truth surfaces, we compare to prior art, and we show compelling results on more than 50 designs from a diverse set of 3D sketch sources.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: sketch-based modeling, surfacing, VR sketching

ACM Reference Format:
Emilie Yu, Rahul Arora, J. Andreas Bærentzen, Karan Singh, and Adrien Bousseau. 2022. Piecewise-Smooth Surface Fitting onto Unstructured 3D Sketches. *ACM Trans. Graph.* 41, 4, Article 1 (July 2022), 16 pages. https://doi.org/10.1145/3528223.3530100

Authors' addresses: Emilie Yu, Inria, Université Côte d'Azur, 2004 Route des Lucioles, Sophia-Antopolis, France, emilie.yu@inria.fr; Rahul Arora, University of Toronto, 40 St. George Street, Toronto, Ontario, Canada, mail@rahularora.xyz; J. Andreas Bærentzen, Technical University of Denmark, Anker Engelunds Vej 1 Bygning 101A, Kongens Lyngby, Denmark, janba@dtu.dk; Karan Singh, University of Toronto, 40 St. George Street, Toronto, Ontario, Canada, karan@dgp.toronto.edu; Adrien Bousseau, Inria, Université Côte d'Azur, 2004 Route des Lucioles, Sophia-Antopolis, France, adrien.bousseau@inria.fr.

0730-0301/2022/7-ART1 $15.00
https://doi.org/10.1145/3528223.3530100

1 INTRODUCTION

Drawing, be it on cave walls, paper, or touch screens, has been a quintessential tool of visual communication for millennia. Beyond the established importance of drawing on surfaces, sketching mid-air in AR/VR environments, like waving a magic wand, provides artists the unprecedented freedom to draw directly in immersive 3D [Adobe 2020; Google 2016; GravitySketch 2017; Smoothstep 2021]. Such unstructured 3D curve sketches, while beautiful to behold and effectively perceived by viewers, are non-trivial to transform into 3D surface models for further design communication and processing. We present, to our knowledge, the first approach to transform such unstructured sketches into piece-wise smooth 3D surface models that preserve salient sketched features (Fig. 1).

Our method is motivated by several key characteristics of 3D sketches, highlighted as insets in Fig. 1(a).

— Artists often depict piecewise-smooth surfaces by strategically placing strokes at sharp surface discontinuities, consistent with traditional sketching practices of representing an object through its feature curves – ridges and valleys [Cole et al. 2008; Gryaditskaya et al. 2019].
— While the main patches that form the envisioned surface are often delimited by strokes, other strokes lie within individual patches to depict sub-parts and decorative details.
— 3D sketches are often imprecise, exhibiting over-sketching or gaps and missing intersections between strokes.

While the relative importance of these characteristics in 3D sketches may vary by artist and the drawing tool used, they underlie both AR/VR sketching and traditional sketch-based 3D modeling systems [Bae et al. 2008; Kim and Bae 2016; Xu et al. 2014], making our approach well-suited in general to surfacing any sketches, that are provided as an unstructured collection of 3D curves.

Existing surfacing methods only partly account for these characteristics. Point cloud surfacing algorithms target densely sampled surfaces and fail on sparse stroke clouds [Hoppe et al. 1992; Kazhdan et al. 2006] or cannot capture sharp features of the sketch (Fig. 1b).

Methods dedicated to 3D drawings focus on well-connected curve networks [Abbasinejad et al. 2011; Bessmeltsev et al. 2012; Orbay and Kara 2012; Sadri and Singh 2014; Zhuang et al. 2013], where surface patches are bounded by closed cycles of curve segments. Due to their strong requirements on curve network topology, such methods cannot process the imprecise, unstructured stroke clouds we target.

To reconstruct piecewise-smooth surfaces from inaccurate 3D sketches, we must jointly determine which parts of the sketch correspond to different surface patches, and recover the geometry of these patches away from the strokes.

A first challenge is to determine where surface patches should lie in the empty space between the strokes. This task is particularly ambiguous for sparse sketches, since the inter-stroke Euclidean proximity does not necessarily denote geodesic proximity on the envisioned surface. We reduce this ambiguity by complementing the input sketch with a low-fidelity *proxy surface*, which we obtain by applying the smooth point cloud surfacing method of Huang et al. [2019], or in cases where this automatic method fails, by asking users to create a simple proxy with low-poly modeling tools (see Section 5). The proxy surface provides us with a manifold domain on which we project nearby strokes, such that our problem of locating the surface patches amounts to *segmenting* the proxy into regions roughly bounded by strokes.

Our second challenge is to shape each proxy region into a surface patch that represents well the geometry of the strokes that project in that region, which is especially difficult when the strokes themselves are sparse and approximate. We address this challenge by representing each patch as an implicit surface defined as the zero level-set of a low degree polynomial – which we refer to as an *implicit polynomial surface*. Such implicit surfaces offer multiple benefits in our context. They have infinite support, allowing the surface regions they capture to grow or shrink arbitrarily as the algorithm progresses, and they are fast to evaluate and fit to stroke points.

Equipped with the concepts of a proxy surface and implicit polynomial surface patches, we cast finding the piecewise-smooth surface that best represents the sketch as a *multi-model fitting* problem [Isack and Boykov 2012]. In a nutshell, our algorithm alternates between refining a segmentation of the proxy into regions well represented by a set of implicit surfaces, and improving the implicit surfaces by fitting them over the strokes within each region. After convergence, we generate the final 3D triangle mesh by projecting the proxy surface onto the zero-level sets of the implicit polynomial surfaces, effectively recovering sharp features at the intersections between patches.

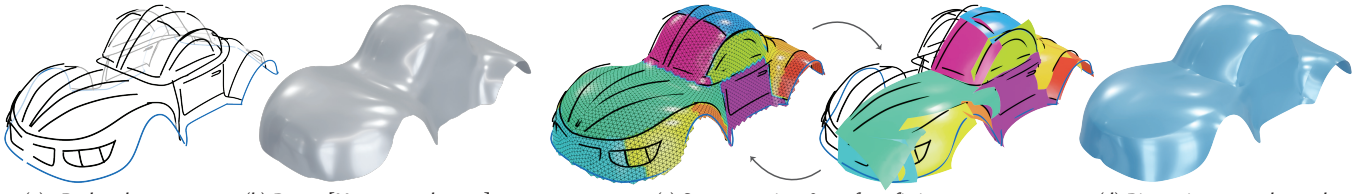
While our method automatically produces piecewise-smooth surfaces aligned with the input strokes, it also supports user indications to further improve the quality of the surface; for instance, to force the creation or removal of surface discontinuities. We also rely on user indications to discard strokes that are not supported by our approach, notably internal strokes that lie inside the outermost depicted surface, small disconnected parts, and so-called *skeletal strokes* that depict thin cylindrical features (see Fig. 15).

In summary, we present the first surfacing method for 3D sketches that produces piecewise-smooth meshes while being oblivious to stroke connectivity. We evaluate our method by surfacing more than 50 sketches created with a variety of VR and sketch-based modeling systems, by comparing to surfacing methods tailored to well-connected curve networks, by measuring deviation from ground truth surfaces from which synthetic sketches were generated, and by studying the impact of the main components of our algorithm. Finally, we provide an interactive visualization of all our results as supplemental materials for close inspection.

2 RELATED WORK

Surfacing curve networks. The problem of surfacing 3D sketches emerged with the advent of practical user interfaces, algorithms and devices to create such sketches. Early 2D interfaces deduce the depth of the strokes from the intersections they form with other strokes [Schmidt et al. 2009; Xu et al. 2014] or with sketching planes [Bae et al. 2008], effectively producing well-connected *curve networks* by construction. As a consequence, a number of surfacing algorithms strongly rely on the connectivity of the curve network to identify closed cycles delimiting surface patches [Abbasinejad et al. 2011; Orbay and Kara 2012; Sadri and Singh 2014; Zhuang et al. 2013]. Each such patch can then be surfaced by propagating geometric information from the boundary curves [Bessmeltsev et al. 2012; Pan et al. 2015; Stanko et al. 2016]. Curve connectivity information provides strong geometric hints, since surface normals can be estimated at each intersection – as done by Pan et al. [2015] to detect sharp features and determine which curves are flow lines. However, the reliance on accurate curve network topology prevents these methods to work on raw, *unstructured stroke clouds* as typically produced via freehand VR sketching [Google 2016; GravitySketch 2017; Smoothstep 2021].

Recent studies have shown that precise sketching in VR is more challenging than in 2D due to the lack of a supporting surface for the



(a) 3D sketch (b) Proxy [Huang et al. 2019] (c) Segmentation & surface fitting (d) Piecewise-smooth result

Fig. 2. *Overview of our method.* We take as input a 3D sketch (a) and a proxy mesh (b), for example obtained with *VIPSS* [Huang et al. 2019]. We iteratively improve a segmentation of the proxy mesh (c, left) and parameters of a set of surface models (c, right) to obtain a decomposition of the surface into smooth patches that approximate the input strokes well. We represent each surface model as the zero level set of an implicit polynomial, which offers infinite spatial support and a good balance between expressivity and complexity. Finally, we project the vertices of the proxy mesh onto the surface models to recover the final piecewise-smooth surface with sharp features (d). When appropriate, we trim the proxy mesh to match user-annotated boundary strokes (a, blue), and we leverage sketch symmetry by surfacing only half of the sketch and mirroring the result.

hand and the need for finer motor control to position strokes in 3D [Arora et al. 2017; Machuca et al. 2019]. Automatic snapping can help correct for inaccuracy [Machuca et al. 2018; Yu et al. 2021a,b], but such assistance can be detrimental to user creativity, as mentioned by users of the recent *CASSIE* system [Yu et al. 2021a] who criticized being forced to think of their designs in terms of curve networks.

Processing 3D sketches to form well-connected curve networks is both challenging and often undesirable. First, many strokes in 3D sketches are not meant to connect to others – see the details on *author1_bulbasaur* (Fig. 16), and on the car (Fig. 1 insets). Simply ignoring such disconnected strokes can alter the original design intent (Fig. 12, red strokes). Secondly, 3D sketches may exhibit oversketching and imprecisions (*vr_controller* Fig. 16), which make forming clean curve networks from 3D strokes a challenging problem, akin to 2D vectorization. We instead propose an approach that is oblivious to stroke connectivity, achieving high robustness to inaccuracy and to detail strokes that lie on the envisioned surface but are not connected to other strokes.

Our work also differs from interactive systems designed to model 3D surfaces by sketching in 2D [Dvorožňák et al. 2018; Li et al. 2017; Nealen et al. 2007]. Users of these systems draw in dedicated interfaces and provide annotations of surface discontinuities. In contrast, we take as input 3D sketches created with a variety of tools, and we propose a multi-model fitting algorithm to automatically locate sharp features where and only where they are needed.

Surfacing point clouds. The 3D sketches we consider can easily be converted to point clouds by sampling points along each stroke, which would allow leveraging the wealth of methods developed for surfacing such unstructured 3D data [Berger et al. 2017]. Unfortunately, most existing methods have been designed to process dense point clouds acquired using 3D scanning technology [Hoppe et al. 1992; Kazhdan et al. 2006], and are doomed to fail on 3D sketches that only provide a very sparse, non-uniform sampling of the envisioned surface. While some methods are robust to missing data, they only fix holes that are relatively small compared to the scale of the overall surface [Hornung and Kobbelt 2006], or guide surface completion by fitting geometric primitives on otherwise dense parts of the point cloud [Schnabel et al. 2009; Tagliasacchi et al. 2009]. In contrast, 3D sketches are dominated by large holes, and only contain 3D information along thin, 1-dimensional subspaces.

The recent *VIPSS* algorithm of Huang et al. [2019] achieves impressive resilience to sparsity and non-uniformity of the point cloud, and has even been demonstrated on unstructured stroke clouds similar to our target sketches. But this robustness is obtained thanks to a global smoothness energy that misses sharp surface discontinuities. Nevertheless, we use *VIPSS* to initialize our method, and focus on the problem of segmenting its result into individual patches forming a more faithful reconstruction of the input sketch.

Our approach relates to algorithms that recover piecewise-smooth surfaces from point clouds by identifying locally-smooth patches and by detecting sharp discontinuities as the intersections of these patches [Fleishman et al. 2005; Huang et al. 2013; Jenke et al. 2008]. However, these methods rely on a dense sampling of the surface *away* from sharp features, while 3D sketches are mostly empty in such regions and are only densely sampled *along* feature curves.

We also take inspiration from methods that reconstruct or approximate 3D shapes as a collection of parametric geometric primitives, such as planes [Cohen-Steiner et al. 2004; Monszpart et al. 2015; Pham et al. 2016], cones, cylinders and spheres [Li et al. 2011; Wu and Kobbelt 2005], generalized cylinders [Zhou et al. 2015], and quadric surfaces [Yan et al. 2012]. Similarly to these methods, we cast the discovery of representative surface patches as a multi-model fitting problem. Our originality is to leverage specifics of 3D sketches to guide this optimization, notably by encouraging large, uniform patches bounded by the input strokes, and by representing free-form surfaces as 4th-order implicit polynomial surfaces.

Note that the resulting segmentation is purely driven by geometry and does not carry any semantic meaning other than when semantic parts happen to be well-represented by different smooth patches – eg, the hood of the car and the windshield (Fig. 1c).

Surfacing stroke clouds. Our work focuses on *sparse* 3D sketches, which contrast with the *dense* VR paintings targeted by Rosales et al. [2019] that are created by covering the surface with large, overlapping ribbons that provide position and orientation samples all over the surface [Google 2016; Rosales et al. 2021].

In the computer vision community, a few methods have been proposed to reconstruct *curve clouds* by matching edges in a multi-view stereo algorithm [Fabbri and Kimia 2010]. Usumezbas et al. [2017] proposed a surfacing algorithm dedicated to such unstructured 3D data, where candidate surface patches are lofted between pairs of

curves. But this method largely relies on the availability of multiple photographs of the shape to select valid candidate patches based on occlusion reasoning.

Closest to our problem statement, Batuhan Arisoy et al. [2012] surface sparse and imprecise 3D sketches by smoothly deforming an initial low-fidelity surface of correct topology, using a discrete guidance vector field that points towards the closest stroke point. This approach produces globally smooth surfaces and requires user intervention to specify strokes that should be inserted into the mesh as sharp edge polylines (see Fig. 17 and 18 in their paper for a visual comparison of their results on similar sketches to ours). In contrast, our multi-model fitting formulation produces piecewise-smooth surfaces automatically.

3 OVERVIEW

Fig. 2 illustrates the main steps of our method, which takes as input a 3D sketch, along with an approximate *proxy surface* obtained with an automatic surfacing algorithm [Huang et al. 2019] or an existing low-poly modeling tool (see the accompanying video for a demonstration of this workflow). After projecting the sketch onto the proxy, our goal is to segment the proxy into regions, each associated with a smooth *surface model*, such that the resulting piecewise-smooth surface satisfies the following desiderata.

- **Reproducing stroke geometry.** The surface models should run close to the input strokes, both for strokes that depict sharp surface discontinuities and for strokes that depict details within smooth areas.
- **Aligning patch boundaries with strokes.** The boundary between neighboring regions should lie along strokes that depict sharp surface discontinuities, yet not all strokes in the sketch depict a discontinuity.
- **Keeping the reconstruction simple.** The surface should be composed of a small number of smooth patches, rather than many intricate patches that would overfit to inaccuracy in the input strokes.

We formulate these competing requirements as energy terms in an optimization. A first energy term measures the distance between each stroke and the surface model it is assigned to. A second term measures the smoothness of the segmentation away from the strokes, encouraging the transitions between models to occur along strokes. This smoothness term also penalizes small, isolated regions, which contributes to satisfy our third desiderata. Finally, a third term measures the complexity of the reconstruction by counting the degrees of freedom of the models used.

While each of our terms has an intuitive interpretation, their combination yields a challenging optimization problem that combines discrete variables (which surface model should be assigned to which region of the proxy) and continuous variables (parameters of these models). We tackle this challenge by expressing our problem within *PEARL*, a general algorithm to solve multi-model fitting problems [Isack and Boykov 2012]. In our context, the algorithm alternates between optimizing the discrete variables describing the segmentation while keeping the model parameters fixed, and optimizing for the continuous model parameters while keeping the segmentation fixed (Fig. 2c).

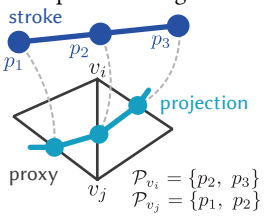
After convergence, we obtain a segmented proxy, as well as a set of surface models represented by implicit polynomial surfaces. The last step of our method aims at converting this representation into a triangle mesh suitable for downstream applications (Fig. 2d). Since the proxy provides us with a mesh of correct topology relatively close to the surface models, we recover the final surface by projecting this mesh onto the zero level-sets of the polynomials.

4 METHOD

We now formulate the main terms of our optimization problem, before describing how we solve this problem by alternating segmentation and fitting. In what follows, we use the terms *points* and *segments* to refer to the stroke polylines, and the terms *vertices* and *edges* to refer to the triangles of the proxy mesh.

4.1 Problem formulation

While our segmentation algorithm relies on the proxy mesh as a computational domain, several of our energy terms are defined as functions of the stroke points. To reason about stroke geometry on the proxy domain, we project each stroke to its nearest location on the proxy and associate it with nearby mesh elements. Specifically, we detect intersections between the projected stroke segments and triangle edges, and associate the corresponding stroke points with the two vertices of the intersected edges. Thanks to this association, our segmentation algorithm assigns one surface model per mesh vertex, yet can fit several surface models to a single stroke point – which is critical for positioning boundaries of surface patches along strokes. Denoting $\mathcal{P}$ the set of all stroke points, we denote $\mathcal{P}_v \subset \mathcal{P}$ the subset associated with a given mesh vertex $v \in \mathcal{V}$, which can contain zero or many stroke points (see inset). This association is built by a closest-point projection of the strokes onto the proxy surface (see Appendix A for more details).



Our goal is to assign each mesh vertex to a surface model f_α, associated with the label $\alpha \in \mathcal{L}$. We denote the corresponding vertex labeling as $l\colon \mathcal{V} \to \mathcal{L}$. Furthermore, we denote as $\boldsymbol{\theta}_\alpha$ the vector of real-valued parameters of the surface model f_α. Our surface model representation, detailed in Section 4.2, is the zero level-set $\mathcal{Z}(f_\alpha)$ of an implicit polynomial. Finally, we denote as $\boldsymbol{\Theta} = \bigoplus_{\alpha\in\mathcal{L}} \boldsymbol{\theta}_\alpha$ the concatenation of the parameter vectors $\boldsymbol{\theta}_\alpha$ for all $\alpha \in \mathcal{L}$.

Given these definitions, we cast our problem as finding the labeling and the associated model parameters that minimize

$$\begin{aligned} E_{\mathcal{L}}(l, \boldsymbol{\Theta}) = {} & E_{\text{fidelity}}(l, \boldsymbol{\Theta}) \\ & + w_{\text{smoothness}} \cdot E_{\text{smoothness}}(l) \\ & + w_{\text{simplicity}} \cdot E_{\text{simplicity}}(l), \end{aligned} \tag{1}$$

where E_{fidelity}, $E_{\text{smoothness}}$, and $E_{\text{simplicity}}$ capture the three desiderata listed in Section 3. Note that the number of labels required for a given sketch is also unknown, since we do not know how many smooth patches are necessary to faithfully represent the 3D surface a priori. We next describe each of the three energy terms.

Fidelity to input strokes. We seek a piecewise-smooth surface that reproduces well the input stroke geometry. We measure this

property by summing over all mesh vertices v the distance between their surface model $\mathcal{Z}\left(f_{l(v)}\right)$ and each of their associated stroke points $\mathbf{p} \in \mathcal{P}_v$:

$$E_{\text{fidelity}}(l, \boldsymbol{\Theta}) = \frac{1}{\epsilon^2 N_p} \sum_{v \in \mathcal{V}} \sum_{\mathbf{p} \in \mathcal{P}_v} \text{dist}\left(\mathcal{Z}\left(f_{l(v)}\right), \mathbf{p}\right)^2, \quad (2)$$

where the normalization term N_p is the total number of stroke points involved in the summation. We detail in Section 4.2 how to compute $\text{dist}\left(\mathcal{Z}(f), \mathbf{p}\right)^2$ efficiently. The scale factor ϵ controls the sensitivity of E_{fidelity} to small deviations of the strokes from the surface models. We experimentally set this parameter to 1% of the input sketch's bounding-box diagonal for all results presented in the paper, but we show the impact of alternate values in Fig. 9.

Smoothness of the labeling. Our smoothness term seeks to concentrate changes of labels along strokes, such that sharp surface discontinuities appear at these locations when transitioning from one surface model to another. Inspired by classic work on edge-aware image segmentation [Boykov and Jolly 2001], we encourage neighboring vertices to share the same label, unless they are separated by a stroke. We achieve this goal by assigning a weight w^e to each edge $e \in \mathcal{E}$ of the mesh, using a low weight $w^e = 1$ if the edge is crossed by a projected stroke segment, and a high weight $w^e = 100$ otherwise. Here we assume that all edges have approximately the same length, see Appendix C for a scaling term in the case of anisotropic meshes. We then define the smoothness energy:

$$E_{\text{smoothness}}(l) = \frac{1}{W} \sum_{\{u,v\} \in \mathcal{E}} w^{uv} \left(1 - \delta\left(l(u), l(v)\right)\right), \quad (3)$$

with δ being the Kronecker delta function, $\delta(i, j) = 1$ if $i = j$, $\delta(i, j) = 0$ otherwise. The normalization term W is the sum of all edge weights. We set $w_{\text{smoothness}} = 10$.

Surface simplicity. To be resilient to the typical inaccuracy of 3D sketches [Arora et al. 2017], we encourage the surface reconstruction to be as simple as possible. We measure the complexity of a solution as the total number of surface parameters involved. Denoting $\dim(\boldsymbol{\theta}_\alpha)$ the number of parameters of a surface model f_α, we define the simplicity energy as:

$$E_{\text{simplicity}}(l) = \frac{1}{D} \sum_{\alpha \in \{l(v) \mid v \in \mathcal{V}\}} \dim(\boldsymbol{\theta}_\alpha), \quad (4)$$

where we set $D = 35$, the number of parameters of a degree 4 model, as a normalizer. Note that by summing over assigned models only, this energy also pushes for using a small number of models to explain the 3D sketch, to the point where a single model can be used to represent multiple non-adjacent regions of the surface. For all results, we set $w_{\text{simplicity}} = 0.01$ (and demonstrate varying values in Fig. 8).

4.2 Energy minimization

Algorithm 1 adapts the general *PEARL* multi-model fitting algorithm [Isack and Boykov 2012] to estimate the labeling l and surface model parameters $\boldsymbol{\Theta}$ that locally minimize Equation 1. After an initialization phase, the algorithm alternates between improving the labeling and refining the model parameters, and terminates when the labeling l no longer changes. We next detail each step.

Algorithm 1: PEARL algorithm [Isack and Boykov 2012] applied to our problem

(0) Initialize variables
Propose m vectors of parameters $\{\boldsymbol{\theta}_1^0, \cdots, \boldsymbol{\theta}_m^0\}$ as initial surface models
$\boldsymbol{\Theta} \longleftarrow \bigoplus_{i=1}^m \boldsymbol{\theta}_i^0$ // Concatenated surface parameters
$\mathcal{L} \longleftarrow \{1, \ldots, m\}$ // Set of existing labels
$l \longleftarrow l\colon v \in \mathcal{V} \mapsto 1$ // Initialize with single label
repeat
  $l_{\text{prev}} \longleftarrow l$
  (1) Optimize labeling
  $l \longleftarrow \alpha$-expansion [Boykov et al. 2001] $\forall \alpha \in \mathcal{L}$ to optimize previous labeling l_{prev} for energy (Equation 1), given the current parametric surface models $\boldsymbol{\Theta}$
  $\mathcal{L} \longleftarrow \{\alpha = l(v) \mid v \in \mathcal{V}\}$ // Remove unassigned labels
  $\boldsymbol{\Theta} \longleftarrow \bigoplus_{\alpha \in \mathcal{L}} \boldsymbol{\theta}_\alpha$ // And the corresponding inactive models
  (2) Optimize parametric surface models
  for $\alpha \in \mathcal{L}$ **do**
    $\mathcal{V}_\alpha \longleftarrow \{v \in \mathcal{V} \mid l(v) = \alpha\}$ // Vertices with label α
    $\boldsymbol{\theta}_\alpha \longleftarrow$ Best fit for points in $\mathcal{P}_\alpha = \bigcup_{v \in \mathcal{V}_\alpha} \mathcal{P}_v$
  end
  (3) Propose new models
  Propose new labels $\mathcal{L}_{\text{new}}$ and models $\boldsymbol{\Theta}_{\text{new}}$
  $\mathcal{L} \longleftarrow \mathcal{L} \cup \mathcal{L}_{\text{new}}$
  $\boldsymbol{\Theta} \longleftarrow \boldsymbol{\Theta} \cup \boldsymbol{\Theta}_{\text{new}}$
until $l = l_{\text{prev}}$;

Segmentation. At each iteration, we first optimize the current labeling l while keeping all surface model parameters $\boldsymbol{\Theta}$ fixed. We perform α-expansion [Boykov et al. 2001] – which finds the optimal change of labeling where some nodes are assigned a label α – for every label $\alpha \in \mathcal{L}$ to efficiently find a local minimum of $E_{\mathcal{L}}$. [1] While $E_{\text{smoothness}}$ and $E_{\text{simplicity}}$ are straightforward to compute for a given labeling l, E_{fidelity} requires computing the distance $\text{dist}\left(\mathcal{Z}(f), \mathbf{p}\right)^2$ between each surface model and the corresponding stroke points. Since the exact distance between a point and the zero level-set of an implicit polynomial surface cannot be computed exactly with a direct method, we employ the first-order approximation proposed by Taubin [1993],

$$\text{dist}\left(\mathcal{Z}(f), \mathbf{p}\right)^2 \approx \frac{f(\mathbf{p})^2}{\|\nabla f(\mathbf{p})\|^2}, \quad (5)$$

which can be computed in linear time with respect to the number of stroke points $\mathbf{p}$. Since the resulting labeling might leave some labels unassigned, we update $\mathcal{L}$ to only keep the selected models.

Model representation and fitting. After each segmentation step, we improve the surface model of each label $\alpha \in \mathcal{L}$ by optimizing its parameters $\boldsymbol{\theta}_\alpha$ to best fit the stroke points $\mathbf{p} \in \mathcal{P}_\alpha$ associated with the vertices $v \in \mathcal{V}_\alpha$. We define each surface model as the zero

[1] We use the C++ multi-label optimization library of Delong et al. [2012] https://vision.cs.uwaterloo.ca/code/

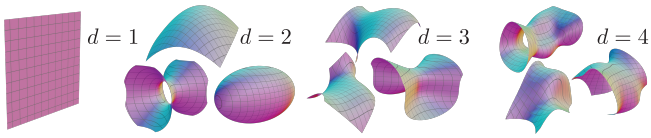


Fig. 3. We represent surface patches as zero level-sets of implicit polynomials of degree up to 4. Low-degree polynomials can represent planes, developable and simple doubly-curved surfaces, while higher-degree polynomials can represent complex freeform surfaces.

level-set of a polynomial $f\colon \mathbb{R}^3 \to \mathbb{R}$,

$$\mathcal{Z}(f) = \{(x, y, z) \mid f(\boldsymbol{\theta}_\alpha; x, y, z) = 0\},$$

with f expressed as

$$f(\boldsymbol{\theta}_\alpha; x, y, z) = \mathbf{X}(x, y, z)^T \boldsymbol{\theta}_\alpha,$$

where $\boldsymbol{\theta}_\alpha = \left[\theta^1 \dots \theta^t\right]^T$ is the $t \times 1$ vector of coefficients and $\mathbf{X}(x, y, z)$ a $t \times 1$ vector of monomials,

$$\mathbf{X}(x, y, z) = \begin{bmatrix} 1 & x & y & z & x^2 & \dots & x^d & y^d & z^d \end{bmatrix}^T.$$

To avoid overfitting to approximate strokes, we limit the expressivity of the surface models by keeping their degree d low. In practice we set d to be at most 4, which corresponds to $t = 35$ parameters. We describe at the end of this section how we introduce lower-degree models at the end of each iteration. Fig. 3 shows how surface models of different degrees capture shapes of varying complexity.

Given the set of stroke points $\mathbf{p} = (x, y, z) \in \mathcal{P}_\alpha$, we seek the vector of coefficients $\boldsymbol{\theta}_\alpha$ that minimize the geometric error between the stroke points and the zero level set of the function f. However, the geometric distance from a point to the zero level-set of a polynomial has no closed form expression, and minimizing it requires an iterative approach [Ahn et al. 2002]. Similarly, minimizing Equation 5 is costly since its derivatives are non-linear with respect to the model parameters θ. A common alternative consists in minimizing the *algebraic distance* to the implicit polynomial surface, which yields a linear least-squares regression:

$$\arg\min_{\boldsymbol{\theta}} \sum_{\mathbf{p} \in \mathcal{P}_\alpha} \left(\mathbf{X}(\mathbf{p})^T \boldsymbol{\theta}\right)^2.$$

As shown by Tasdizen et al. [2000], this minimization problem can be made more stable by also encouraging the gradient of the implicit function to align with prescribed normals at a set of points $\mathcal{P}'_\alpha$:

$$\arg\min_{\boldsymbol{\theta}} \sum_{\mathbf{p} \in \mathcal{P}_\alpha} \left(\mathbf{X}(\mathbf{p})^T \boldsymbol{\theta}\right)^2 + \mu \sum_{\mathbf{p} \in \mathcal{P}'_\alpha} \left(1 - \mathbf{n}_\mathbf{p} \cdot \nabla X(\mathbf{p})^T \boldsymbol{\theta}\right)^2,$$

where $\nabla X(\mathbf{p})$ is a $t \times 3$ matrix denoting the gradient of the monomial vector $X(\mathbf{p})$, and $\mathbf{n}_\mathbf{p}$ denotes the normal vector at the point $\mathbf{p}$. Since we do not have normal information at the stroke points, we encourage alignment with the proxy mesh normals at the mesh vertices. Furthermore, we exclude normals from vertices that lie close to the strokes, since strokes often denote surface discontinuities for which the smooth proxy mesh is only a crude approximation. We thus define the set of points for the alignment term as

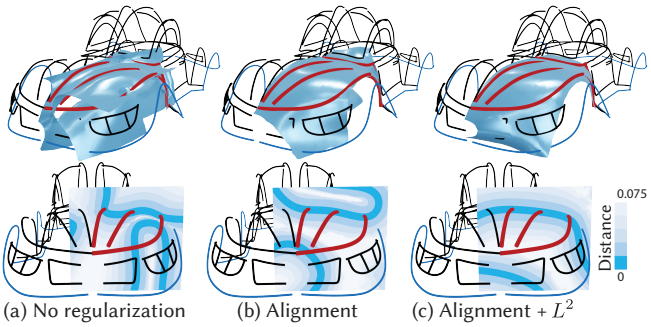


Fig. 4. Fitting an implicit polynomial surface to a subset of strokes (thick red strokes). We visualize a slice of the unsigned distance field in the second row (distances consider a sketch bounding box diagonal of unit length). (a) Minimizing the algebraic distance without regularization yields level sets with spurious turns close to the data points. (b) By encouraging the gradient to align with the proxy normals [Tasdizen et al. 2000], the implicit function is well behaved in a neighborhood around the data points. (c) Adding L^2 regularization avoids overfitting to noise in the data points and yields a more stable zero level-set.

$\mathcal{P}'_\alpha = \{v \in \mathcal{V}_\alpha \mid \mathcal{P}_v = \emptyset\}$. Finally, we also include an L^2 regularization term to penalize large polynomial coefficients [Tasdizen et al. 2000], yielding:

$$\hat{\boldsymbol{\theta}}_\alpha = \arg\min_{\boldsymbol{\theta}} \sum_{\mathbf{p} \in \mathcal{P}_\alpha} (\mathbf{X}(\mathbf{p})^T \boldsymbol{\theta})^2 + \mu \sum_{\mathbf{p} \in \mathcal{P}'_\alpha} (1 - \mathbf{n}_\mathbf{p} \cdot \nabla X(\mathbf{p})^T \boldsymbol{\theta})^2 + \lambda \|\boldsymbol{\theta}\|^2, \tag{6}$$

with $\mu = 0.1$ and $\lambda = 1$. Fig. 4 illustrates the impact of each of these regularization terms. We provide additional details on how to solve this minimization problem in Appendix B.

Note that $\hat{\boldsymbol{\theta}}_\alpha$ does not necessarily decrease E_{fidelity} (Equation 2) given a fixed labeling, since it minimizes the algebraic distance augmented with regularization terms – and not the geometric distance. To guarantee convergence to a local minimum of Algorithm 1, we only update the parameter vector $\boldsymbol{\theta}_\alpha$ if this yields a decrease of E_{fidelity} for the vertices labeled as α. If not, we keep the previous parameter vector and introduce the new parameter vector as a different model.

Initialization. Starting the algorithm with a good initial guess leads to higher quality solutions (see evaluation in Fig. 7). We obtain this initial guess by leveraging the observation that some of the strokes depict boundaries of smooth surface patches. Assuming that this is the case for all strokes, we compute an over-segmentation of the proxy surface by running spectral clustering [Shi and Malik 2000] on the mesh, using the same edge weight w^e as in $E_{\text{smoothness}}$. We then fit an implicit polynomial surface to the strokes associated to the vertices of each cluster to obtain our initial set of models.

Spectral clustering requires the number of clusters to be specified in advance. Yet, the appropriate number varies among sketches, depending of their level of details. We address this challenge by running spectral clustering with different numbers of clusters (in practice, 20, 30, 40, and 50 clusters), which produces surface models of different scales.

For this initial surface fitting, we boost the regularization weights μ and λ by a factor 10 to limit complexity of the models and prevent the appearance of sub-optimal large models across sharp features.

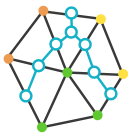
New model proposal. We end each iteration by proposing new surface models, which helps decrease the energy $E_{\mathcal{L}}$ in subsequent iterations [Isack and Boykov 2012]. First, we introduce models of lower degree by fitting a polynomial of degree $d-1$ to the set of stroke points $\mathcal{P}_\alpha$ for each active model f_α of degree $d > 1$. Second, we propose new models by merging pairs of neighboring regions and fitting a polynomial to the union of their stroke points. We prioritize merging regions that are separated by edges with a high weight w^e, as this strategy is more likely to yield a decrease in $E_{\text{smoothness}}$. In practice, we select three pairs of regions to be merged per iteration.

4.3 Extracting the surface mesh

The outcome of our multi-model fitting algorithm is a set of implicit surfaces that extend infinitely beyond the strokes they capture (Fig. 2c). Our goal is now to extract a triangular mesh from this arrangement of surfaces. One option to reach this goal would be to trim each surface along its intersection with other surfaces, for example by meshing the isosurfaces, resolving intersections [Cherchi et al. 2020], and then selecting the set of patches that best cover the strokes while forming a closed manifold [Bauchet and Lafarge 2020; Du et al. 2021]. Unfortunately, the halfspaces defined by our implicit polynomial surfaces are not guaranteed to bound the desired shape, as two surfaces that describe adjacent regions might not intersect. Even when neighboring surfaces do intersect, they can be nearly tangential to each other, which makes the detection of intersections prone to numerical inaccuracies. We bypass all these difficulties by leveraging the proxy mesh, as it provides us with a good estimate of the mesh we are looking for. We formulate our problem as the local minimization of an energy that projects the mesh towards the implicit surfaces assigned to each region, regularized by a smoothness term:

$$E_{\text{mesh}} = E_{\text{project}} + E_{\text{regularize}} \tag{7}$$

Insertion of segmentation boundaries. Before attempting to minimize Equation 7, we first need to insert edges that split the mesh triangles crossed by segmentation boundaries, so that these boundaries can emerge as sharp surface discontinuities in the optimized mesh. The inserted vertices (blue circles in inset) inherit the two or three labels of the regions they separate, yielding a multi-labeling function that we denote $\hat{l}$.



Projection to the implicit surfaces. We project each mesh vertex onto its associated surface patches by minimizing its distance to the zero-level sets of the corresponding polynomials:

$$E_{\text{project}}(\mathcal{V}) = \sum_{v \in \mathcal{V}} \sum_{\alpha \in \hat{l}(v)} \frac{1}{|\hat{l}(v)|} \operatorname{dist}\left(\mathcal{Z}(f_\alpha), v\right)^2, \tag{8}$$

where we use the first-order approximation from Equation 5 to compute the distance.

Regularization. Minimizing Equation 8 sometimes pulls neighboring vertices in opposite directions, yielding a distorted surface. We achieve smoother results by regularizing the optimization with a 2D Laplacian term on the mesh triangles, and a 1D Laplacian to favor smooth sharp feature curves:

$$E_{\text{regularize}}(\mathcal{V}) = \gamma_{2D} \sum_{v \in \mathcal{V}} \mathbf{w}_v ||\Delta_{2D} v||^2 + \gamma_{1D} \sum_{v \in \mathcal{V}_{\text{seams}}} ||\Delta_{1D} v||^2, \tag{9}$$

where $\mathcal{V}_{\text{seams}}$ is the set of vertices inserted along segmentation boundaries, and Δ_{2D} and Δ_{1D} denote the discrete graph Laplacian operator for triangles and edges respectively [Nealen et al. 2006]. We set $\gamma_{2D} = 100$ and $\gamma_{1D} = 500$.

The weight $\mathbf{w}_v$ controls the strength of the surface regularization, which we want to vanish along segmentation boundaries that correspond to sharp surface discontinuities. We set $\mathbf{w}_v = 1$ for vertices away from segmentation boundaries, and $\mathbf{w}_v = 0$ for boundary vertices that are shared by more than two segmentation regions. For vertices that lie in-between two regions, we adjust their weight according to the angle formed by the implicit surface normals on each side of the boundary, with $\mathbf{w}_v = 0$ when the angle is greater than $\pi/3$, $\mathbf{w}_v = 1$ when the angle is smaller than $\pi/8$, and $\mathbf{w}_v$ varies linearly in-between. Effectively, this preserves sharpness of the boundary between regions that intersect sharply, while favoring smoothness elsewhere.

In Appendix D, we describe the special case of open meshes, which necessitate an additional term to prevent the mesh boundary to contract under the effect of the regularization.

We minimize E_{mesh} with L-BFGS, by precomputing the gradient matrices for the quadratic terms.

5 IMPLEMENTATION DETAILS

Proxy creation. Our method requires a proxy surface that has the same topology as the desired result, and that lies close to the envisioned surface, so that projecting the strokes onto the proxy yields the same embedding as it would on the envisioned surface.

The automatic *VIPSS* surfacing algorithm [Huang et al. 2019] produced suitable proxies for many of our results. We additionally re-meshed the output of *VIPSS* to obtain a high-resolution, uniform mesh [Jakob et al. 2015]. However, *VIPSS* sometimes fails to surface complex sketches that exhibit many sharp features, a very sparse sampling, or thin features. In these cases, we experimented with multiple casual modeling tools to create our proxies in just a few minutes, including low-poly modeling with *Blender* [Blender 2020] (Fig. 5c), rough VR sculpting with *Adobe Medium* [Adobe 2020] (Fig. 5d), assembly of simple geometric primitives (Fig. 17 *author1_bulbasaur*). We provide an example modeling session in our accompanying video.

Manually-created proxy meshes are often approximate, and would not yield satisfying results if projected directly on the implicit surface models (Section 4.3). We obtain better results by first attracting the proxy towards the strokes, which we achieve by projecting the stroke points onto the proxy and by using these points as control vertices for a *Least-Squares mesh* defined by the initial proxy connectivity [Sorkine and Cohen-Or 2004]. Fig. 5 shows how this attraction brings approximate proxy meshes much closer to the intended surface.

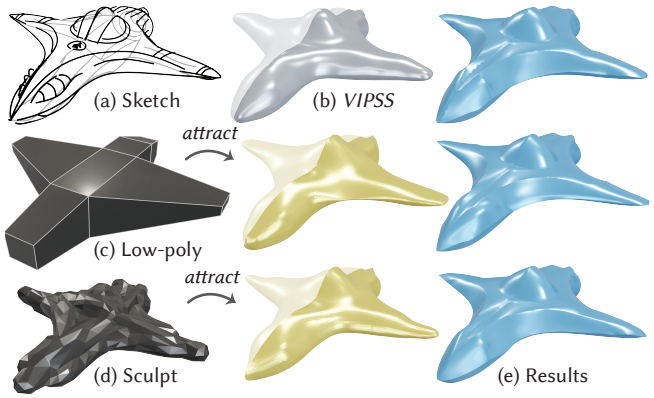


Fig. 5. For the same sketch, we demonstrate how a proxy mesh can be created with *VIPSS* (b), low-poly modeling with *Blender* [Blender 2020] (c), and rough VR sculpting in *Adobe Medium* [Adobe 2020] (d). We automatically improve the proxy mesh by remeshing, smoothing, and attracting it to the strokes (middle column). When the sketch is symmetric, we only process half of the proxy mesh. Note that despite a drastically different original appearance, all three proxy meshes yield a similar result (e), faithful to the input sketch (a).

Symmetric sketches. Many VR sketching and sketch-based modeling tools provide a mirror plane to ease the creation of symmetric sketches. When this is the case, we only surface half of the sketch and mirror the result, which speeds up computation and yields surfaces that are symmetric by construction.

Open surfaces. We support open surfaces by asking users to annotate strokes that denote the boundary of the envisioned surface (colored in blue in all input sketches), and by trimming the surface mesh along these strokes. We provide additional details about this procedure in Appendix D.

User control. Most results shown in this paper were obtained automatically. Nevertheless, an additional strength of our formulation is that it admits user control naturally. Fig. 6 illustrates two scribble-based controls that we offer users to refine the segmentation:

— *Inserting a new region.* Users can scribble over a part of the mesh to trigger the insertion of a new region in that part (Figure 6, yellow scribbles). This interaction is particularly useful to recover details that might have been over-smoothed.
— *Merging regions.* Users can scribble over multiple regions to merge them into a single one (Figure 6, green scribbles). This interaction can be used to correct for over-segmentation.

We implement region insertion by introducing a new model, which we fit to the stroke points associated with the scribbled-on mesh vertices. We implement region merging in a similar manner, except that we fit the new model to the stroke points associated with every vertex that shares a label with any scribbled-on vertex and lies on the same connected component. For both edits, we also penalize the use of other models by adding a term to Equation 1:

$$E_{\text{penalty}}(l) = \rho \sum_{v \in \mathcal{V}_{\text{scribbled}}} 1 - \delta\left(l(v), l_{\text{new}}\right) \tag{10}$$

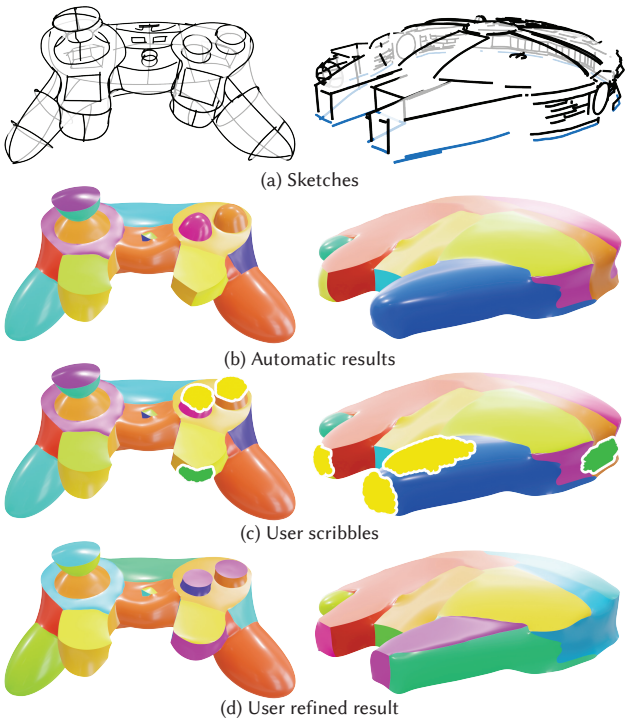


Fig. 6. Our automatic method might produce over-segmented regions or miss geometric details (b). The user can indicate desired changes by scribbling on a preview of the result (c). Yellow scribbles trigger region insertion, green scribbles trigger region merging. Encouraging the segmentation to respect these constraints yields the desired result (d). Left column sketch ©Jacopo Colò.

where $\mathcal{V}_{\text{scribbled}}$ denotes the set of scribbled vertices, l_{new} denotes the label of the new model, and the penalty ρ is set to $(10\epsilon)^2$, with ϵ from Section 4.1. We then update the solution by re-running algorithm 1 from step (1) until convergence.

This combination of new model proposals and a penalties on other models effectively pushes the optimization towards a different local minimum that satisfies the user indications. We describe an edge-collapse algorithm in Appendix C that reduces the complexity of the mesh on which we compute the labeling, making our method react faster to user edits.

6 EVALUATION AND RESULTS

6.1 Algorithm evaluation

Initialization strategy. We initialize our method by over-segmenting the proxy mesh using spectral clustering (Section 4.2). Fig. 7 compares this strategy with two baselines, over multiple runs of the method. The first baseline strategy creates the initial surface models by sampling random sets of points among all stroke points in the sketch. The second baseline applies spectral clustering but keeps the default regularization weights rather than scaling their values during the initial surface fitting.

This evaluation reveals that our strategy is more stable, as it makes the optimization converge to approximately the same energy for

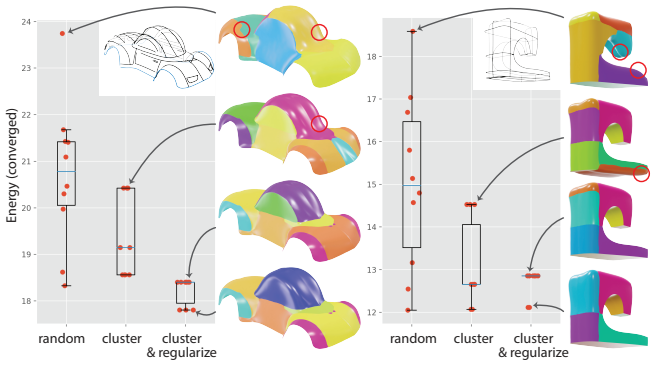


Fig. 7. Comparison of different initialization strategies. Each plot shows the energy (Equation 1) of the converged state over 10 runs with different random seeds. Initializing the surface models with a random selection of stroke points yields high variance across runs (left column of each plot). Spectral clustering identifies local patches that form good candidate models, but the default regularization weights used to fit these models can result in complex surfaces that cover large portions of the sketch (middle column of each plot, red circles in segmentation results). Boosting the regularization weights favors simpler models at the start of the optimization, yielding better and more stable results after convergence (right column of each plot).

different random seeds. In addition, our strategy is more effective, as evidenced by the lower or comparable median energy values at convergence. Finally, visualizing the segmentation produced by each strategy highlights the benefit of boosting the regularization when fitting the initial models, as it prevents the creation of complex models covering large regions of the sketch, which would be difficult to remove in subsequent steps of the optimization.

While our initialization strategies guarantees low variance and similar results regardless of random seed (see bottom two inset results), for all subsequent results we reduce the influence of initialization by running our method 10 times and selecting the solution with the lowest energy.

Segmentation parameters. Fig. 8 shows the effect of our simplicity term (Equation 1) by varying its weight. The result is over-segmented when we omit the simplicity term altogether (Fig. 8a). Increasing this parameter (Fig. 8b, c) leads to smaller model counts m, and to models of lower degrees (see the spaceship cockpit), at the cost of an increase in mean fitting residual $\bar{\epsilon}$. The simplicity energy also encourages using the same model across disconnected regions (Fig. 8c, yellow ellipsoid), which cannot be achieved with the smoothness energy alone.

The other tunable parameter of our method is the factor ϵ that defines the scale of the fitting error we tolerate (Equation 2). Increasing this factor leads to results that follow the strokes more loosely, which can be useful if the input sketch is imprecise and needs to be smoothed (Fig. 9).

Influence of proxy mesh. Fig. 10 illustrates the influence of the proxy resolution and shape on our results. Since the proxy serves as a discrete domain for our segmentation algorithm, its resolution impacts the size of the regions we can detect. At low resolution, neighboring strokes will be lumped together and sharp features will be misplaced

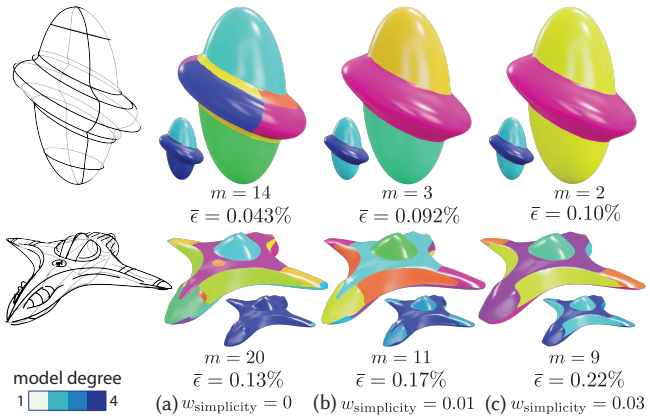


Fig. 8. We demonstrate the impact of the simplicity energy by varying the weight $w_{\text{simplicity}}$ in Equation 1 on two sketches. We measure the number m of models used in the segmentation, and the mean deviation $\bar{\epsilon}$ between stroke points and the models of the vertices they are associated with, as a % of the bounding box diagonal.

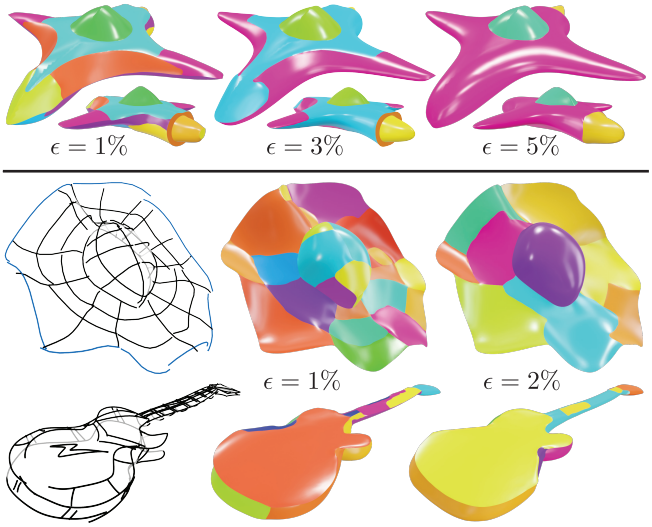


Fig. 9. We demonstrate the impact of the fidelity energy by varying the error scale factor ϵ in Equation 2 on the spaceship sketch (top row). In bottom rows, we show concrete examples of sketches with imprecise strokes where a higher ϵ gives better results compared to the default $\epsilon = 1\%$ parameter. The error scale factor is measured in % of the sketch bounding box diagonal.

(Fig. 10a, third row). In addition, a low-resolution mesh might lack degrees of freedom to follow the curvature of the implicit surfaces, and is thus unable to reproduce the shape of smooth high curvature parts of the sketch (Fig. 10a, fourth row).

The proxy also serves to embed the strokes into a manifold domain representative of the envisioned surface. When the proxy is too far from the desired surface, strokes that should be disjoint might end up projecting to the same location on the proxy and be approximated by the same surface patch (Fig. 10b, top). A slightly more precise proxy mesh resolves the issue (Fig. 10b, bottom).

(a) Resolution (b) Geometry

Fig. 10. Our method depends on the resolution of the input proxy surface (a) as well as on its shape (b). The proxy should have sufficient resolution to capture fine details, and be close enough to the envisioned surface to avoid thin parts of the sketch to collapse. Left column sketch ©Arturo Tolentino.

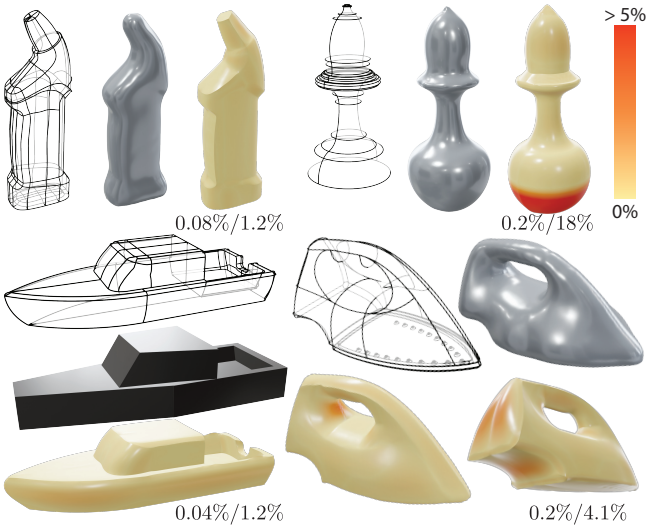


Fig. 11. We evaluate our method on synthetic sketches extracted from ground truth surfaces using *FlowRep* [Gori et al. 2017] (bottle, boat) or from the boundary representation of CAD models (iron, bishop). The deviation is measured as a % of the sketch bounding-box diagonal, and is low almost everywhere except in places where the sketch lacks strokes to disambiguate the intended surface (interior of the iron handle, bottom of the bishop). We provide the median and maximum deviation for each sketch.

Table 1. For each sketch in the paper, we provide the runtimes for the initialization **(1)** – which corresponds to the strokes-proxy association (Appendix A) and step 0 of algorithm 1; for the iterative segmentation **(2)**, and for the final mesh optimization **(3)**. We also give the number of iterations of algorithm 1 necessary to converge to a stable solution.

Sketch name	(1)	(2)	(3)	It.	In paper
robbins_car1	46.3s	11.4s	68.3s	8	Fig. 1
author1_car	16.7s	1.9 s	41.4s	5	Fig. 2
flowrep_boat	17.1s	1.6 s	45.7s	3	Fig. 11
flowrep_bottle	17.6s	2.2 s	20 s	3	Fig. 11
onshape_bishop	9.2 s	1.2 s	14.8s	4	Fig. 11
onshape_iron	20.6s	5.6 s	25.8s	6	Fig. 11
flowsurf_beetle	14.6s	0.8 s	31.2s	2	Fig. 12
ils_roadster	9.3 s	1.5 s	35.3s	4	Fig. 12
t2f_car_97	18.2s	1.6 s	18.7s	3	Fig. 16
author2_guitar	16.2s	1.6 s	26.8s	3	Fig. 16
robbins_car2	62.3s	9.5 s	64.5s	6	Fig. 16
author1_building	32.9s	4.8 s	41.9s	4	Fig. 16
ils_speaker	16.9s	1.2 s	23.5s	3	Fig. 16
swh_vr_controller	146.4s	79.1s	35 s	8	Fig. 16
author1_bulbasaur	11.1s	3.7 s	23.2s	5	Fig. 17
cassie_hat	17.7s	1.2 s	60.3s	3	Fig. 17
tolentino_shoe	57 s	8.2 s	38.8s	5	Fig. 17

6.2 Results and comparisons

Fig. 16 and Fig. 17 provide a gallery of results obtained by surfacing 3D sketches from varied sources, ranging from clean curve networks created with 2D sketch-based modeling interfaces [Bae et al. 2008; Xu et al. 2014], to imprecise and over-sketched stroke clouds created with 2D and VR ideation tools [Colò 2021; Kim and Bae 2016; Yu et al. 2021a]. All these results were obtained without user correction. We use a light gray material to shade proxy meshes computed automatically with *VIPSS* [Huang et al. 2019], and a dark gray material for the ones created manually. Note how our method recovers the fine details and sharp edges depicted in the sketch even from very smooth, approximate proxy meshes. Our supplementary website displays each result with an interactive 3D viewer.

We detail in Table 1 the performance of our algorithm on all results shown in the main paper. Timings vary from a few seconds on simple sketches up to a few minutes on complex sketches composed of many strokes. The bottlenecks reside in the initialization of the algorithm and in the final mesh optimization, while the segmentation algorithm takes less than 10 seconds in most cases. Users can thus refine the segmentation multiple times with relatively short wait times and only compute the final mesh once satisfied.

Comparison against ground truth surfaces. We evaluate our method quantitatively by comparing our results to ground truth surfaces for which curve networks are available. We obtain such data from *FlowRep* [Gori et al. 2017], a method to generate descriptive curve networks from 3D meshes, as well as from CAD models for which we extract the boundary representation (B-rep) [Koch et al. 2019]. Fig. 11 visualizes the results of this experiment, where the color map indicates that our results are very close to ground truth except in ambiguous regions devoid of strokes.

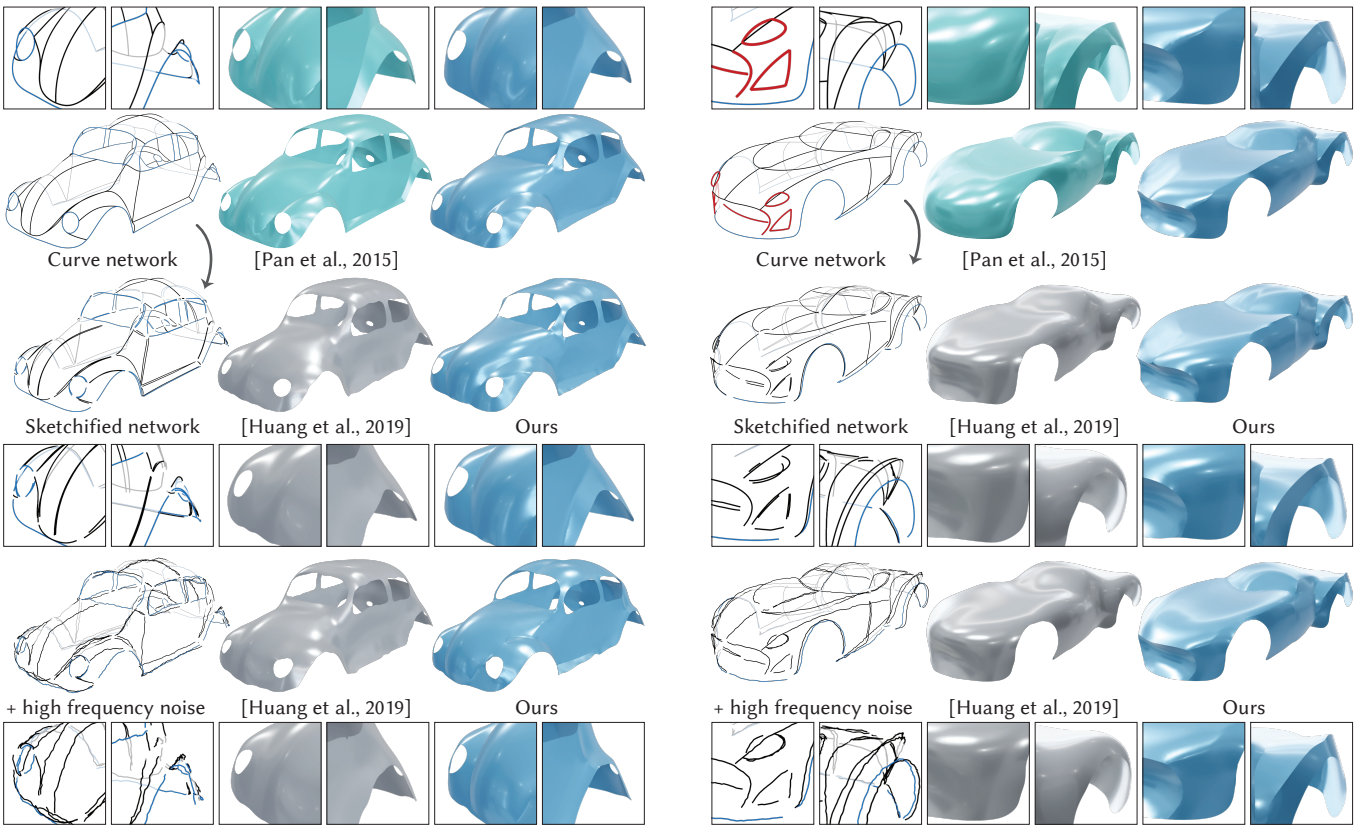


Fig. 12. Our method can surface well-connected curve networks, similarly to [Pan et al. 2015]. More importantly, it also achieves similar results from imprecise sketches that contain duplicate strokes, gaps (middle row) or high-frequency noise (bottom row). Our method also accounts for disconnected parts, like the red stroke that depicts a concavity on the car, which Pan et al. [2015] ignored.

Comparison against curve network surfacing. Fig. 12 compares our method to the one by Pan et al. [2015], which is the most recent method for surfacing well-connected curve networks. The two methods produce very similar results on the original connected network. However, a unique strength of our method is to also produce similar results from disconnected, noisy sketches, which we synthesized by perturbing the original network (displacing and duplicating strokes, introducing gaps). Our method also accounts for strokes that are not connected to the main network, which Pan et al. [2015] had to ignore (Fig. 12, red strokes at the front of the car).

In Fig. 13, we compare to Pan et al. [2015] on a more complex input. By balancing fidelity with smoothness and simplicity, our method tends to miss small details, such as the buttons on top of the machine. Our smooth surface models also do not capture well the generalized cylinder that forms the nozzle. In contrast, by assuming that the input strokes form a clean curve network, Pan et al. can trust every curve to create interpolating surfaces and can leverage connectivity information to detect which curves are sharp features.

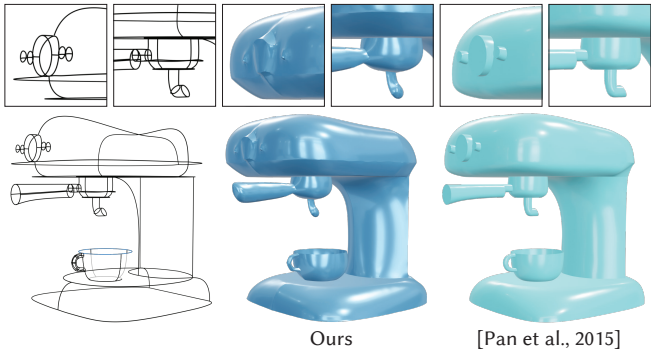
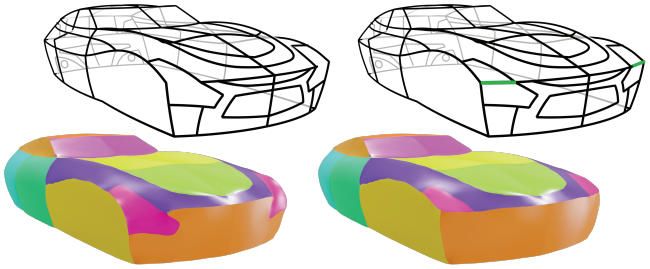


Fig. 13. Our method tends to miss small yet important details (middle) in an effort to produce smooth, simple outputs robust to imprecise drawings. The method by Pan et al. [2015] does not face this challenge as they consider that all input curves are drawn precisely, and as such should be kept in the output. Similar to Pan et al. [2015], we surface the coffee cup separately from the coffee machine.

6.3 Limitations

Relying on geometric criteria only. We emphasize that our method relies purely on geometric criteria to place sharp features in the final surface. As a consequence, our method can miss semantically-important features if they do not contribute significantly to the shape, such as the round headlights of the car in Fig. 1.



(a) No stroke at sharp feature (b) Adding a stroke

Fig. 14. Our method cannot take higher-level cues into account, and will thus fail to align with user intent if a critical sharp feature stroke is missing, as on the car headlight (a). Adding a stroke and re-running the method successfully recovers that feature (b).

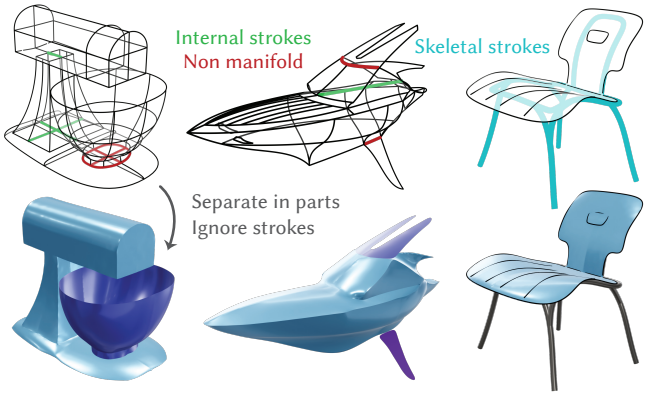


Fig. 15. Our current implementation does not automatically support non-manifold surfaces, as present on the boat and blender (red strokes). Furthermore, our assumption that strokes lie on the surface they depict is not always true, as there may be some strokes inside the outermost depicted surface (green) or strokes that depict tubular structures (cyan). A workaround is to manually separate the sketch into parts and surface them individually, or mark out some strokes to be ignored by our method (second row).

Sharp features not depicted by the strokes. Our algorithm assumes that all sharp features of the intended surface occur along some of the input strokes. Fig. 14 illustrates a limitation of this assumption, where the artist implicitly indicated that the headlight presents a sharp feature by sharp corners in neighboring strokes. In such cases, users need to draw a few additional strokes to obtain the intended surface (Fig. 14b). While our assumption that sharp features are explicitly represented seems to hold in our dataset, further analysis of how people draw in 3D is needed to quantitatively evaluate this hypothesis. In the future, more global cues such as stroke tangent continuity or sketch symmetry could be leveraged to make surfacing better aligned with viewer perception.

Other types of strokes. Since we assume that all strokes lie on the intended surface, our method cannot handle strokes that lie inside the shape, or that depict the skeleton of a tubular structure (Fig. 15). While we simply discarded such strokes to produce our results (see supplemental materials for all original input sketches), future work could attempt to identify these strokes and surface them with dedicated representations, such as generalized cylinders for skeletal stokes [Zhou et al. 2015].

Non manifold configurations. In theory, our method could recover non-manifold surfaces if provided with a non-manifold proxy mesh. In practice, we only surfaced manifold shapes because we implemented our algorithm using a manifold data-structure to represent the mesh. As a work-around, it is typically possible to manually separate the sketch into multiple manifold pieces that can be surfaced separately (Fig. 15, bottom row and coffee cup in Fig. 13). An exciting direction for future work would be to perform topology analysis of the unstructured sketch to detect and process non-manifold parts automatically.

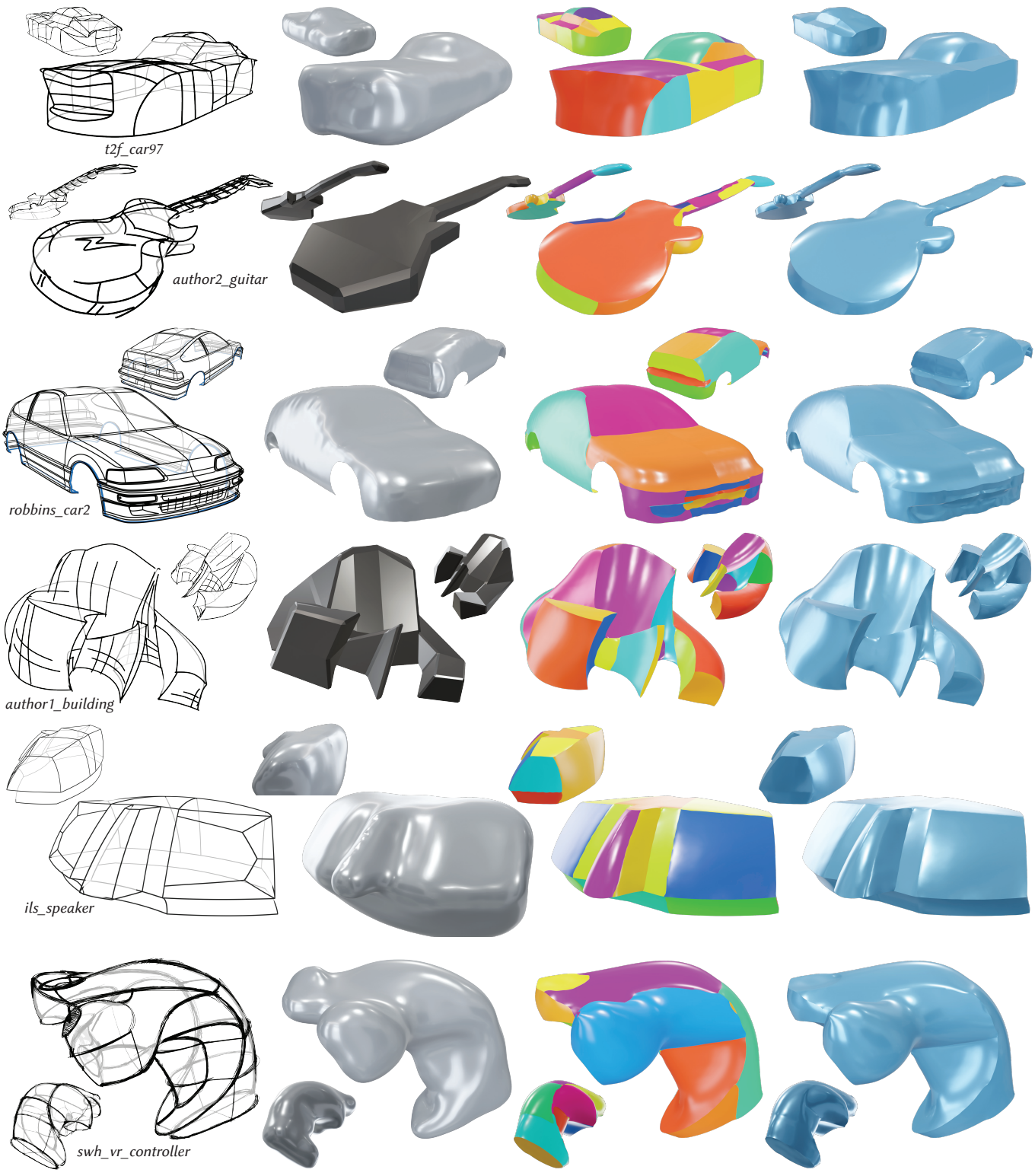
Iterative sketch creation and surfacing. We did not consider the problem of surfacing a sketch as it is being created and refined live by an artist. While our method can be made more efficient to reduce computation time with more engineering effort, it is not trivial how to take into account new strokes or edits on an existing stroke without globally affecting the surface. Moreover, partial sketches introduce more ambiguity to the surfacing problem since some regions may be altogether undefined at a given stage of the process.

7 CONCLUSION

After decades of research, a number of robust algorithms now exist to surface dense point clouds [Berger et al. 2017]. In contrast, very few methods have been proposed to surface the sparse *stroke clouds* that designers produce when sketching in VR or with sketch-based modeling systems. Inspired by the unique characteristics of this emerging form of 3D data, we have proposed an approach to locate smooth patches in unstructured 3D sketches, and to optimize their geometry to produce a piecewise-smooth surface aligned with salient strokes. The resulting 3D meshes can benefit numerous tasks. For instance, we used them to occlude hidden strokes in all figures of this paper, which helps perceive the correct shape from the sketch via relative depth cues [Arora et al. 2017]. Additionally, our sketch-aligned surfaces are compatible with all downstream 3D processing and modeling tasks. This opens exciting avenues to integrate 3D sketching with other creation modalities such as sculpting [Adobe 2020; Blender 2020; Pixologic 2016]. As we have demonstrated, sculpting can be used to create a rough proxy surface, that is then automatically refined by our method to align with a sketch. It is easy to further refine the surface by sculpting, and in the future we would like to investigate tighter coupling between surface and curve-based editing methods, from both interaction design and geometric representation perspectives.

ACKNOWLEDGMENTS

We thank the artists who kindly shared some of their artworks with us, James Robbins, Jacopo Colò, Arturo Tolentino. We thank the anonymous reviewers for their helpful comments. This work was supported by ERC Starting Grant D3 (ERC-2016-STG 714221), NSERC, Advokat Bent Thorbergs Fond (66.531) and by software and research donations from Adobe.



(a) Input sketch (b) Proxy surface (c) Result

Fig. 16. Results of our method applied to a variety of 3D sketches (a). We display proxy meshes computed with *VIPSS* in light gray, and the ones created manually in dark gray (b). For each sketch, we show the segmented patches with random colors, and the final surface in blue (c). Sketch *robbins_car2* ©James Robbins.

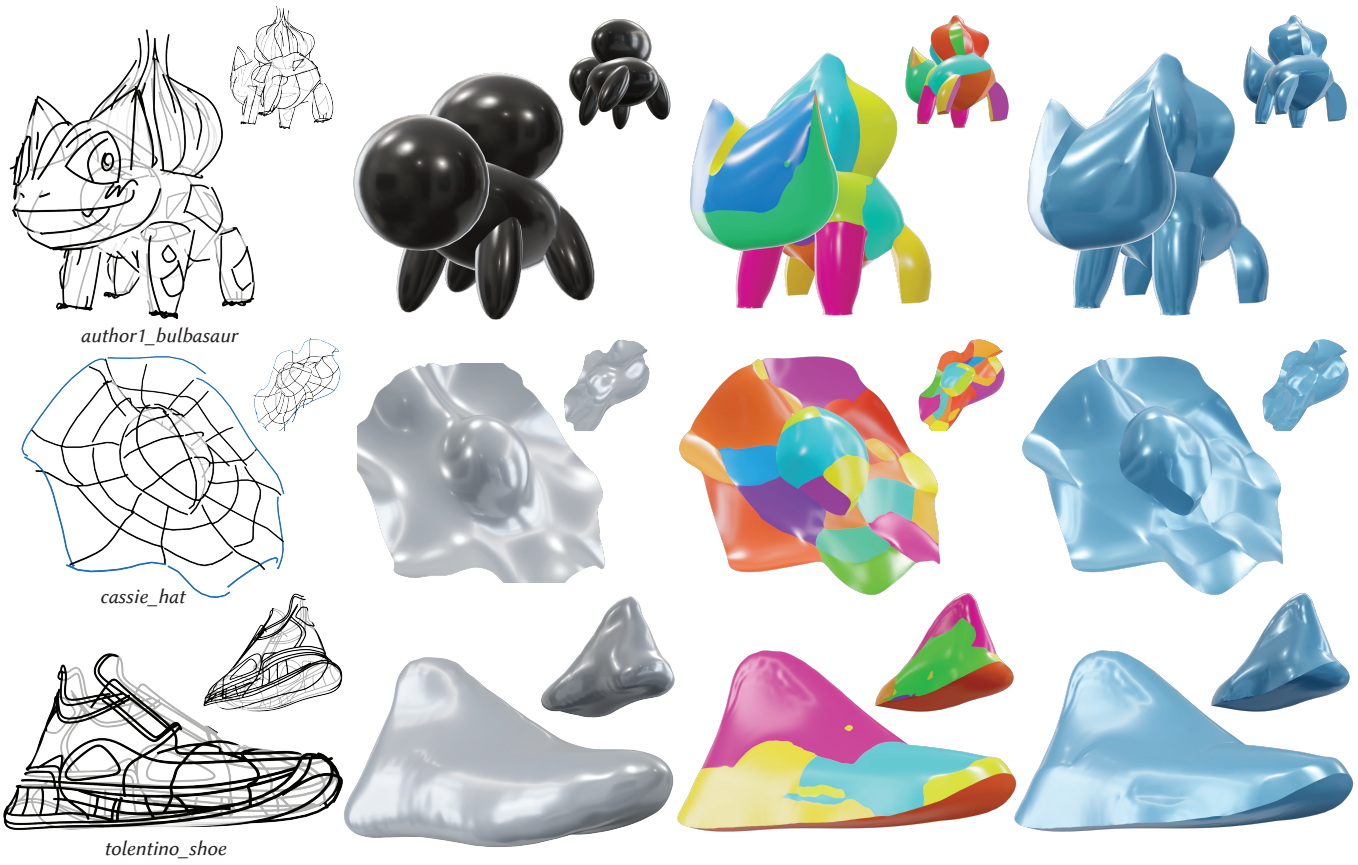


Fig. 17. Results of our method applied to sketches of organic shapes (a). We display proxy meshes computed with *VIPSS* in light gray, and the ones created manually in dark gray (b). For each sketch, we show the segmented patches with random colors, and the final surface in blue (c). Sketch *tolentino_shoe* ©Arturo Tolentino.

REFERENCES

Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta. 2011. Surface patches from unorganized space curves. In *Computer Graphics Forum*, Vol. 30.

Adobe. 2020. Adobe Medium. https://www.adobe.com/products/medium.html.

Sung Joon Ahn, W. Rauh, Hyung Suck Cho, and H.-J. Warnecke. 2002. Orthogonal distance fitting of implicit curves and surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 5 (2002).

Rahul Arora, Rubaiat Habib Kazi, Fraser Anderson, Tovi Grossman, Karan Singh, and George W Fitzmaurice. 2017. Experimental Evaluation of Sketching on Surfaces in VR. In *ACM Conference on Human Factors in Computing Systems (CHI)*, Vol. 17.

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *ACM Symposium on User Interface Software and Technology (UIST)*.

Erhan Batuhan Arisoy, Gunay Orbay, and Levent Burak Kara. 2012. Free form surface skinning of 3d curve clouds for conceptual shape design. *Journal of computing and information science in engineering* 12, 3 (2012).

Jean-Philippe Bauchet and Florent Lafarge. 2020. Kinetic shape reconstruction. *ACM Transactions on Graphics* 39, 5 (2020).

Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2017. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, Vol. 36.

Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. 2012. Design-driven quadrangulation of closed 3d curves. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 31, 6 (2012).

Blender. 2020. Blender. https://www.blender.org/.

Yuri Boykov, Olga Veksler, and Ramin Zabih. 2001. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence* 23, 11 (2001).

Yuri Y. Boykov and Marie-Pierre Jolly. 2001. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *IEEE International Conference on Computer Vision*.

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and Robust Mesh Arrangements using Floating-point Arithmetic. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 39, 6 (2020).

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*. 905–914.

Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. 2008. Where do people draw lines? In *ACM Transactions on Graphics (Proc. SIGGRAPH)*.

Jacopo Colò. 2021. Penzil. https://www.penzil.app/.

Andrew Delong, Anton Osokin, Hossam N Isack, and Yuri Boykov. 2012. Fast approximate energy minimization with label costs. *International journal of computer vision* 96, 1 (2012).

Xingyi Du, Qingnan Zhou, Nathan Carr, and Tao Ju. 2021. Boundary-Sampled Halfspaces: A New Representation for Constructive Solid Modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 40, 4 (2021).

Marek Dvorožňák, Saman Sepehri Nejad, Ondřej Jamriška, Alec Jacobson, Ladislav Kavan, and Daniel Sýkora. 2018. Seamless Reconstruction of Part-Based High-Relief Models from Hand-Drawn Images. In *Proceedings of International Symposium on Sketch-Based Interfaces and Modeling*. Article 5.

Ricardo Fabbri and Benjamin B. Kimia. 2010. 3D Curve Sketch: Flexible Curve-Based Stereo Reconstruction and Calibration. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Shachar Fleishman, Daniel Cohen-Or, and Cláudio T Silva. 2005. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics* 24, 3 (2005).
Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Annual conference on computer graphics and interactive techniques (SIGGRAPH)*. 209–216.
Google. 2016. Tilt Brush. https://www.tiltbrush.com.
Giorgio Gori, Alla Sheffer, Nicholas Vining, Enrique Rosales, Nathan Carr, and Tao Ju. 2017. FlowRep: Descriptive Curve Networks for Free-Form Design Shapes. *ACM Transaction on Graphics (Proc. SIGGRAPH)* 36, 4 (2017).
GravitySketch. 2017. Gravity Sketch. https://www.gravitysketch.com/.
Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia C Pont, Frédo Durand, and Adrien Bousseau. 2019. OpenSketch: a richly-annotated dataset of product design sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 38, 6 (2019).
Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1992. Surface reconstruction from unorganized points. In *Annual conference on computer graphics and interactive techniques (SIGGRAPH)*. 71–78.
Alexander Hornung and Leif Kobbelt. 2006. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Symposium on geometry processing*. 41–50.
H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. Zhang. 2013. Edge-Aware Point Set Resampling. *ACM Transactions on Graphics* 32 (2013). Issue 1.
Zhiyang Huang, Nathan Carr, and Tao Ju. 2019. Variational implicit point set surfaces. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 38, 4 (2019).
Hossam Isack and Yuri Boykov. 2012. Energy-based geometric multi-model fitting. *International journal of computer vision* 97, 2 (2012), 123–147.
Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-Aligned Meshes. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 34, 6 (Nov. 2015).
Philipp Jenke, Michael Wand, Wolfgang Straßer, and A AKA. 2008. Patch-Graph Reconstruction for Piecewise Smooth Surfaces. In *VMV*. Citeseer, 3–12.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.
Yongkwan Kim and Seok-Hyung Bae. 2016. SketchingWithHands: 3D sketching handheld products with first-person hand posture. In *ACM Symposium on User Interface Software and Technology (UIST)*.
Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. Abc: A big cad model dataset for geometric deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition*.
Changjian Li, Hao Pan, Yang Liu, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces Through 2D Sketching. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 36, 4 (2017).
Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. 2011. Globfit: Consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*.
Mayra D Barrera Machuca, Paul Asente, Wolfgang Stuerzlinger, Jingwan Lu, and Byungmoon Kim. 2018. Multiplanes: Assisted freehand VR Sketching. In *Proceedings of the Symposium on Spatial User Interaction*. ACM.
Mayra Donaji Barrera Machuca, Wolfgang Stuerzlinger, and Paul Asente. 2019. The Effect of Spatial Ability on Immersive 3D Drawing. In *Proceedings of the ACM Conference on Creativity & Cognition (C&C'19)*.
Aron Monszpart, Nicolas Mellado, Gabriel J Brostow, and Niloy J Mitra. 2015. RAPter: rebuilding man-made scenes with regular arrangements of planes. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 34, 4 (2015).
Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2006. Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*. 381–389.
Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 26, 3 (2007).
GüNay Orbay and Levent Burak Kara. 2012. Sketch-based surface design using malleable curve networks. *Computers & Graphics* 36, 8 (2012), 916–929.
Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang. 2015. Flow aligned surfacing of curve networks. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 34, 4 (2015).
Trung T. Pham, Markus Eich, Ian Reid, and Gordon Wyeth. 2016. Geometrically Consistent Plane Extraction for Dense Indoor 3D Maps Segmentation. In *IROS*.
Pixologic. 2016. ZBrush. http://pixologic.com/features/about-zbrush.php.
Enrique Rosales, Chrystiano Araújo, Jafet Rodriguez, Nicholas Vining, Dongwook Yoon, and Alla Sheffer. 2021. AdaptiBrush: Adaptive General and Predictable VR Ribbon Brush. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)* 40, 1 (2021).
Enrique Rosales, Jafet Rodriguez, and Alla Sheffer. 2019. SurfaceBrush: from virtual reality drawings to manifold surfaces. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 38, 4 (2019).
Bardia Sadri and Karan Singh. 2014. Flow-complex-based shape reconstruction from 3d curves. *ACM Transactions on Graphics (TOG)* 33, 2 (2014).
Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic drawing of 3D scaffolds. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 28, 5 (2009).
Ruwen Schnabel, Patrick Degener, and Reinhard Klein. 2009. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, Vol. 28.
Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000).
Smoothstep. 2021. Quill. https://quill.art/.
Olga Sorkine and Daniel Cohen-Or. 2004. Least-squares meshes. In *Proceedings Shape Modeling Applications, 2004*. IEEE, 191–199.
Tibor Stanko, Stefanie Hahmann, Georges-Pierre Bonneau, and Nathalie Saguin-Sprynski. 2016. Smooth interpolation of curve networks with surface normals. In *Eurographics 2016 Short Papers*. Eurographics Association, 21–24.
Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. 2009. Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 papers*. 1–9.
T. Tasdizen, J.-P. Tarel, and D.B. Cooper. 2000. Improving the stability of algebraic curves for applications. *IEEE Transactions on Image Processing* 9, 3 (2000).
Gabriel Taubin. 1993. An improved algorithm for algebraic curve and surface fitting. In *1993 (4th) International Conference on Computer Vision*. IEEE, 658–665.
Anil Usumezbas, Ricardo Fabbri, and Benjamin B. Kimia. 2017. The Surfacing of Multiview 3D Drawings via Lofting and Occlusion Reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition*.
Jianhua Wu and Leif Kobbelt. 2005. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum* 24, 3 (2005).
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 Transactions on Graphics (Proc. SIGGRAPH)* 33, 4 (2014).
Dong-Ming Yan, Wenping Wang, Yang Liu, and Zhouwang Yang. 2012. Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design* 44, 11 (2012).
Emilie Yu, Rahul Arora, Tibor Stanko, J Andreas Bærentzen, Karan Singh, and Adrien Bousseau. 2021a. CASSIE: Curve and Surface Sketching in Immersive Environments. In *ACM Conference on Human Factors in Computing Systems (CHI)*. 1–14.
Xue Yu, Stephen DiVerdi, Akshay Sharma, and Yotam Gingold. 2021b. ScaffoldSketch: Accurate Industrial Design Drawing in VR. In *ACM Symposium on User Interface Software and Technology (UIST)*.
Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or. 2015. Generalized Cylinder Decomposition. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 34, 6 (2015).
Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. 2013. A general and efficient method for finding cycles in 3D curve networks. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 32, 6 (2013).

A ASSOCIATING 3D STROKES WITH THE PROXY MESH

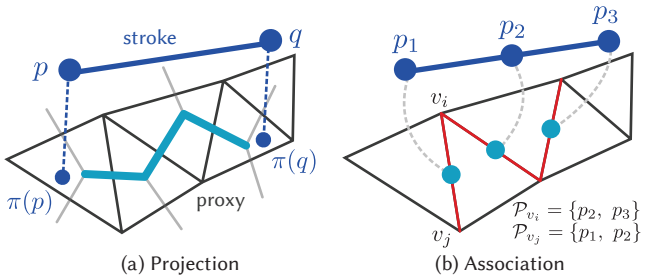
We provide additional implementation details on how we associate stroke points $p \in \mathcal{P}$ and proxy mesh vertices $v \in \mathcal{V}$.

We start by projecting each stroke point to its closest face on the proxy mesh. We then trace out stroke segments as geodesics lying on the mesh surface. Note that we do not need to trace precise geodesics on the mesh, since we are only interested in detecting which mesh edges are crossed by a given stroke segment. Therefore, we approximate the geodesics with shortest paths in the dual graph of the mesh (light blue path in Fig. 18a). This gives us the list of primal mesh edges that are crossed by the projected stroke segment.

We can then associate stroke points to these crossed edges (Fig. 18, red) by sampling additional points on the stroke segment (Fig. 18, blue). If n edges are crossed while tracing a segment, we sample n regularly-spaced points on that segment. Each of these points is then associated to the corresponding mesh edge.

Finally, we associate to each vertex v_i the stroke points associated with each edge of its 1-ring. If there are multiple stroke points associated to a single edge, we only associate the closest point to v_i.

The projection of stroke segments onto the proxy also serves to define the edge weights of the graph, since we assign a low weight to edges crossed by a projected segment – red edges in Fig. 18b.



(a) Projection (b) Association

Fig. 18. Associating stroke information to the proxy graph. (a) Stroke points are projected onto the mesh and we find the shortest path (light blue) between the mesh faces on the dual graph of the mesh (in grey). (b) The shortest path yields a list of edges crossed by the projected segment (red). We associate stroke points sampled from the stroke segment with neighboring vertices of each crossed edge.

B REGULARIZED FITTING OF IMPLICIT POLYNOMIALS

From Equation 6, we obtain the parameters $\hat{\boldsymbol{\theta}}_\alpha$ that best fit the stroke points by solving the linear system

$$(M^T M + \mu M_{\text{align}}^T M_{\text{align}} + \lambda I)\ \hat{\boldsymbol{\theta}}_\alpha = \mu M_{\text{align}}^T. \tag{11}$$

M is a $(|\mathcal{P}_\alpha| \times t)$ matrix composed of rows of the monomial vector $X(\mathbf{p})$ for every point in $\mathcal{P}_\alpha$. And given the set of points $\mathcal{P}'_\alpha$ for the alignment term, M_{align} is a $(1 \times t)$ vector defined as:

$$M_{\text{align}} = \sum_{\mathbf{p} \in \mathcal{P}'_\alpha} \mathbf{n_p}^T \nabla \mathbf{X}(\mathbf{p}). \tag{12}$$

To keep the L^2 regularization position and scale independent, and resilient to non uniform scaling in the point set $\mathcal{P}_\alpha$, we apply a transformation to the space of monomials composed of a translation – to center the data – and a non uniform scaling – to normalize it. In practice, we compute the column-wise mean values μ and standard deviations S of M, and consider the transformed polynomial function:

$$f(\boldsymbol{\theta}_\alpha; x, y, z) = \left(\frac{\mathbf{X}(x, y, z) - \mu}{S}\right)^T \hat{\boldsymbol{\theta}}_\alpha,$$

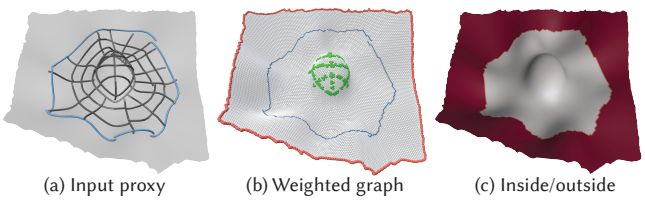
where the division by the vector S is a column-wise division. As a consequence, the vectors $X(\mathbf{p})$ that compose matrix M are transformed by subtracting μ and dividing by S, and the vector M_{align} is divided by S.

C GRAPH SIMPLIFICATION

Algorithm 1 has a complexity that increases linearly with the number of nodes in the graph to be labeled [Boykov et al. 2001]. Yet, we note that only the vertices that are associated to stroke points contribute to E_{fidelity}, other vertices being solely determined by the smoothness and simplicity terms. We leverage this observation to simplify the graph away from the strokes, which we achieve by performing greedy edge collapses [Garland and Heckbert 1997; Hoppe et al. 1992]. In our context, we are not interested in preserving the 3D shape of the original mesh, but rather its connectivity. Therefore, we prioritize collapses that yield vertices of low valence, and stop collapsing edges whenever any further collapse would yield a vertex of high valence (> 12 in practice). We exclude edges that have a vertex associated with stroke points to preserve high resolution near the strokes.

This simplification produces a mesh with spatially-varying resolution, which we account for by adjusting the edge weight to $\tilde{w}^e = w^e c^e$, where w^e is the original edge weight as defined in Section 4.1, and c^e scales this weight according to the size of the triangles adjacent to the edge, as measured by the Euclidean distance between the vertices opposite the edge on the two adjacent faces.

After running algorithm 1 on the simplified graph, we propagate the labeling to the vertices of the original graph based on proximity. This simple strategy speeds up Algorithm 1 by a factor of 2.9 on average, while yielding qualitatively similar results.



(a) Input proxy & labeled strokes (b) Weighted graph & inside/outside nodes (c) Inside/outside labeling

Fig. 19. To locate the boundary of an open surface, we ask the user to indicate which strokes correspond to boundaries in the input sketch (a, blue strokes). We perform a graph cut on the mesh to separate inside nodes (b, green) from outside nodes (b, red), while encouraging cuts to happen at edges crossed by boundary strokes (b, blue edges). The resulting labeling separates the inside of the surface (c, white) from the outside (c, red).

D BOUNDARIES OF OPEN MESHES.

While the Laplacian regularization in Equation 9 favors smooth, regular meshes, it has the adversarial effect of shrinking open meshes. We prevent this effect by adding to E_{mesh} a term that constrains boundary vertices of open meshes to stay close to their associated stroke points

$$E_{\text{attract}} = \sum_{v \in \mathcal{V}_{\text{boundary}}} \sum_{\mathbf{p} \in \mathcal{P}_v} ||v - \mathbf{p}||^2 \tag{13}$$

where $\mathcal{V}_{\text{boundary}}$ is the set of vertices associated to the boundary strokes of the sketch.

However, we do not know a priori where an open surface should be trimmed to align with the sketch boundary. We thus ask users to annotate the boundary strokes of sketches depicting open surfaces (shown in blue in all sketches). After projecting these strokes over the proxy mesh, we segment the mesh into an interior and an exterior region separated by the boundary strokes (Fig. 19c), and trim the exterior region to obtain a mesh whose boundary $\mathcal{V}_{\text{boundary}}$ aligns with the boundary strokes. We perform this segmentation with a graph cut, where we encourage cuts along edges crossed by boundary strokes by setting their cost to zero (Fig. 19b). The graph cut source node is linked to all boundary vertices of the input mesh (red in Fig. 19b) and the sink node is linked to vertices of the mesh that have associated stroke points but are geodesically far from the boundary strokes (green in Fig. 19b).

For ease of comparison, we also apply this boundary cutting step to results of Huang et al. [2019] against which we compare.